Package ‘SentimentAnalysis’

August 23, 2023

Type Package
Title Dictionary-Based Sentiment Analysis
Version 1.3-5
Date 2023-08-23

Description Performs a sentiment analysis of textual contents in R. This implementation utilizes various existing dictionaries, such as Harvard IV, or finance-specific dictionaries. Furthermore, it can also create customized dictionaries. The latter uses LASSO regularization as a statistical approach to select relevant terms based on an exogenous response variable.

License MIT + file LICENSE

URL https://github.com/sfeuerriegel/SentimentAnalysis

BugReports https://github.com/sfeuerriegel/SentimentAnalysis/issues

Depends R (>= 2.10)
Imports tm (>= 0.6), qdapDictionaries, ngramrr (>= 0.1), moments, stringdist, glmnet, spikeslab (>= 1.1), ggplot2
Suggests testthat, knitr, rmarkdown, SnowballC, XML, mgcv
LazyData true
Encoding UTF-8
RoxygenNote 7.2.3
VignetteBuilder knitr

NeedsCompilation no

Author Nicolas Proellochs [aut, cre], Stefan Feuerriegel [aut]

Maintainer Nicolas Proellochs <nicolas@nproellochs.com>

Repository CRAN

Date/Publication 2023-08-23 20:10:03 UTC
## R topics documented:

- `analyzeSentiment` .................................................. 3
- `compareDictionaries` ............................................. 6
- `compareToResponse` ................................................ 7
- `convertToBinaryResponse` ....................................... 8
- `convertToDirection` ............................................... 9
- `countWords` ......................................................... 10
- `DictionaryGI` .......................................................... 12
- `DictionaryHE` .......................................................... 12
- `DictionaryLM` ......................................................... 13
- `enetEstimation` ...................................................... 14
- `extractWords` ........................................................ 15
- `generateDictionary` ............................................... 15
- `glmEstimation` ........................................................ 20
- `lassoEstimation` ....................................................... 21
- `lmEstimation` .......................................................... 22
- `loadDictionaryGI` .................................................... 23
- `loadDictionaryHE` .................................................... 23
- `loadDictionaryLM` .................................................... 24
- `loadDictionaryLM_Uncertainty` .................................... 24
- `loadDictionaryQDAP` ................................................ 25
- `loadImdb` ............................................................... 25
- `lookupEstimationMethod` .......................................... 26
- `ngram_tokenize` ....................................................... 26
- `numEntries` ........................................................... 27
- `numNegativeEntries` ................................................ 28
- `numPositiveEntries` ................................................ 29
- `plot.SentimentDictionaryWeighted` .............................. 29
- `plotSentiment` ........................................................ 30
- `plotSentimentResponse` ............................................ 31
- `predict.SentimentDictionaryWeighted` ........................... 32
- `preprocessCorpus` .................................................... 34
- `print.SentimentDictionaryWordlist` .............................. 34
- `read` ................................................................. 35
- `ridgeEstimation` ...................................................... 36
- `ruleLinearModel` .................................................... 37
- `ruleNegativity` ...................................................... 38
- `rulePositivity` ....................................................... 38
- `ruleRatio` ............................................................ 39
- `ruleSentiment` ........................................................ 39
- `ruleSentimentPolarity` ............................................. 40
- `ruleWordCount` ....................................................... 40
- `SentimentDictionary` ................................................. 41
- `SentimentDictionaryBinary` ....................................... 41
- `SentimentDictionaryWeighted` .................................... 42
- `SentimentDictionaryWordlist` .................................... 43
- `spikeslabEstimation` ................................................ 44
analyzeSentiment

**Description**

Performs sentiment analysis of given object (vector of strings, document-term matrix, corpus).

**Usage**

```r
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)
```

```r
## S3 method for class 'Corpus'
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)
```

```r
## S3 method for class 'character'
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)
```
analyzeSentiment

## S3 method for class 'data.frame'
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)

## S3 method for class 'TermDocumentMatrix'
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)

## S3 method for class 'DocumentTermMatrix'
analyzeSentiment(
  x,
  language = "english",
  aggregate = NULL,
  rules = defaultSentimentRules(),
  removeStopwords = TRUE,
  stemming = TRUE,
  ...
)

**Arguments**

- **x**: A vector of characters, a data.frame, an object of type `Corpus`, `TermDocumentMatrix` or `DocumentTermMatrix`
- **language**: Language used for preprocessing operations (default: English)
- **aggregate**: A factor variable by which documents can be grouped. This helpful when joining e.g. news from the same day or move reviews by the same author
- **rules**: A named list containing individual sentiment metrics. Therefore, each entry consists itself of a list with first a method, followed by an optional dictionary.
- **removeStopwords**: Flag indicating whether to remove stopwords or not (default: yes)
- **stemming**: Perform stemming (default: TRUE)
- **...**: Additional parameters passed to function for e.g. preprocessing
analyzeSentiment

Details
This function returns a data.frame with continuous values. If one desires other formats, one needs to convert these. Common examples of such formats are binary response values (positive / negative) or tertiary (positive, neutral, negative). Hence, consider using the functions convertToBinaryResponse and convertToDirection, which can convert a vector of continuous sentiment scores into a factor object.

Value
Result is a matrix with sentiment values for each document across all defined rules

See Also
c ompareToResponse for evaluating the results, convertToBinaryResponse and convertToDirection for getting binary results, generateDictionary for dictionary generation, plotSentiment and plotSentimentResponse for visualization

Examples
## Not run:
library(tm)

# via vector of strings
corpus <- c("Positive text", "Neutral but uncertain text", "Negative text")
sentiment <- analyzeSentiment(corpus)
compareToResponse(sentiment, c(+1, 0, -2))

# via Corpus from tm package
data("crude")
sentiment <- analyzeSentiment(crude)

# via DocumentTermMatrix (with stemmed entries)
dtm <- DocumentTermMatrix(VCorpus(VectorSource(c("posit posit", "negat neutral"))))
sentiment <- analyzeSentiment(dtm)
compareToResponse(sentiment, convertToBinaryResponse(c(+1, -1)))

# By adapting the parameter rules, one can incorporate customized dictionaries
# e.g. in order to adapt to arbitrary languages
dictionaryAmplifiers <- SentimentDictionary(c("more", "much"))
sentiment <- analyzeSentiment(corpus,
   rules=list("Amplifiers"=list(ruleRatio,
                             dictionaryAmplifiers)))

# On can also restrict the number of computed methods to the ones of interest
# in order to achieve performance optimizations
sentiment <- analyzeSentiment(corpus,
   rules=list("SentimentLM"=list(ruleSentiment,
                             loadDictionaryLM())))
sentiment

## End(Not run)
compareDictionaries  

Compares two dictionaries

Description

Routine compares two dictionaries in terms of how similarities and differences. Among the calculated measures are the total of distinct words, the overlap between both dictionaries, etc.

