Package 'languagePredictR'

March 30, 2021

Type Package

Title Predict Outcomes from Natural Language

Version 0.1.0

Description This package uses natural language responses to predict binary and continuous outcome variables. Regression models regularized with a LASSO constraint identify word choices that are predictive of desired outcomes, with options to plot various useful metrics.

License GPL-3

Depends R (¿= 4.0.0)

Encoding UTF-8

LazyData true

Imports dplyr,
textclean,
hunspell,
ggplot2,
stringr,
tidyr,
data.table,
stats,
tm,
glmnet,
quanteda,
gridExtra,
pROC,
koRpus,
koRpus.lang.en,
reshape2,
scales,
rlist,
egg,
grid,
tidyverse,
pbapply,
progress,
grDevices,
graphics,
rlang,

tibble,
methods,
lubridate,
stopwords,
igraph,
doParallel,
Matrix,
yardstick,
linkcomm

RoxygenNote 7.1.1

R topics documented:

`analyze_roc` *Analyze ROC Curves*

Description

This function analyzes ROC curves from the results of the `assess_models` function

Usage

```
analyze_roc(..., plot = TRUE, plot_diagonal = FALSE)
```

Arguments

...	Output(s) of the language_model, comparison_model, or test_language_model functions
plot	If TRUE, plots a matrix displaying the results of all model comparisons. Defaults to TRUE.
plot_diagonal	if TRUE, the matrix plot will show repeated (inverted) values on the opposite diagonal. Defaults to FALSE.

Value

A dataframe with the results of statistical tests conducted on the ROCs for each model pairing

See Also

language_model, comparison_model, test_language_model

Examples

```
## Not run:
strong_movie_review_data$cleanText = clean_text(strong_movie_review_data$text)
mild_movie_review_data$cleanText = clean_text(mild_movie_review_data$text)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for strong reviews (ratings of 1 or 10)
movie_model_strong = language_model(strong_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for mild reviews (ratings of 4 or 7)
movie_model_mild = language_model(mild_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)

# Analyze ROC curves
auc_tests = analyze_roc(movie_model_strong, movie_model_mild)

## End(Not run)
```

assess_models *Create Model Assessment*

Description

This function is deprecated; all dependent functions (e.g. `plot_roc()` or `plot_predictor_words`) now take individual models as arguments. This function assesses one or more models created by the `language_model` function.

Usage

```
assess_models(...)
```

Arguments

...	Models generated by the `language_model` function, and/or two-column dataframes with a predictor variable and an outcome variable

Details

The primary purpose of this function is to be used with other functions included in this package, such as plot_roc() or predictor_word_plots() All necessary calculations are performed by this function, so output plots and analyses can be performed quickly and modified as needed This function can be used to assess models generated by the `language_model`, as well as simple predictors that could be compared with language models.

Value

An object of the type "modelAssessment"

Examples

```
## Not run:
strong_movie_review_data$cleanText = clean_text(strong_movie_review_data$text)
mild_movie_review_data$cleanText = clean_text(mild_movie_review_data$text)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for strong reviews (ratings of 1 or 10)
movie_model_strong = language_model(strong_movie_review_data,
                                    outcomeVariableColumnName = "valence",
                                    outcomeVariableType = "binary",
                                    textColumnName = "cleanText")

# Using language to predict "Positive" vs. "Negative" reviews
# Only for mild reviews (ratings of 4 or 7)
movie_model_mild = language_model(mild_movie_review_data,
                                    outcomeVariableColumnName = "valence",
                                    outcomeVariableType = "binary",
                                    textColumnName = "cleanText")

# Create the model assessment
# movie_assessment = assess_models(movie_model_strong, movie_model_mild)
```

```
## End(Not run)
```

---

check_spelling *Check Spelling*

Description

This function performs spell-checking on input text. It can provide a list of errors and probable correct spellings, as well as returning the input text with all errors corrected.

Usage

```
check_spelling(inputText, mode, customSpellingList)
```

Arguments

inputText	A character string or vector of character strings
mode	This defines the mode of operation. Options include "output", "replace", or "both". See Details below.
customSpellingList	(Optional argument) If provided, the function will use this list to correct spelling errors. Must be in the same format as the result of "output" mode, and only works in "replace" mode.

Details

This function has three modes: In the "output" mode, a dataframe is produced with three columns: the spelling errors present in the input text, how frequently they appear, and the most likely correct spelling for each word. A frequency graph is also plotted. In the "replace" mode, a character string (or vector of character strings) is produced, where all of the spelling errors identified are replaced by their most likely correct spelling. When `customSpellingList = TRUE`, the "replace" mode will only correct words in the provided list In the "both" mode, both of the above results will be produced (i.e. a list containing a dataframe of errors and suggestions, as well as the text with corrected spellings)

As a warning, this function is particularly slow, and make take a significantly long time on a sizeable vector of character strings.

Value

A dataframe (`mode="output"`), a character string or vector of character strings (`mode="replace"`), or a two-object list containing both results (`mode="both"`)

Examples

```
myString = "I went to the stroe and bought some egggs for a good porce!"

spell_check_results = check_spelling(myString, mode = "output")
spell_check_results
# error    freq    suggested_correction
# egggs    1       eggs
# stroe    1       store
```

```
# porce    1       pore

spell_correction_results = check_spelling(myString, mode = "replace")
spell_correction_results
# "I went to the store and bought some eggs for a good pore!"

error = c("egggs", "stroe", "porce")
suggested_correction = c("eggs", "store", "price")
my_corrections = data.frame(error=error, suggested_correction = suggested_correction)
correction_results = check_spelling(myString, mode = "replace", customSpellingList = my_corrections)
correction_results
# "I went to the store and bought some eggs for a good price!"
```

---

clean_text *Clean Input Text*

Description

This function cleans text by:
-Setting all text to lowercase
-Removing non-ASCII characters
-Expanding contractions ("don't" –¿ "do not")
-Removing punctuation
-Removing symbols (if replaceSymbol is FALSE)
-Removing numbers (if replaceNumber is FALSE)

Usage

```
clean_text(
  inputText,
  replaceSymbol = FALSE,
  replaceNumber = FALSE,
  removeStopwords = FALSE
)
```

Arguments

inputText	A character string or vector of character strings
replaceSymbol	If TRUE, symbols are replaced with their equivalent (e.g. "@" becomes "at"). Defaults to FALSE.
replaceNumber	If TRUE, numbers are replaced with their equivalent (e.g. "20" becomes "twenty", "3rd" becomes "third"). Defaults to FALSE.
removeStopwords	If TRUE, stopwords are removed (see [stopwords()])

Value

A character string (or vector of character strings) with cleaned text.

Examples

```
myString = "He gave his last $10 to Sally's sister because she's nice."

cleanText = clean_text(myString)
# "he gave his last to sally sister because she is nice"

cleanText = clean_text(myString, replaceNumber = TRUE)
# "he gave his last ten to sally sister because she is nice"

cleanText = clean_text(myString, replaceSymbol = TRUE)
# "he gave his last dollar to sally sister because she is nice"
```

---

comparison_model *Create Comparison Model*

Description

This function creates a regression model using a single numeric variable as a predictor, and a specified variable as the outcome. It is intended for comparison against models that use language as a predictor (created by language_model).

Usage

```
comparison_model(input, outcome, outcomeType, predictor, progressBar = TRUE)
```

Arguments

input	A dataframe containing a column with predictor data (numeric variable) and an outcome variable (numeric or two-level factor)
outcome	A string consisting of the column name for the outcome variable in inputDataframe
outcomeType	A string consisting of the type of outcome variable being used - options are "binary" or "continuous"
predictor	A string consisting of the column name for the predictor data in inputDataframe
progressBar	Show a progress bar. Defaults to TRUE.

Value

An object of the type "compModel"

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)

# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence_language = language_model(movie_review_data1,
                                      outcome = "valence",
                                      outcomeType = "binary",
                                      text = "cleanText")

summary(movie_model_valence_language)
```

```
# Is it possible that people write more for negative reviews?
# How does that compare to the language predictors?
movie_review_data1$word_count = corpus(movie_model_data1$cleanText) %>% tokens() %>% ntoken()

# Using word count to predict "Positive" vs. "Negative" reviews
movie_model_valence_wordcount = comparison_model(movie_review_data1,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    predictor = "word_count")

summary(movie_model_valence_wordcount)

## End(Not run)
```

`compModel-class` *compModel Class*

Description

compModel Class

Slots

`call` The function called to generate this model, with all arguments specified by the user

`data_predictor` The predictor variable input to create the model

`data_outcome` The outcome variable input to create the model

`type` Model type, "binary" or "continuous"

`predictor` The name of the column in the test dataframe containing the data_predictor

`outcome` The name of the column in the test dataframe containing the data_outcome

`y` The dependent (outcome) variable

`glm` The general linear model created

`predicted_y` The predicted outcomes based on the model and original predictor data

`predicted_probabilities` (If binary) The predicted probabilities of the outcomes based on the model and original predictor data

`roc` (If binary) The ROC calculated using the predicted_y

`roc_ci` (If binary) The boostrapped confidence interval calculated for the ROC

`corr` (If continuous) The correlation using the predicted_y

`level0` The bottom/first level of a binary variable, or the lowest value of a continuous variable

`level1` The top/second level of a binary variable, or the highest value of a continuous variable

`idiosync_participant_words` *Idiosyncratic Participant Words*

Description

This function identifies participants' words that are idiosyncratic (i.e. used multiple times by a single participant, and never by any other participant). It can also be used to remove these words.

Usage