Usage

```r
compareDictionaries(d1, d2)
```

Arguments

- `d1` is the first sentiment dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary` or `SentimentDictionaryWeighted`
- `d2` is the first sentiment dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary` or `SentimentDictionaryWeighted`

Value

Returns list with different metrics depending on dictionary type

Note

Currently, this routine only supports the case where both dictionaries are of the same type

See Also

- `SentimentDictionaryWordlist`, `SentimentDictionaryBinary`, `SentimentDictionaryWeighted`

for the specific classes

Examples

```r
d1 <- SentimentDictionary(c("uncertain", "possible", "likely"))
d2 <- SentimentDictionary(c("rather", "intend", "likely"))
cmp <- compareDictionaries(d1, d2)

d1 <- SentimentDictionary(c("increase", "rise", "more"),
c("fall", "drop"))
d2 <- SentimentDictionary(c("positive", "rise", "more"),
c("negative", "drop"))
cmp <- compareDictionaries(d1, d2)

# Weighted dictionary example

d1 <- SentimentDictionary(c("increase", "decrease", "exit"),
c(+1, -1, -10),
rep(NA, 3))
```
compareToResponse

```r
d2 <- SentimentDictionary(c("increase", "decrease", "drop", "neutral"),
                          c(+2, -5, -1, 0),
                          rep(NA, 4))
cmp <- compareDictionaries(d1, d2)
```

**compareToResponse**  
Compare sentiment values to existing response variable

**Description**

This function compares the calculated sentiment values with an external response variable. Examples of such an exogenous response are stock market movements or IMDb move rating. Both usually reflect a "true" value that the sentiment should match.

**Usage**

```r
compareToResponse(sentiment, response)
```

## S3 method for class 'logical'

```r
compareToResponse(sentiment, response)
```

## S3 method for class 'factor'

```r
compareToResponse(sentiment, response)
```

## S3 method for class 'integer'

```r
compareToResponse(sentiment, response)
```

## S3 method for class 'data.frame'

```r
compareToResponse(sentiment, response)
```

## S3 method for class 'numeric'

```r
compareToResponse(sentiment, response)
```

**Arguments**

- `sentiment`  
  Matrix with sentiment scores for each document across several sentiment rules

- `response`  
  Vector with "true" response. This vector can either be of a continuous numeric or binary values. In case of the latter, FALSE is matched to a negative sentiment value, while TRUE is matched to a non-negative one.

**Value**

Matrix with different performance metrics for all given sentiment rules
convertToBinaryResponse

Examples

```r
sentiment <- matrix(c(5.5, 2.9, 0.9, -1),
dinnames=list(c("A", "B", "C", "D"), c("Sentiment")))

# continuous numeric response variable
response <- c(5, 3, 1, -1)
compareToResponse(sentiment, response)

# binary response variable
response <- c(TRUE, TRUE, FALSE, FALSE)
compareToResponse(sentiment, response)
```

convertToBinaryResponse

Convert continuous sentiment to direction

Description

This function converts continuous sentiment scores into a their corresponding binary sentiment class. As such, the result is a factor with two levels indicating positive and negative content. Neutral documents (with a sentiment score of 0) are counted as positive.

Usage

```r
convertToBinaryResponse(sentiment)
```

Arguments

- `sentiment`: Vector, matrix or data.frame with sentiment scores.

Details

If a matrix or data.frame is provided, this routine does not touch all columns. In fact, it scans for those where the column name starts with "Sentiment" and changes these columns only. Hence, columns with pure negativity, positivity or ratios or word counts are ignored.

Value

If a vector is supplied, it returns a factor with two levels representing positive and negative content. Otherwise, it returns a data.frame with the corresponding columns being exchanged.

See Also

convertToDirection
Examples

sentiment <- c(-1, -0.5, +1, 0.6, 0)
convertToBinaryResponse(sentiment)
convertToDirection(sentiment)

df <- data.frame(No=1:5, Sentiment=sentiment)
df
convertToBinaryResponse(df)
convertToDirection(df)

---

convertToDirection  
*Convert continuous sentiment to direction*

Description

This function converts continuous sentiment scores into a their corresponding sentiment direction. As such, the result is a factor with three levels indicating positive, neutral and negative content. In contrast to `convertToBinaryResponse`, neutral documents have their own category.

Usage

`convertToDirection(sentiment)`

Arguments

sentiment  
Vector, matrix or data.frame with sentiment scores.

Details

If a matrix or data.frame is provided, this routine does not touch all columns. In fact, it scans for those where the column name starts with "Sentiment" and changes these columns only. Hence, columns with pure negativity, positivity or ratios or word counts are ignored.

Value

If a vector is supplied, it returns a factor with three levels representing positive, neutral and negative content. Otherwise, it returns a data.frame with the corresponding columns being exchanged.

See Also

`convertToBinaryResponse`
Examples

```r
countWords <- c(-1, -0.5, +1, 0.6, 0)
convertToBinaryResponse(countWords)
convertToDirection(countWords)

df <- data.frame(No=1:5, Sentiment=countWords)
df
convertToBinaryResponse(df)
convertToDirection(df)
```

### countWords

**Count words**

Function counts the words in each document

#### Usage

```r
countWords(
  x,
  aggregate = NULL,
  removeStopwords = TRUE,
  language = "english",
  ...)
```

## S3 method for class 'Corpus'

```r
countWords(
  x,
  aggregate = NULL,
  removeStopwords = TRUE,
  language = "english",
  ...)
```

## S3 method for class 'character'

```r
countWords(
  x,
  aggregate = NULL,
  removeStopwords = TRUE,
  language = "english",
  ...)
```

## S3 method for class 'data.frame'

```r
countWords(
```
```
x,
aggregate = NULL,
removeStopwords = TRUE,
language = "english",
...
)

## S3 method for class 'TermDocumentMatrix'
countWords(
  x,
  aggregate = NULL,
  removeStopwords = TRUE,
  language = "english",
  ...
)

## S3 method for class 'DocumentTermMatrix'
countWords(
  x,
  aggregate = NULL,
  removeStopwords = TRUE,
  language = "english",
  ...
)

Arguments

x                      A vector of characters, a data.frame, an object of type Corpus, TermDocumentMatrix or DocumentTermMatrix
aggregate              A factor variable by which documents can be grouped. This helpful when joining e.g. news from the same day or movie reviews by the same author
removeStopwords        Flag indicating whether to remove stopwords or not (default: yes)
language               Language used for preprocessing operations (default: English)
...                    Additional parameters passed to function for e.g. preprocessing

Value

Result is a matrix with word counts for each document across

Examples

documents <- c("This is a test", "an one more")

# count words (without stopwords)
countWords(documents)

# count all words (including stopwords)
countWords(documents, removeStopwords=FALSE)
```
DictionaryGI

**Dictionary with opinionated words from the Harvard-IV dictionary as used in the General Inquirer software**

**Description**

Dictionary with a list of positive and negative words according to the psychological Harvard-IV dictionary as used in the General Inquirer software. This is a general-purpose dictionary developed by the Harvard University.

**Usage**

data(DictionaryGI)

**Format**

A list with different terms according to Henry

**Note**

All words are in lower case and non-stemmed

**Source**

https://inquirer.sites.fas.harvard.edu/homecat.htm

**Examples**

data(DictionaryGI)
summary(DictionaryGI)

DictionaryHE

**Dictionary with opinionated words from Henry’s Financial dictionary**

**Description**

Dictionary with a list of positive and negative words according to the Henry’s finance-specific dictionary. This dictionary was first presented in the *Journal of Business Communication* among one of the early adopters of text analysis in the finance discipline.

**Usage**

data(DictionaryHE)

**Format**

A list with different wordlists according to Henry
Note

All words are in lower case and non-stemmed

References


Examples

data(DictionaryHE)
summary(DictionaryHE)

---

DictionaryLM

*Dictionary with opinionated words from Loughran-McDonald Financial dictionary*

Description

Dictionary with a list of positive, negative and uncertainty words according to the Loughran-McDonald finance-specific dictionary. This dictionary was first presented in the *Journal of Finance* and has been widely used in the finance domain ever since.

Usage

data(DictionaryLM)

Format

A list with different terms according to Loughran-McDonald

Note

All words are in lower case and non-stemmed

Source

https://sraf.nd.edu/loughranmcdonald-master-dictionary/

References


Examples

data(DictionaryLM)
summary(DictionaryLM)
enetEstimation

Elastic net estimation

Description

Function estimates coefficients based on elastic net regularization.

Usage

enetEstimation(
  x,
  response,
  control = list(alpha = 0.5, s = "lambda.min", family = "gaussian", grouped = FALSE),
  ...)

Arguments

x An object of type DocumentTermMatrix.
response Response variable including the given gold standard.
control (optional) A list of parameters defining the model as follows:
  • "alpha" Abstraction parameter for switching between LASSO and ridge regularization (with default alpha=0.5). Best option is to loop over this parameter and test different alternatives.
  • "s" Value of the parameter lambda at which the elastic net is evaluated. Default is s="lambda.1se" which takes the calculated minimum value for \( \lambda \) and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by lambda="lambda.min".
  • "family" Distribution for response variable. Default is family="gaussian". For non-negative counts, use family="poisson". For binary variables family="binomial". See glmnet for further details.
  • "grouped" Determines whether grouped function is used (with default FALSE).
...

Value

Result is a list with coefficients, coefficient names and the model intercept.
extractWords

Extract words from dictionary

Description

Returns all entries from a dictionary.

Usage

extractWords(d)

Arguments

d    Dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary` or `SentimentDictionaryWeighted`

Examples

extractWords(SentimentDictionary(c("uncertain", "possible", "likely"))) # returns 3
extractWords(SentimentDictionary(c("increase", "rise", "more"),
    c("fall", "drop"))) # returns 5
extractWords(SentimentDictionary(c("increase", "decrease", "exit"),
    c(+1, -1, -10),
    rep(NA, 3))) # returns 3

generateDictionary

Generates dictionary of decisive terms

Description

Routine applies method for dictionary generation (LASSO, ridge regularization, elastic net, ordinary least squares, generalized linear model or spike-and-slab regression) to the document-term matrix in order to extract decisive terms that have a statistically significant impact on the response variable.

Usage

generateDictionary(
    x,
    response,
    language = "english",
    modelType = "lasso",
    filterTerms = NULL,
    control = list(),
    minWordLength = 3,
    sparsity = 0.9,
weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
...

## S3 method for class 'Corpus'
generateDictionary(
  x,
  response,
  language = "english",
  modelType = "lasso",
  filterTerms = NULL,
  control = list(),
  minWordLength = 3,
  sparsity = 0.9,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...)

## S3 method for class 'character'
generateDictionary(
  x,
  response,
  language = "english",
  modelType = "lasso",
  filterTerms = NULL,
  control = list(),
  minWordLength = 3,
  sparsity = 0.9,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...)

## S3 method for class 'data.frame'
generateDictionary(
  x,
  response,
  language = "english",
  modelType = "lasso",
  filterTerms = NULL,
  control = list(),
  minWordLength = 3,
  sparsity = 0.9,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...)

## S3 method for class 'TermDocumentMatrix'
generateDictionary(
generateDictionary

```r
genenerateDictionary(
  x,
  response,
  language = "english",
  modelType = "lasso",
  filterTerms = NULL,
  control = list(),
  minWordLength = 3,
  sparsity = 0.9,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...
)
```

## S3 method for class 'DocumentTermMatrix'

generateDictionary(
  x,
  response,
  language = "english",
  modelType = "lasso",
  filterTerms = NULL,
  control = list(),
  minWordLength = 3,
  sparsity = 0.9,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...
)

### Arguments

**x**  
A vector of characters, a data.frame, an object of type `Corpus`, `TermDocumentMatrix` or `DocumentTermMatrix`.

**response**  
Response variable including the given gold standard.

**language**  
Language used for preprocessing operations (default: English).

**modelType**  
A string denoting the estimation method. Allowed values are `lasso`, `ridge`, `enet`, `lm` or `glm` or `spikeslab`.

**filterTerms**  
Optional vector of strings (default: `NULL`) to filter terms that are used for dictionary generation.

**control**  
(optional) A list of parameters defining the model used for dictionary generation. If `modelType=lasso` is selected, individual parameters are as follows:

- "s" Value of the parameter lambda at which the LASSO is evaluated. Default is `s="lambda.1se"` which takes the calculated minimum value for lambda and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by `lambda="lambda.min"`.
- "family" Distribution for response variable. Default is `family="gaussian"`. For non-negative counts, use `family="poisson"`. For binary variables `family="binomial"`. See `glmnet` for further details.
- "grouped" Determines whether grouped LASSO is used (with default `FALSE`).
If `modelType=ridge` is selected, individual parameters are as follows:

- "s" Value of the parameter lambda at which the ridge is evaluated. Default is `s="lambda.1se"` which takes the calculated minimum value for \( \lambda \) and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by `lambda="lambda.min"`.
- "family" Distribution for response variable. Default is `family="gaussian"`. For non-negative counts, use `family="poisson"`. For binary variables `family="binomial"`. See `glmnet` for further details.
- "grouped" Determines whether grouped function is used (with default FALSE).

If `modelType=enet` is selected, individual parameters are as follows:

- "alpha" Abstraction parameter for switching between LASSO (with \( \alpha=1 \)) and ridge regression (\( \alpha=0 \)). Default is \( \alpha=0.5 \). Recommended option is to test different values between 0 and 1.
- "s" Value of the parameter lambda at which the elastic net is evaluated. Default is `s="lambda.1se"` which takes the calculated minimum value for \( \lambda \) and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by `lambda="lambda.min"`.
- "family" Distribution for response variable. Default is `family="gaussian"`. For non-negative counts, use `family="poisson"`. For binary variables `family="binomial"`. See `glmnet` for further details.
- "grouped" Determines whether grouped function is used (with default FALSE).

If `modelType=lm` is selected, no parameters are passed on.

If `modelType=glm` is selected, individual parameters are as follows:

- "family" Distribution for response variable. Default is `family="gaussian"`. For non-negative counts, use `family="poisson"`. For binary variables `family="binomial"`. See `glm` for further details.

If `modelType=spikeslab` is selected, individual parameters are as follows:

- "n.iter1" Number of burn-in Gibbs sampled values (i.e., discarded values). Default is 500.
- "n.iter2" Number of Gibbs sampled values, following burn-in. Default is 500.