```
idiosync_participant_words(
  inputDataframe,
  mode,
  textColumnName,
  participantColumnName
)
```

Arguments

`inputDataframe`	A dataframe containing a column with text data (character strings) and participant IDs
`mode`	This defines the mode of operation. Options include "output", "remove", or "both". See Details below.
`textColumnName`	A string consisting of the name of the column in `inputDataframe` which contains text data
`participantColumnName`	A string consisting of the name of the column in `inputDataframe` which contains participant IDs

Details

This function has three modes: In the "output" mode, a dataframe is produced with three columns: the participant who produced the idiosyncratic words, the words, and how frequently they are used by the participant. In the "remove" mode, a character string (or vector of character strings) is produced, where all of the idiosyncratic words are removed. In the "both" mode, both of the above results will be produced (i.e. a list containing a dataframe of idiosyncratic words, as well as the text with those words removed)

Value

A dataframe (`mode="output"`), a character string or vector of character strings (`mode="remove"`), or a two-object list containing both results (`mode="both"`)

See Also

`idiosync_response_words`

Examples

```
myStrings = c("Last week while I was walking in the park, I saw a firetruck go by. It was red.",
              "My dog loves to go on walks in the park every day of the week.",
              "Where I live, it snows all winter long. It's so cold outside.",
          "My kids love to play in the snow. They love to collect snow to build snowmen.",
              "When I was younger, I used to visit my grandmother every week.",
          "In the summertime, we would get together with my grandmother to bake cookies.")
mydataframe = data.frame(text=myStrings, participant=c(1,1,2,2,3,3), stringsAsFactors = FALSE)
idiosync_output = idiosync_participant_words(mydataframe, "output", "text", "participant")
idiosync_output
# participant     feature       frequency
# 1                park          2
# 1                go            2
# 2                love          2
# 2                snow          2
# 3                grandmother   2

idiosync_removed = idiosync_participant_words(mydataframe, "remove", "text", "participant")
idiosync_removed
# "Last week while I was walking in the, I saw a firetruck by. It was red."
# "My dog loves to on walks in the every day of the week."
# "Where I live, it snows all winter long. It's so cold outside."
# "My kids to play in the. They to collect to build snowmen."
# "When I was younger, I used to visit my every week."
# "In the summertime, we would get together with my to bake cookies."
```

`idiosync_response_words`

Idiosyncratic Response Words

Description

This function identifies response words that are idiosyncratic (i.e. appear multiple times in a single response, and not in any other responses). It can also be used to remove these words.

Usage

```
idiosync_response_words(
  inputDataframe,
  mode,
  textColumnName,
  participantColumnName
)
```

Arguments

`inputDataframe`	A dataframe containing a column with text data (character strings)
`mode`	This defines the mode of operation. Options include "output", "remove", or "both". See Details below.

`textColumnName` A string consisting of the name of the column in `inputDataframe` which contains text data

`participantColumnName`
(Optional argument) A string consisting of the name of the column in `inputDataframe` which contains participant IDs

Details

This function has three modes: In the "output" mode, a dataframe is produced with three columns: the response with idiosyncratic words, the words, and how frequently they appear in that response. If a `participantColumnName` is provided, a fourth column with participant IDs is included. In the "remove" mode, a character string (or vector of character strings) is produced, where all of the idiosyncratic words are removed. In the "both" mode, both of the above results will be produced (i.e. a list containing a dataframe of idiosyncratic words, as well as the text with those words removed)

Value

A dataframe (`mode="output"`), a character string or vector of character strings (`mode="remove"`), or a two-object list containing both results (`mode="both"`)

See Also

`idiosync_participant_words`

Examples

```
myStrings = c("I like going to the park. The park is one of my favorite places to visit.",
              "Today is really rainy, but I'm a fan of this kind of weather to be honest.",
               "Yesterday, a bright red car with shiny red wheels drove past the house.")
mydataframe = data.frame(text=myStrings, stringsAsFactors = FALSE)
idiosync_output = idiosync_response_words(mydataframe, textColumnName = "text", mode = "output")
idiosync_output
# response_number     feature       frequency
# 1                   park          2
# 3                   red           2