`minWordLength` Removes words given a specific minimum length (default: 3). This preprocessing is applied when the input is a character vector or a corpus and the document-term matrix is generated inside the routine.

`sparsity` A numeric for removing sparse terms in the document-term matrix. The argument `sparsity` specifies the maximal allowed sparsity. Default is `sparsity=0.9`, however, this is only applied when the document-term matrix is calculated inside the routine.

`weighting` Weights a document-term matrix by e.g. term frequency - inverse document frequency (default). Other variants can be used from `DocumentTermMatrix`.

... Additional parameters passed to function for e.g. preprocessing or `glmnet`. 
Value
Result is a matrix which sentiment values for each document across all defined rules

Source
doi:10.1371/journal.pone.0209323

References

See Also
analyzeSentiment,predict.SentimentDictionaryWeighted,plot.SentimentDictionaryWeighted and compareToResponse for advanced evaluations

Examples
# Create a vector of strings
documents <- c("This is a good thing!",
               "This is a very good thing!",
               "This is okay.",
               "This is a bad thing.",
               "This is a very bad thing.")
response <- c(1, 0.5, 0, -0.5, -1)

# Generate dictionary with LASSO regularization
dictionary <- generateDictionary(documents, response)

# Show dictionary
dictionary
summary(dictionary)
plot(dictionary)

# Compute in-sample performance
sentiment <- predict(dictionary, documents)
compareToResponse(sentiment, response)
plotSentimentResponse(sentiment, response)

# Generate new dictionary with spike-and-slab regression instead of LASSO regularization
library(spikeslab)
dictionary <- generateDictionary(documents, response, modelType="spikeslab")

# Generate new dictionary with tf weighting instead of tf-idf
library(tm)
dictionary <- generateDictionary(documents, response, weighting=weightTf)
sentiment <- predict(dictionary, documents)
compareToResponse(sentiment, response)

# Use instead lambda.min from the LASSO estimation
dictionary <- generateDictionary(documents, response, control=list(s="lambda.min"))
sentiment <- predict(dictionary, documents)
compareToResponse(sentiment, response)

# Use instead OLS as estimation method
dictionary <- generateDictionary(documents, response, modelType="lm")
sentiment <- predict(dictionary, documents)
sentiment

dictionary <- generateDictionary(documents, response, modelType="lm",
filterTerms = c("good", "bad"))
sentiment <- predict(dictionary, documents)
sentiment

dictionary <- generateDictionary(documents, response, modelType="lm",
filterTerms = extractWords(loadDictionaryGI()))
sentiment <- predict(dictionary, documents)
sentiment

# Generate dictionary without LASSO intercept
dictionary <- generateDictionary(documents, response, intercept=FALSE)
dictionary$intercept

## Not run:
imdb <- loadImdb()

# Generate Dictionary
dictionary_imdb <- generateDictionary(imdb$Corpus, imdb$Rating, family="poisson")
summary(dictionary_imdb)
compareDictionaries(dictionary_imdb, loadDictionaryGI())

# Show estimated coefficients with Kernel Density Estimation (KDE)
plot(dictionary_imdb)
plot(dictionary_imdb) + xlim(c(-0.1, 0.1))

# Compute in-sample performance
pred_sentiment <- predict(dict_imdb, imdb$Corpus)
compareToResponse(pred_sentiment, imdb$Rating)

# Test a different sparsity parameter
dictionary_imdb <- generateDictionary(imdb$Corpus, imdb$Rating, family="poisson", sparsity=0.99)
summary(dictionary_imdb)
pred_sentiment <- predict(dict_imdb, imdb$Corpus)
compareToResponse(pred_sentiment, imdb$Rating)

## End(Not run)
lassoEstimation

Description
Function estimates coefficients based on LASSO regularization.

Usage
lassoEstimation(
  x,
  response,
  control = list(alpha = 1, s = "lambda.min", family = "gaussian", grouped = FALSE),
  ...
)

Arguments
x An object of type DocumentTermMatrix.
response Response variable including the given gold standard.
control (optional) A list of parameters defining the LASSO model as follows:
  • "family"Distribution for response variable. Default is family="gaussian".
    For non-negative counts, use family="poisson". For binary variables
    family="binomial". See glm for further details.
... Additional parameters passed to function for glm.

Value
Result is a list with coefficients, coefficient names and the model intercept.

lassoEstimation Lasso estimation

Description
Function estimates coefficients based on LASSO regularization.

Usage
lassoEstimation(
  x,
  response,
  control = list(alpha = 1, s = "lambda.min", family = "gaussian", grouped = FALSE),
  ...
)

Arguments
x An object of type DocumentTermMatrix.
response Response variable including the given gold standard.
control (optional) A list of parameters defining the LASSO model as follows:
lmEstimation

• "s" Value of the parameter lambda at which the LASSO is evaluated. Default is s="lambda.1se" which takes the calculated minimum value for \( \lambda \) and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by lambda="lambda.min".

• "family" Distribution for response variable. Default is family="gaussian". For non-negative counts, use family="poisson". For binary variables family="binomial". See glmnet for further details.

• "grouped" Determines whether grouped LASSO is used (with default FALSE).

... Additional parameters passed to function for glmnet.

Value

Result is a list with coefficients, coefficient names and the model intercept.

Description

Function estimates coefficients based on ordinary least squares.

Usage

lmEstimation(x, response, control = list(), ...)

Arguments

x An object of type DocumentTermMatrix.
response Response variable including the given gold standard.
control (optional) A list of parameters (not used).
... Additional parameters (not used).

Value

Result is a list with coefficients, coefficient names and the model intercept.
**loadDictionaryGI**

*Loads Harvard-IV dictionary into object*

**Description**

Loads Harvard-IV dictionary (as used in General Inquirer) into a standardized dictionary object

**Usage**

loadDictionaryGI()

**Value**

object of class *SentimentDictionary*

**Note**

Result is a list of stemmed words in lower case

---

**loadDictionaryHE**

*Loads Henry’s finance-specific dictionary into object*

**Description**

Loads Henry’s finance-specific dictionary into a standardized dictionary object

**Usage**

loadDictionaryHE()

**Value**

object of class *SentimentDictionary*

**Note**

Result is a list of stemmed words in lower case
loadDictionaryLM
 loads Loughran-McDonald dictionary into object

Description
Loads Loughran-McDonald financial dictionary into a standardized dictionary object (here, categories positive and negative are considered)

Usage
loadDictionaryLM()

Value
object of class SentimentDictionary

Note
Result is a list of stemmed words in lower case

loadDictionaryLM_Uncertainty
 loads uncertainty words from Loughran-McDonald into object

Description
Loads uncertainty words from Loughran-McDonald into a standardized dictionary object

Usage
loadDictionaryLM_Uncertainty()

Value
object of class SentimentDictionary

Note
Result is a list of stemmed words in lower case
**loadDictionaryQDAP**

*Description*

Loads polarity words from data object `key.pol` which is by the package qdap. This is then converted into a standardized dictionary object.

*Usage*

```r
loadDictionaryQDAP()
```

*Value*

Object of class `SentimentDictionary`.

*Note*

Result is a list of stemmed words in lower case.

*Source*

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

*References*


---

**loadImdb**

*Description*

Function downloads IMDb dataset and prepares corresponding user ratings for easy usage.

*Usage*

```r
loadImdb()
```

*Value*

Returns a list where entry named `Corpus` contains the IMDb reviews, and `Rating` is the corresponding scaled rating.
References


Examples

```r
# Not run:
imdb <- loadImdb()
dictionary <- generateDictionary(imdb$Corpus, imdb$Rating)

# End(Not run)
```

`lookupEstimationMethod`

*Estimation method*

**Description**

Decides upon an estimation method for dictionary generation. Input is a name for the estimation method, output is the corresponding function object.

**Usage**

`lookupEstimationMethod(type)`

**Arguments**

- `type` A string denoting the estimation method. Allowed values are `lasso`, `ridge`, `enet`, `lm`, `glm` or `spikeslab`.

**Value**

Function that implements the specific estimation method.

`ngram_tokenize`

*N-gram tokenizer*

**Description**

A tokenizer for use with a document-term matrix from the tm package. Supports both character and word ngrams, including own wrapper to handle non-Latin encodings.

**Usage**

`ngram_tokenize(x, char = FALSE, ngmin = 1, ngmax = 3)`
numEntries

Arguments

x    input string
char boolean value specifying whether to use character (char = TRUE) or word n-grams (char = FALSE, default)
ngmin integer giving the minimum order of n-gram (default: 1)
ngmax integer giving the maximum order of n-gram (default: 3)

Examples

library(tm)
en <- c("Romeo loves Juliet", "Romeo loves a girl")
en.corpus <- VCorpus(VectorSource(en))
tdm <- TermDocumentMatrix(en.corpus,
    control=list(wordLengths=c(1,Inf),
    tokenize=function(x) ngram_tokenize(x, char=TRUE, ngmin=3, ngmax=3)))
inspect(tdm)

ch <- c("abab", "aabb")
ch.corpus <- VCorpus(VectorSource(ch))
tdm <- TermDocumentMatrix(ch.corpus,
    control=list(wordLengths=c(1,Inf),
    tokenize=function(x) ngram_tokenize(x, char=TRUE, ngmin=1, ngmax=2)))
inspect(tdm)

numEntries

Number of words in dictionary

Description

Counts total number of entries in dictionary.

Usage

numEntries(d)

Arguments

d Dictionary of type SentimentDictionaryWordlist, SentimentDictionaryBinary or SentimentDictionaryWeighted

See Also

numPositiveEntries and numNegativeEntries for more option to count the number of entries
Examples

numEntries(SentimentDictionary(c("uncertain", "possible", "likely"))) # returns 3
numEntries(SentimentDictionary(c("increase", "rise", "more"),
  c("fall", "drop"))) # returns 5
numEntries(SentimentDictionary(c("increase", "decrease", "exit"),
  c(+1, -1, -10),
  rep(NA, 3))) # returns 3

numNegativeEntries  Number of negative words in dictionary

Description

Counts total number of negative entries in dictionary.

Usage

numNegativeEntries(d)

Arguments

d is a dictionary of type SentimentDictionaryBinary or SentimentDictionaryWeighted

Note

Entries in SentimentDictionaryWeighted with a weight of 0 are not counted here

See Also

numEntries and numPositiveEntries for more option to count the number of entries

Examples

numNegativeEntries(SentimentDictionary(c("increase", "rise", "more"),
  c("fall", "drop"))) # returns 2
numNegativeEntries(SentimentDictionary(c("increase", "decrease", "exit"),
  c(+1, -1, -10),
  rep(NA, 3))) # returns 2
numPositiveEntries

numPositiveEntries  Number of positive words in dictionary

Description
Counts total number of positive entries in dictionary.

Usage
numPositiveEntries(d)

Arguments
d  is a dictionary of type SentimentDictionaryBinary or SentimentDictionaryWeighted

Note
Entries in SentimentDictionaryWeighted with a weight of 0 are not counted here

See Also
numEntries and numNegativeEntries for more option to count the number of entries

Examples
numPositiveEntries(SentimentDictionary(c("increase", "rise", "more"),
  c("fall", "drop")))  # returns 3
numPositiveEntries(SentimentDictionary(c("increase", "decrease", "exit"),
  c(+1, -1, -10),
  rep(NA, 3)))  # returns 1

plot.SentimentDictionaryWeighted

plot.SentimentDictionaryWeighted  KDE plot of estimated coefficients

Description
Function performs a Kernel Density Estimation (KDE) of the coefficients and then plot these using
ggplot. This type of plot allows to inspect whether the distribution of coefficients is skew. This can reveal if there are more positive terms than negative or vice versa.

Usage
## S3 method for class 'SentimentDictionaryWeighted'
plot(x, color = "gray60", theme = ggplot2::theme_bw(), ...)

---

---
**plotSentiment**

**Arguments**
- **x**: Dictionary of class `SentimentDictionaryWeighted`
- **color**: Color for filling the density plot (default: gray color)
- **theme**: Visualization theme for `ggplot` (default: a black-white theme)
- ... Additional parameters passed to function.

**Value**
Returns a plot of class `ggplot`.

**See Also**
- `plotSentiment` and `plotSentimentResponse` for further plotting options.

**Examples**

```r
d <- SentimentDictionaryWeighted(paste0(character(100), 1:100), rnorm(100), numeric(100))
plot(d)

# Change color in plot
plot(d, color="red")

library(ggplot2)
# Extend plot with additional layout options
plot(d) + ggtitle("KDE plot")
plot(d) + theme_void()
```

---

**plotSentiment**

*Line plot with sentiment scores*

**Description**

Simple line plot to visualize the evolvement of sentiment scores. This is especially helpful when studying a time series of sentiment scores.

**Usage**

```r
plotSentiment(
  sentiment,
  x = NULL,
  cumsum = FALSE,
  xlab = "",
  ylab = "Sentiment"
)
```
plotSentimentResponse

Arguments

- **sentiment**: data.frame or numeric vector with sentiment scores
- **x**: Optional parameter with labels or time stamps on x-axis.
- **cumsum**: Parameter deciding whether the cumulative sentiment is plotted (default: cumsum=FALSE).
- **xlab**: Name of x-axis (default: empty string).
- **ylab**: Name of y-axis (default: "Sentiment").

Value

Returns a plot of class ggplot

See Also

plotSentimentResponse and plot.SentimentDictionaryWeighted for further plotting options

Examples

```r
sentiment <- data.frame(Dictionary=runif(20))
plotSentiment(sentiment)
plotSentiment(sentiment, cumsum=TRUE)

# Change name of x-axis
plotSentiment(sentiment, xlab="Tone")

library(ggplot2)
# Extend plot with additional layout options
plotSentiment(sentiment) + ggtitle("Evolving sentiment")
plotSentiment(sentiment) + theme_void()
```

plotSentimentResponse Scatterplot with trend line between sentiment and response

Description

Generates a scatterplot where points pairs of sentiment and the response variable. In addition, the plot addas a trend line in the form of a generalized additive model (GAM). Other smoothing variables are possible based on geom_smooth. This functions is helpful for visualization the relationship between computed sentiment scores and the gold standard.