idiosync_removed = idiosync_response_words(mydataframe, textColumnName = "text", mode = "remove")
idiosync_removed
# "I like going to the. The is one of my favorite places to visit."
# "Today is really rainy, but I'm a fan of this kind of weather to be honest."
# "Yesterday, a bright car with shiny wheels drove past the house."
```

---

langModel-class *langModel Class*

Description

langModel Class

Slots

`call` The function called to generate this model, with all arguments specified by the user

`data_text` The text input to create the corpus/model

`data_outcome` The outcome variable input to create the model

`type` Model type, "binary" or "continuous"

`text` The name of the column in the input dataframe containing the data_text

`outcome` The name of the column in the input dataframe containing the data_outcome

`tokens` The list of tokens in the language corpus

`ngrams` The ngrams used to generate the tokens

`dfmWeightScheme` The weight scheme used to create the document-frequency matrix

`x` The document-frequency matrix

`y` The dependent (outcome) variable

`cv` The final model

`lambda` The lambda value used

`predicted_y` The predicted outcomes based on the model and original language data

`predicted_probabilities` (If binary) The predicted probabilities of the outcomes based on the model and original language data

`roc` (If binary) The ROC calculated using the predicted_y

`roc_ci` (If binary) The boostrapped confidence interval calculated for the ROC

`corr` (If continuous) The correlation using the predicted_y

`level0` The bottom/first level of a binary variable, or the lowest value of a continuous variable

`level1` The top/second level of a binary variable, or the highest value of a continuous variable

`cat0raw` The predictors (word ngrams) predicting the level0 outcome, with their model weights

`cat1raw` The predictors (word ngrams) predicting the level1 outcome, with their model weights

`p_value` The p-value estimated via permutation test

`lossMeasure` The loss measure chosen for the LASSO regression cross-validation

`cvm_type` The loss measure - virtually always the same as lossMeasure except when the default "deviance" is specified

`cvm` The value of the loss measure for the cross-validated model at the chosen lambda level

`permuted_cvms` The value of the loss measure for each cross-validated model at the chosen lambda level across all randomized permutations

`permutationK` The number of permutations conducted for permutation testing

`minimum_p` The minimum p_value that can be achieved given the number permutationK specified

`st_err_p` The standard error of the p_value based on the number permutationK specified

language_model *Create Language Model*

Description

This function creates a regression model using input text as predictors, and a specified variable as the outcome.

Usage

```
language_model(
  input,
  outcome,
  outcomeType,
  text,
  ngrams = "1",
  dfmWeightScheme = "count",
  lossMeasure = "deviance",
  lambda = "lambda.min",
  parallelCores = NULL,
  permutePValue = FALSE,
  permutationK = 1000,
  permuteByGroup = NULL,
  progressBar = TRUE
)
```

Arguments

input	A dataframe containing a column with text data (character strings) and an outcome variable (numeric or two-level factor)
outcome	A string consisting of the column name for the outcome variable in inputDataframe
outcomeType	A string consisting of the type of outcome variable being used - options are "binary" or "continuous"
text	A string consisting of the column name for the text data in inputDataframe
ngrams	A string defining the ngrams to serve as predictors in the model. Defaults to "1". For more information, see the okens_ngrams function in the quanteda package
dfmWeightScheme	A string defining the weight scheme you wish to use for constructing a document-frequency matrix. Default is "count". For more information, see the dfm_weight function in the quanteda package
lossMeasure	A string defining the loss measure to use. Must be one of the options given by cv.glmnet. Default is "deviance".
lambda	A string defining the lambda value to be used. Default is "lambda.min". For more information, see the cv.glmnet function in the glmnet package
parallelCores	An integer defining the number of cores to use in parallel processing for model creation. Defaults to NULL (no parallel processing).

`permutePValue` If TRUE, a permutation test is run to estimate a p-value for the model (i.e. whether the language provided significantly predicts the outcome variable). Warning: this can take a while depending on the size of the dataset and number of permutations!

`permutationK` The number of permutations to run in a permutation test. Only used if `permutePValue = TRUE`. Defaults to 1000.

`permuteByGroup` A string consisting of the column name defining a grouping variable in the dataset (often a participant number). This means that when permutations are randomized, they will permute items on a group level rather than trial level. Default is NULL (no group variable considered).

`progressBar` Show a progress bar. Defaults to TRUE.

Details

This is the core function of the `languagePredictR` package. It largely follows the analysis laid out in Dobbins & Kantner 2019 (see References).

In the broadest terms, this serves as a wrapper for the quanteda (text analysis) and glmnet (modeling) packages.
The input text is converted into a document-frequency matrix (sometimes called a document-feature matrix) where each row represents a string of text, and each column represents a word that appears in the entire text corpus.
Each cell is populated by a value defined by the dfmWeightScheme. For example, the default, "count", means that each word column contains a value representing the number of times that word appears in the given text string.
This matrix is then used to train a regression algorithm appropriate to the outcome variable (standard linear regression for continuous variables, logistic regression for binary variables). See the documentation for the `cv.glmnet` function in the `glmnet` package for more information.
10-fold cross validation is currently implemented to reduce overfitting to the data.
Additionally, a LASSO constraint is used (following Tibshirani, 1996; see References) to eliminate weakly-predictive variables. This reduces the number of predictors (i.e. word engrams) to sparse, interpretable set.

Value

An object of the type "langModel"

References

Dobbins, I. G., & Kantner, J. (2019). The language of accurate recognition memory. *Cognition, 192*, 103988.
Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological), 58*(1), 267-288.

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)

# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence = language_model(movie_review_data1,
```

```
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText")

summary(movie_model_valence)

# Using language to predict 1-10 scale ratings,
# but using both unigrams and bigrams, as well as a proportion weighting scheme
movie_model_rating = language_model(movie_review_data1,
                                    outcomeV = "rating",
                                    outcomeType = "continuous",
                                    textC = "cleanText",
                                    ngrams = "1:2",
                                    dfmWeightScheme = "prop")

summary(movie_model_rating)

## End(Not run)
```

`lemmatize` *Lemmatize Text*

Description

This function performs lemmatization on input text by reducing words to their base units.