Usage

```r
plotSentimentResponse(
  sentiment,
  response,
  smoothing = "gam",
  xlab = "Sentiment",
  ylab = "Response"
)
```
predict.SentimentDictionaryWeighted

Arguments

sentiment  data.frame with sentiment scores
response   Vector with response variables of the same length
smoothing  Smoothing functionality. Default is smoothing="gam" to utilize a generalized additive model (GAM). Other options can be e.g. a linear trend line (smoothing="lm"); see geom_smooth for a full list of options.
xlab       Description on x-axis (default: "Sentiment").
ylab       Description on y-axis (default: "Sentiment").

Value

Returns a plot of class ggplot

See Also

plotSentiment and plot.SentimentDictionaryWeighted for further plotting options

Examples

sentiment <- data.frame(Dictionary=runif(10))
response <- sentiment[[1]] + rnorm(10)

plotSentimentResponse(sentiment, response)

# Change x-axis
plotSentimentResponse(sentiment, response, xlab="Tone")

library(ggplot2)
# Extend plot with additional layout options
plotSentimentResponse(sentiment, response) + ggtitle("Scatterplot")
plotSentimentResponse(sentiment, response) + theme_void()

predict.SentimentDictionaryWeighted

Prediction for given dictionary

Description

Function takes a dictionary of class SentimentDictionaryWeighted with weights as input. It then applies this dictionary to textual contents in order to calculate a sentiment score.
Usage

## S3 method for class 'SentimentDictionaryWeighted'
predict(
  object,
  newdata = NULL,
  language = "english",
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE),
  ...
)

Arguments

object Dictionary of class SentimentDictionaryWeighted.
newdata A vector of characters, a data.frame, an object of type Corpus, TermDocumentMatrix or DocumentTermMatrix.
language Language used for preprocessing operations (default: English).
weighting Function used for weighting of words; default is a link to the tf-idf scheme.
... Additional parameters passed to function for e.g. preprocessing.

Value
data.frame with predicted sentiment scores.

See Also

SentimentDictionaryWeighted, generateDictionary and compareToResponse for default dictionary generations

Examples

#' # Create a vector of strings
documents <- c("This is a good thing!",
  "This is a very good thing!",
  "This is okay.",
  "This is a bad thing.",
  "This is a very bad thing."
)
response <- c(1, 0.5, 0, -0.5, -1)

# Generate dictionary with LASSO regularization
dictionary <- generateDictionary(documents, response)

# Compute in-sample performance
sentiment <- predict(dictionary, documents)
compareToResponse(sentiment, response)
**preprocessCorpus**  
*Default preprocessing of corpus*

**Description**

Preprocess existing corpus of type `Corpus` according to default operations. This helper function groups all standard preprocessing steps such that the usage of the package is more convenient.

**Usage**

```r
preprocessCorpus(
  corpus,
  language = "english",
  stemming = TRUE,
  verbose = FALSE,
  removeStopwords = TRUE
)
```

**Arguments**

- `corpus`  
  Corpus object which should be processed
- `language`  
  Default language used for preprocessing (i.e. stop word removal and stemming)
- `stemming`  
  Perform stemming (default: TRUE)
- `verbose`  
  Print preprocessing status information
- `removeStopwords`  
  Flag indicating whether to remove stopwords or not (default: yes)

**Value**

Object of `Corpus`

---

**print.SentimentDictionaryWordlist**  
*Output content of sentiment dictionary*

**Description**

Prints entries of sentiment dictionary to the screen
### Usage

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

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

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

### Arguments

- `x`  
  Sentiment dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary` or `SentimentDictionaryWeighted`

- `...`  
  Additional parameters passed to specific sub-routines

### See Also

- `summary` for showing a brief summary

### Examples

```r
print(SentimentDictionary(c("uncertain", "possible", "likely")))
print(SentimentDictionary(c("increase", "rise", "more"),
                         c("fall", "drop")))
print(SentimentDictionary(c("increase", "decrease", "exit"),
                         c(+1, -1, -10),
                         rep(NA, 3)))
```

---

### Description

This routine reads a sentiment dictionary from a text file. Such a text file can be created e.g. via `write`. The dictionary type is recognized according to the internal format of the file.

### Usage

```r
read(file)
```

### Arguments

- `file`  
  File name pointing to text file

### Value

Dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary` or `SentimentDictionaryWeighted`
See Also

*write* for creating such a file

Examples

d.out <- SentimentDictionary(c("uncertain", "possible", "likely"))
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

d.out <- SentimentDictionary(c("increase", "rise", "more"),
                             c("fall", "drop"))
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

d.out <- SentimentDictionary(c("increase", "decrease", "exit"),
                             c(+1, -1, -10),
                             rep(NA, 3),
                             intercept=5)
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

unlink("example.dict")

---

ridgeEstimation  
*Ridge estimation*

Description

Function estimates coefficients based on ridge regularization.

Usage

```r
ridgeEstimation(
  x,  
  response,  
  control = list(s = "lambda.min", family = "gaussian", grouped = FALSE),  
  ... 
)
```

Arguments

- **x** An object of type `DocumentTermMatrix`.
- **response** Response variable including the given gold standard.
- **control** (optional) A list of parameters defining the model as follows:
• "s"Value of the parameter lambda at which the ridge is evaluated. Default is s="lambda.1se" which takes the calculated minimum value for \( \lambda \) and then subtracts one standard error in order to avoid overfitting. This often results in a better performance than using the minimum value itself given by lambda="lambda.min".

• "family"Distribution for response variable. Default is family="gaussian". For non-negative counts, use family="poisson". For binary variables family="binomial". See glmnet for further details.

• "grouped" Determines whether grouped function is used (with default FALSE). Additional parameters passed to function for glmnet.

Value

Result is a list with coefficients, coefficient names and the model intercept.

ruleLinearModel  

Sentiment based on linear model

Description

Sentiment score as denoted by a linear model.

Usage

ruleLinearModel(dtm, d)

Arguments

dtm  Document-term matrix
d  Dictionary of type SentimentDictionaryWeighted

Value

Continuous sentiment score
**ruleNegativity**

**Ratio of negative words**

**Description**

Ratio of words labeled as negative in that dictionary compared to the total number of words in the document. Here, it uses the entry `negativeWords` of the `SentimentDictionaryBinary`.

**Usage**

`ruleNegativity(dtm, d)`

**Arguments**

- `dtm` Document-term matrix
- `d` Dictionary of type `SentimentDictionaryBinary`

**Value**

Ratio of negative words compared to all

---

**rulePositivity**

**Ratio of positive words**

**Description**

Ratio of words labeled as positive in that dictionary compared to the total number of words in the document. Here, it uses the entry `positiveWords` of the `SentimentDictionaryBinary`.