Usage

```
lemmatize(
  inputText,
  method = "direct",
  treetaggerDirectory,
  progressBar = TRUE
)
```

Arguments

`inputText`	A character string or vector of character strings
`method`	Either 'direct' (which uses a predefined list of words and their lemmas) or 'treetagger' (which uses the software `TreeTagger`, implemented through the `koRpus` package)
`treetaggerDirectory`	the filepath to the location of your installation of the `treetagger` library (See Details below)
`progressBar`	Show a progress bar. Defaults to TRUE.

Details

This function is essentially a wrapper for the `treetag` function from the `[koRpus]` package. In turn, koRpus implements the TreeTagger software package (available here: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/). The software must be downloaded

and installed on your local computer in order to use the `lemmatize` function. Once installed, the `treetaggerDirectory` argument should consist of the path where the software was installed.

This function performs "lemmatization," which is one form of reducing words to their most basic units. It is more thorough than "stemming," which only removes suffixes. E.g. for the words "walked" and "dogs," both lemmatization and stemming would reduce the words to "walk" and "dog." However, stemming would ignore "ran" and "geese," while lemmatization would properly render these "run" and "goose."

Value

A dataframe with lemmatized text, as well as columns with information about parts of speech

See Also

the `treetag` function from the `koRpus` package, as well as the treetagger documentation: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

Examples

```
myStrings = c("I walked in the park with both of my dogs.",
"The largest geese ran very fast.")
## Not run:
lemmatized_data = lemmatize(myStrings, "~/path/to/TreeTagger")
lemmatized_data$lemma_text
## End(Not run)
# "I walk in the park with both of my dog."
# "The large goose run very fast."
```

`lemma_data` *Lemma Data*

Description

A dataset containing inflected and base forms of text, used for the `lemmatize` function.

Usage

```
lemma_data
```

Format

A data frame with 47,366 rows and 2 variables:

inflected_form the original, inflected word (e.g. 'dogs' or 'walked')

lemma the base form of the word (e.g. 'dog' or 'walk')

Source

https://github.com/tm4ss/tm4ss.github.io/blob/master/resources/baseform_en.tsv

mild_movie_review_data *Mild Movie Reviews from IMDB*

Description

A dataset containing the text and ratings of 2000 movie reviews from IMDB. 1000 are mildly negative (rating of 4) and 1000 are mildly positive (rating of 7).

Usage

```
mild_movie_review_data
```

Format

A data frame with 2000 rows and 3 variables:

text text of the user's movie review

valence valence of the user's movie review, `Positive` (rating of 4) or `Negative` (rating of 7)

rating user's rating of the movie, on a scale of 1-10

Source

http://ai.stanford.edu/~amaas/data/sentiment/

modelAssessment-class *modelAssessment Class*

Description

modelAssessment Class

Slots

`type` Model type, "binary" or "continuous"

`model_labels` Label names and basic info for each model in the assessment

`auc_polygon_df` Dataframe for plotting the AUC polygon

`roc_ci_df` Dataframe for plotting the ROC confidence intervals

`roc_curve_df` Dataframe for plotting the ROC curves

`auc_ci_labels_df` Dataframe for AUC labels

`auc_tests` Dataframe of significance tests for AUCs

`cat_data` Dataframe of predictors (word engrams) with their model weights

---

`movie_review_data1` *Movie Reviews from IMDB*

Description

A dataset containing the text and ratings of 2000 movie reviews from IMDB. 1000 are negative (rating 1-4) and 1000 are positive (rating 7-10).

Usage

```
movie_review_data1
```

Format

A data frame with 2000 rows and 3 variables:

text text of the user's movie review

valence valence of the user's movie review, `Positive` (rating 1-4) or `Negative` (rating 7-10)

rating user's rating of the movie, on a scale of 1-10

Source

http://ai.stanford.edu/~amaas/data/sentiment/

`movie_review_data2` *Movie Reviews from IMDB*

Description

A dataset containing the text and ratings of 2000 movie reviews from IMDB. 1000 are negative (rating 1-4) and 1000 are positive (rating 7-10).

Usage

```
movie_review_data2
```

Format

A data frame with 2000 rows and 3 variables:

text text of the user's movie review

valence valence of the user's movie review, `Positive` (rating 1-4) or `Negative` (rating 7-10)

rating user's rating of the movie, on a scale of 1-10

Source

http://ai.stanford.edu/~amaas/data/sentiment/

node_edge *Create Node-Edge Table*

Description

Creates a table of nodes and edges based on a language corpus. Edge weights represent the average distance between two words, corrected by their frequency of appearance.

Usage

```
node_edge(
  input,
  maxDist = 4,
  removeStopwords = FALSE,
  binaryPenalty = FALSE,
  showProgress = TRUE
)
```

Arguments

input	Either a vector of strings, or a model generated by the language_model function
maxDist	The maximum distance to consider two words being co-occurent. Default is 4 (i.e. for "I went to the store," "I" and "store" are 4 words apart)
removeStopwords	If TRUE, words in quanteda stopwords function are excluded from analysis. Defaults to FALSE.
binaryPenalty	If TRUE, edge weights for each word pair will be penalized according to the edge weight of that word pair in the opposing text dataset. See Details.
showProgress	IF TRUE, progress bars are displayed. Defaults to TRUE.

Details

This function quantifies the relationship between words in the provided text.
It computes a measure of inverse average distance between word pairs, and then multiples that by a Dice coefficient (to control for frequency of occurrence and co-occurence of words)
Specifically, the formula used is:

$$weight = \frac{1}{\bar{D}} * \frac{2 * |X \cap Y|}{|X| + |Y|}$$

where:
$\bar{D}$ = mean distance between word X and word Y
$|X \cap Y|$ = number of co-occurences of word X and word Y
$|X|$ = number of occurences of word X across all pairs
$|Y|$ = number of occurences of word Y across all pairs

If a model predicting a binary outcome variable is provided (and thus two separate word networks will be plotted, for the text corresponding to each variable),

the `binaryPenalty` argument is available. This penalizes the edge weights for a given network by the strength of the edge weight for the same word pair in the opposing network. So if the word pair "my house" appears in the text for Outcome 0 with a weight of .33, and it also appears in the text for Outcome 1 with a weight of .21, applying the `binaryPenalty` would result in weights of `.33*(1 -.21)` and `.21*(1 -.33)`. If a word only appears in one Outcome text, its weight is unmodified.

The output dataframe will contain the following columns:
-**first** and **second** columns: the nodes specifying the two words of the pair
-**inverse_mean_distance**: the mean distance between the word pair, computed as an inverse to give greater weight to words that are closer together ($\frac{1}{D}$)
-**cooc_count**: the number of co-occurences of the **first** and **second** words ($|X \cap Y|$)
-**first_count**: the number of times the **first** word appears in a pair ($|X|$)
-**second_count**: the number of times the **first** word appears in a pair ($|Y|$)
-**weight**: the final weight, calculated as above

Value

A dataframe with node and edge weight information, along with occurence counts. See "Details."

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)

# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence = language_model(movie_review_data1,
                                     outcomeVariableColumnName = "valence",
                                     outcomeVariableType = "binary",
                                     textColumnName = "cleanText")

node_edge_table = node_edge(movie_model_valence)

## End(Not run)
```

`overview_plots` *Overview Plots*

Description

This function creates a set of plots showing basic information about a corpus of text data.

Usage

```
overview_plots(inputDataframe, textColumnName, participantColumnName)
```

Arguments

inputDataframe	A dataframe containing a column with text data (character strings)
textColumnName	A string consisting of the name of the column in inputDataframe which contains text data
participantColumnName	(Optional argument) A string consisting of the name of the column in inputDataframe which contains participant IDs

Details

If a participantColumnName is not provided, three graphs will be produced: -A pie chart with the total number of words in the provided corpus, divided into "Unique" words (those only used once), and "Repeated" words (those used at least twice). -A density plot with the length of each individual response in the corpus (in words) -A bar plot of the 25 most common words in the corpus, arranged by frequency

If a participantColumnName is provided, an additional two graphs will be produced: -The average number of words per response, plotted by participant (the overall average will be displayed as a vertical line) -The total number of words produced by each participant, compared to the total number of unique words produced by each participant

Value

Nothing (this function plots a series of graphs)

Examples

```
## Not run:
overview_plots(movie_review_data1, "text")

## End(Not run)
```

---

plot_cluster *Plot Individual Cluster*

Description

Plots individual clusters from the plot_word_network function. Can be helpful if these need to be plotted separately for any reason.

Usage

```
plot_cluster(network_input, cluster_number)
```

Arguments

network_input	The output of the plot_word_network function

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)

# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence = language_model(movie_review_data1,
                                     outcomeVariableColumnName = "valence",
                                     outcomeVariableType = "binary",
                                     textColumnName = "cleanText")

node_edge_table = node_edge(movie_model_valence)
network_output = plot_word_network(node_edge_table, clusterType="node")
plot_cluster(network_output[[1]], cluster_number=1)

## End(Not run)
```

`plot_predictor_words` *Plot Predictor Words*

Description

This function plots predictive words from the results of the `assess_models` function

Usage

```
plot_predictor_words(
  ...,
  topX,
  colors = c("blue", "orange"),
  plot_titles,
  model_names,
  xaxis_range,
  standard_xaxis = TRUE,
  flip_graphs = FALSE,
  print_individual = TRUE,
  print_summary = TRUE
)
```