**Usage**

`rulePositivity(dtm, d)`

**Arguments**

- `dtm` Document-term matrix
- `d` Dictionary of type `SentimentDictionaryBinary`

**Value**

Ratio of positive words compared to all
**ruleRatio**  
*Ratio of dictionary words*

**Description**
Ratio of words in that dictionary compared to the total number of words in the document

**Usage**
\[
\text{ruleRatio}(\text{dtm}, \text{d})
\]

**Arguments**
- \(\text{dtm}\)  
  Document-term matrix
- \(\text{d}\)  
  Dictionary of type \text{SentimentDictionaryWordlist} with words belonging to a single category

**Value**
Ratio of dictionary words compared to all

---

**ruleSentiment**  
*Sentiment score*

**Description**
Sentiment score defined as the difference between positive and negative word counts divided by the total number of words.

**Usage**
\[
\text{ruleSentiment}(\text{dtm}, \text{d})
\]

**Arguments**
- \(\text{dtm}\)  
  Document-term matrix
- \(\text{d}\)  
  Dictionary of type \text{SentimentDictionaryBinary}

**Details**
Given the number of positive words \(P\) and the number of negative words \(N\). Further, let \(T\) denote the total number of words in that document. Then, the sentiment ratio is defined as

\[
\frac{P - N}{T}
\]

Here, it uses the entries \text{negativeWords} and \text{positiveWords} of the \text{SentimentDictionaryBinary}.
Value
Sentiment score in the range of -1 to 1.

ruleSentimentPolarity  
_Sentiment polarity score_

Description
Sentiment score defined as the difference between positive and negative word counts divided by the sum of positive and negative words.

Usage
ruleSentimentPolarity(dtm, d)

Arguments
- `dtm`  Document-term matrix
- `d`  Dictionary of type `SentimentDictionaryBinary`

Details
Given the number of positive words \( P \) and the number of negative words \( N \). Then, the sentiment ratio is defined as
\[
\frac{P - N}{P + N}
\]
. Here, it uses the entries `negativeWords` and `positiveWords` of the `SentimentDictionaryBinary`.

Value
Sentiment score in the range of -1 to 1.

ruleWordCount  
_Counts word frequencies_

Description
Counts total word frequencies in each document

Usage
ruleWordCount(dtm)

Arguments
- `dtm`  Document-term matrix
SentimentDictionary

Value

- Total number of words

---

SentimentDictionary

Create new sentiment dictionary based on input

Description

Depending on the input, this function creates a new sentiment dictionary of different type.

Usage

SentimentDictionary(...)  

Arguments

...  

Arguments as passed to one of the three functions SentimentDictionaryWordlist, SentimentDictionaryBinary or SentimentDictionaryWeighted

See Also

SentimentDictionaryWordlist, SentimentDictionaryBinary, SentimentDictionaryWeighted

---

SentimentDictionaryBinary

Create a sentiment dictionary of positive and negative words

Description

This routine creates a new object of type SentimentDictionaryBinary that stores two separate vectors of negative and positive words.

Usage

SentimentDictionaryBinary(positiveWords, negativeWords)

Arguments

- positiveWords is a vector containing the entries labeled as positive
- negativeWords is a vector containing the entries labeled as negative

Value

- Returns a new object of type SentimentDictionaryBinary
See Also

`SentimentDictionary`

Examples

```r
# generate a dictionary with positive and negative words
d <- SentimentDictionaryBinary(c("increase", "rise", "more"),
    c("fall", "drop"))
summary(d)
# alternative call
d <- SentimentDictionary(c("increase", "rise", "more"),
    c("fall", "drop"))
summary(d)
```

---

### SentimentDictionaryWeighted

*Create a sentiment dictionary of words linked to a score*

**Description**

This routine creates a new object of type `SentimentDictionaryWeighted` that contains a number of words, each linked to a continuous score (i.e. weight) for specifying its polarity. The scores can later be interpreted as a linear model.

**Usage**

```r
SentimentDictionaryWeighted(
    words, scores,
    idf = rep(1, length(words)),
    intercept = 0
)
```

**Arguments**

- `words` is a collection (vector) of different words as strings.
- `scores` are the corresponding scores or weights denoting the word’s polarity.
- `idf` provide further details on the frequency of words in the corpus as an additional source for normalization.
- `intercept` is an optional parameter for shifting the zero level (default: 0).

**Value**

Returns a new object of type `SentimentDictionaryWordlist`
Note

The intercept is useful when the mean or median of a response variable is not exactly located at zero. For instance, stock market returns have slight positive bias.

Source

doi:10.1371/journal.pone.0209323

References


See Also

SentimentDictionary

Examples

# generate dictionary (based on linear model)
d <- SentimentDictionaryWeighted(c("increase", "decrease", "exit"),
c(+1, -1, -10),
rep(NA, 3))
summary(d)
# alternative call
d <- SentimentDictionaryWeighted(c("increase", "decrease", "exit"),
c(+1, -1, -10))
summary(d)
# alternative call
d <- SentimentDictionary(c("increase", "decrease", "exit"),
c(+1, -1, -10),
rep(NA, 3))
summary(d)

SentimentDictionaryWordlist

Create a sentiment dictionary consisting of a simple wordlist

Description

This routine creates a new object of type SentimentDictionaryWordlist

Usage

SentimentDictionaryWordlist(wordlist)

Arguments

wordlist is a vector containing the individual entries as strings
spikeslabEstimation

Value

Returns a new object of type SentimentDictionaryWordlist

See Also

SentimentDictionary

Examples

# generate a dictionary with "uncertainty" words
d <- SentimentDictionaryWordlist(c("uncertain", "possible", "likely"))
summary(d)
# alternative call
d <- SentimentDictionary(c("uncertain", "possible", "likely"))
summary(d)

spikeslabEstimation  Spike-and-slab estimation

Description

Function estimates coefficients based on spike-and-slab regression.

Usage

spikeslabEstimation(
  x,
  response,
  control = list(n.iter1 = 500, n.iter2 = 500),
  ...
)

Arguments

x  An object of type DocumentTermMatrix.
response  Response variable including the given gold standard.
control  (optional) A list of parameters defining the LASSO model. Default isn.iter1=500 and n.iter2=500. See spikeslab for details.
...

Value

Result is a list with coefficients, coefficient names and the model intercept.
**summary.SentimentDictionaryWordlist**

*Output summary information on sentiment dictionary*

**Description**

Output summary information on sentiment dictionary

**Usage**

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

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

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

**Arguments**

- `object` Sentiment dictionary of type `SentimentDictionaryWordlist`, `SentimentDictionaryBinary`, or `SentimentDictionaryWeighted`
- `...` Additional parameters passed to specific sub-routines

**See Also**

`print` for output the entries of a dictionary

**Examples**

```r
summary(SentimentDictionary(c("uncertain", "possible", "likely")))
summary(SentimentDictionary(c("increase", "rise", "more"),
                           c("fall", "drop")))
summary(SentimentDictionary(c("increase", "decrease", "exit"),
                            c(+1, -1, -10),
                            rep(NA, 3)))
```

**toDocumentTermMatrix**

*Default preprocessing of corpus and conversion to document-term matrix*

**Description**

Preprocess existing corpus of type `Corpus` according to default operations. This helper function groups all standard preprocessing steps such that the usage of the package is more convenient. The result is a document-term matrix.
transformIntoCorpus

Usage

toDocumentTermMatrix(
  x,
  language = "english",
  minWordLength = 3,
  sparsity = NULL,
  removeStopwords = TRUE,
  stemming = TRUE,
  weighting = function(x) tm::weightTfIdf(x, normalize = FALSE)
)

Arguments

x Corpus object which should be processed
language Default language used for preprocessing (i.e. stop word removal and stemming)
minWordLength Minimum length of words used for cut-off; i.e. shorter words are removed. Default is 3.
sparsity A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one. Default is NULL in order suppress this functionality.
removeStopwords Flag indicating whether to remove stopwords or not (default: yes)
stemming Perform stemming (default: TRUE)
weighting Function used for weighting of words; default is a a link to the tf-idf scheme.