Arguments

`...`	Output(s) of the `language_model` function
`topX`	The number of most-predictive words to plot
`colors`	A two-element vector containing the colors of the plotted bars. Defaults to c("blue", "orange")
`plot_titles`	A vector of titles for the plots
`model_names`	A vector of names for the individual models
`xaxis_range`	A maximum value for the x-axis
`standard_xaxis`	If TRUE, the x-axis on all graphs will be the same. If FALSE, it will adjust to fit each individual graph. Defaults to TRUE.

flip_graphs Flips the graphs horizontally. Defaults to FALSE (low-value outcome variable on the left, high-value outcome variable on the right)

print_individual
If TRUE, prints an individual graph for each model. Defaults to TRUE.

print_summary If TRUE, prints a summary graph with all models. Defaults to TRUE.

Value

Nothing (this function plots a series of graphs)

See Also

language_model

Examples

```
strong_movie_review_data$cleanText = clean_text(strong_movie_review_data$text)
mild_movie_review_data$cleanText = clean_text(mild_movie_review_data$text)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for strong reviews (ratings of 1 or 10)
movie_model_strong = language_model(strong_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for mild reviews (ratings of 4 or 7)
movie_model_mild = language_model(mild_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)

# Analyze ROC curves
plot_predictor_words(movie_model_strong, movie_model_mild)
```

---

plot_roc *Plot ROC curves*

Description

This function plots ROC curves from the results of the assess_models function

Usage

```
plot_roc(
  ...,
  individual_plot = TRUE,
  combined_plot = TRUE,
  facet_plot = TRUE,
```

```
  facet_summary = TRUE,
  colors,
  model_names,
  plot_auc_polygon = TRUE,
  plot_ci = TRUE,
  line_size = 1,
  print_auc = TRUE,
  print_ci = TRUE,
  print_auc_ci_font_size = 4,
  print_auc_ci_x,
  print_auc_ci_y,
  plot_legend = TRUE,
  plot_title,
  facet_n_row = NULL,
  facet_n_col = 2
)
```

Arguments

...	Output(s) of the language_model, comparison_model, or test_language_model functions
individual_plot	If TRUE, graphs individual ROC curves for each model. Defaults to TRUE.
combined_plot	If TRUE, and modelAssessment contains multiple models, graphs a plot with all ROC curves overlapping. Defaults to TRUE.
facet_plot	If TRUE, and modelAssessment contains multiple models, graphs a faceted plot with all ROC curves included. Defaults to TRUE.
facet_summary	If TRUE, and modelAssessment contains multiple models, the facet_plot will include a plot with all ROC curves overlapping. Defaults to TRUE.
colors	A vector of colors to use for each model's ROC curve.
model_names	A vector of strings to use as titles/names for each model.
plot_auc_polygon	If TRUE, the area below with ROC curve with the lowest AUC will be shaded in. Defaults to TRUE.
plot_ci	If TRUE, a confidence band will be plotted around each ROC curve. Defaults to TRUE.
line_size	A numeric representing the width of the ROC curve line. Defaults to 1.
print_auc	If TRUE, the value of the AUC will be printed on the plot. Defaults to TRUE.
print_ci	If TRUE, the range of the confidence interval will be printed on the plot. Defaults to TRUE.
print_auc_ci_font_size	The font size for printed values for the AUC and confidence interval. Defaults to 4.
print_auc_ci_x	A vector of x (horizontal) positions determining where on the plot the AUC and confidence interval values will be printed.
print_auc_ci_y	A vector of y (vertical) positions determining where on the plot the AUC and confidence interval values will be printed.

plot_legend	If TRUE, a legend will be printed on all plots.
plot_title	The title of the plot
facet_n_row	The number of rows used to plot the facet_plot. Defaults to NULL.
facet_n_col	The number of columns used to plot the facet_plot. Defaults to 2.

Value

Nothing (this function plots a series of graphs)

See Also

language_model, comparison_model, test_language_model

Examples

```
## Not run:
strong_movie_review_data$cleanText = clean_text(strong_movie_review_data$text)
mild_movie_review_data$cleanText = clean_text(mild_movie_review_data$text)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for strong reviews (ratings of 1 or 10)
movie_model_strong = language_model(strong_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)

# Using language to predict "Positive" vs. "Negative" reviews
# Only for mild reviews (ratings of 4 or 7)
movie_model_mild = language_model(mild_movie_review_data,
                                    outcome = "valence",
                                    outcomeType = "binary",
                                    text = "cleanText",
                                    progressBar = FALSE)


# Plot ROC curves
plot_roc(movie_model_strong, movie_model_mild)

## End(Not run)
```

---

plot_word_network *Plot Word Network*

Description

Plots a word network of adjacent words.