Value

Object of DocumentTermMatrix

See Also

DocumentTermMatrix for the underlying class

transformIntoCorpus  Transforms the input into a Corpus object

Description

Takes the given input of characters and transforms it into a Corpus. The input is checked to match the expected class and format.

Usage

transformIntoCorpus(x)
write

Arguments

x A list, data.frame or vector consisting of characters

Value

The generated Corpus

Note

Factors are automatically casted into characters but with printing a warning

See Also

preprocessCorpus for further preprocessing, analyzeSentiment for subsequent sentiment analysis

Examples

transformIntoCorpus(c("Document 1", "Document 2", "Document 3"))
transformIntoCorpus(list("Document 1", "Document 2", "Document 3"))
transformIntoCorpus(data.frame("Document 1", "Document 2", "Document 3"))

write
Write dictionary to text file

Description

This routine exports a sentiment dictionary to a text file which can be the source for additional problems or controlling the output.

Usage

write(d, file)

## S3 method for class 'SentimentDictionaryWordlist'
write(d, file)

## S3 method for class 'SentimentDictionaryBinary'
write(d, file)

## S3 method for class 'SentimentDictionaryWeighted'
write(d, file)

Arguments

d Dictionary of type SentimentDictionaryWordlist, SentimentDictionaryBinary or SentimentDictionaryWeighted

file File to which the dictionary should be exported
See Also

read for later access

Examples

d.out <- SentimentDictionary(c("uncertain", "possible", "likely"))
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

d.out <- SentimentDictionary(c("increase", "rise", "more"),
                              c("fall", "drop"))
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

\[
d.out <- \text{SentimentDictionary}(c("increase", "decrease", "exit"),
                                 c(+1, -1, -10),
                                 \text{rep(NA, 3)},
                                 \text{intercept=5})
\]
write(d.out, "example.dict")
d.in <- read("example.dict")
print(d.in)

unlink("example.dict")
Index

* corpus
  preprocessCorpus, 34
  toDocumentTermMatrix, 45
  transformIntoCorpus, 46

* datasets
  DictionaryGI, 12
  DictionaryHE, 12
  DictionaryLM, 13
  loadImdb, 25

* dictionary
  compareDictionaries, 6
  extractWords, 15
  generateDictionary, 15
  numEntries, 27
  numNegativeEntries, 28
  numPositiveEntries, 29
  predict.SentimentDictionaryWeighted, 32
  print.SentimentDictionaryWordlist, 34
  read, 35
  SentimentDictionary, 41
  SentimentDictionaryBinary, 41
  SentimentDictionaryWeighted, 42
  SentimentDictionaryWordlist, 43
  summary.SentimentDictionaryWordlist, 45
  write, 47

* evaluation
  compareToResponse, 7
  convertToBinaryResponse, 8
  convertToDirection, 9
  generateDictionary, 15
  plot.SentimentDictionaryWeighted, 29
  plotSentiment, 30
  plotSentimentResponse, 31
  predict.SentimentDictionaryWeighted, 32

* plots
  plot.SentimentDictionaryWeighted, 29
  plotSentiment, 30
  plotSentimentResponse, 31

* preprocessing
  ngram_tokenize, 26
  preprocessCorpus, 34
  toDocumentTermMatrix, 45
  transformIntoCorpus, 46

* rules
  ruleLinearModel, 37
  ruleNegativity, 38
  ruleRatio, 39
  ruleSentiment, 39
  ruleSentimentPolarity, 40
  ruleWordCount, 40

* sentiment
  analyzeSentiment, 3, 19, 47
  compareToResponse, 5, 7, 19, 33
  convertToBinaryResponse, 5, 8, 9
  convertToDirection, 5, 8, 9
  Corpus, 4, 11, 17, 33, 34, 45, 46
  countWords, 10
  DictionaryGI, 12
  DictionaryHE, 12
  DictionaryLM, 13
  DocumentTermMatrix, 4, 11, 14, 17, 18, 21, 22, 33, 36, 44, 46
  enetEstimation, 14

49
extractWords, 15

generateDictionary, 5, 15, 33
geom_smooth, 31, 32
ggplot, 29–32
glm, 18, 21
glmEstimation, 20
glmnet, 14, 17, 18, 22, 37

key.pol, 25

lassoEstimation, 21
lmEstimation, 22
loadDictionaryGI, 23
loadDictionaryHE, 23
loadDictionaryLM, 24
loadDictionaryLM_Uncertainty, 24
loadDictionaryQDAP, 25
loadImdb, 25
lookupEstimationMethod, 26

ngram_tokenize, 26
numEntries, 27, 28, 29
numNegativeEntries, 27, 28, 29
numPositiveEntries, 27, 28, 29

plot.SentimentDictionaryWeighted, 19, 29, 31, 32
plotSentiment, 5, 30, 30, 32
plotSentimentResponse, 5, 30, 31, 31
predict.SentimentResponse, 5, 30, 31, 31
preprocessCorpus, 34, 47
print, 45
print.SentimentDictionaryBinary
(print.SentimentDictionaryWordlist, 34)
print.SentimentDictionaryWeighted
(print.SentimentDictionaryWordlist, 34)
print.SentimentDictionaryWordlist, 34

read, 35, 48
ridgeEstimation, 36
ruleLinearModel, 37
ruleNegativity, 38
rulePositivity, 38
ruleRatio, 39
ruleSentiment, 39
ruleSentimentPolarity, 40

ruleWordCount, 40

SentimentDictionary, 23–25, 41, 42–44
SentimentDictionaryBinary, 6, 15, 27–29, 35, 38–41, 41, 45, 47
SentimentDictionaryWeighted, 6, 15, 27–30, 32, 33, 35, 37, 41, 42, 45, 47
SentimentDictionaryWordlist, 6, 15, 27, 35, 39, 41, 43, 45, 47

spikeslab, 44
spikeslabEstimation, 44
summary, 35
summary.SentimentDictionaryBinary
(summary.SentimentDictionaryWordlist), 45
summary.SentimentDictionaryWeighted
(summary.SentimentDictionaryWordlist), 45
summary.SentimentDictionaryWordlist, 45

TermDocumentMatrix, 4, 11, 17, 33
toDocumentTermMatrix, 45
transformIntoCorpus, 46

write, 35, 36, 47