Usage

```
plot_word_network(
  input,
  model = NULL,
  topX = 100,
  graphIndividual = TRUE,
  graphCombined = FALSE,
  directed = FALSE,
  removeVerticesBelowDegree = 2,
  clusterType = "none",
  clusterNodeMethod = "infomap",
  plotUnclusteredNetwork = TRUE,
  plotClusteredNetwork = TRUE,
  plotIndividualClusters = TRUE,
  plotIndividualClusterFacet = TRUE,
  plotClusterLegend = TRUE,
  edgeColor = "darkgray",
  edgeAlpha = 0.5,
  edgeCurve = 0.15,
  modelNodeColors = c("lightblue", "orange"),
  modelNodeSizeRange = c(5, 10),
  nodeLabelSize = 1,
  nodeLabelColor = "black",
  plotTitle = NULL,
  layout = NULL
)
```

Arguments

input	An input dataframe, typically the output from the node_edge function
model	Optional - if node_edge used a model as input, the same model can be provided here for extra functionality
topX	The number of word pairs to include in the graphed network. Chosen word pairs are selected from those with the greatest number of co-occurrences. Defaults to 100.
graphIndividual	If TRUE, individual graphs are produced for both outcomes of the model. Default is TRUE.
graphCombined	If TRUE, a network is graphed based on the entire language corpus. Default is FALSE.
directed	Determines if the network is directed (direction of edges matters) or not. Defaults to FALSE (the output from node_edge does not yield directional edge information, so only change this if using your own dataframe).
removeVerticesBelowDegree	An integer which determines the minimum number of edges a node must have to be included. Default is 2.
clusterType	The type of clustering to perform. "node" clusters by nodes (using the method defined by clusterNodeMethod), "edge" clusters by edges (using the lincomm package. Defaults to "none".

`clusterNodeMethod`
: If clustering by "node", this determines the method used. Options are the same clustering options given in the `igraph` package.

`plotUnclusteredNetwork`
: If TRUE, the network is plotted with no clustering displayed. Defaults to TRUE.

`plotClusteredNetwork`
: If TRUE, the network is plotted with clustering displayed (shaded regions for "node" clustering, colored edges for "edge" clustering). Defaults to TRUE.

`plotIndividualClusters`
: If TRUE, each cluster is plotted in a separate graph. Defaults to TRUE.

`plotIndividualClusterFacet`
: If TRUE, each cluster is plotted in a faceted section of a single graph. Defaults to TRUE.

`plotClusterLegend`
: If TRUE, plots a legend with cluster numbers and corresponding colors. Defaults to TRUE.

`edgeColor`
: The color of the edges. Default is "darkgray".

`edgeAlpha`
: The alpha of the edges. Default is 0.5.

`edgeCurve`
: If greater than 0, edges will be curved with a radius corresponding to the value. Default is 0.15. A value of 0 yields straight edges.

`modelNodeColors`
: The color shading for nodes that are predictive words in the provided model. Must be a vector of two values. Defaults to c("lightblue", "orange").

`modelNodeSizeRange`
: The sizing for nodes that are predictive words in the provided model. Must be a vector of two values (the minimum plotted size and maximum plotted size). Defaults to c(5, 10).

`nodeLabelSize`
: The size of the text for node labels. Defaults to 1.

`nodeLabelColor`
: The color of the node labels. Defaults to "black".

`plotTitle`
: The title of the plot(s). If a model is used, it must be a vector of three strings. If not, it must be a single string.

`layout`
: An `igraph` layout object, helpful if you want to re-graph the same network multiple times with the same layout

Value

A list containing the `igraph` network object, `igraph` layout, and clustering data (if applicable) - this can be used with the `plot_cluster` function, or graphed manually with the `igraph` package

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)

# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence = language_model(movie_review_data1,
                                     outcomeVariableColumnName = "valence",
```

```
                                    outcomeVariableType = "binary",
                                    textColumnName = "cleanText")

node_edge_table = node_edge(movie_model_valence)
plot_word_network(node_edge_table)

## End(Not run)
```

`strong_movie_review_data` *Strong Movie Reviews from IMDB*

Description

A dataset containing the text and ratings of 2000 movie reviews from IMDB. 1000 are strongly negative (rating of 1) and 1000 are strongly positive (rating of 10).

Usage

```
strong_movie_review_data
```

Format

A data frame with 2000 rows and 3 variables:

text text of the user's movie review

valence valence of the user's movie review, `Positive` (rating of 1) or `Negative` (rating of 10)

rating user's rating of the movie, on a scale of 1-10

Source

http://ai.stanford.edu/~amaas/data/sentiment/

`summary.compModel` *Summary (compModel)*

Description

Summary (compModel)

Usage

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

Arguments

`object`	The compModel object to summarize
`...`	Additional arguments

`summary.langModel` *Summary (langModel)*

Description

Summary (langModel)

Usage

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

Arguments

`object`	The langModel object to summarize
`...`	Additional arguments

`summary.testAssessment` *Summary (testAssessment)*

Description

Summary (testAssessment)

Usage

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

Arguments

`object`	The testAssessment object to summarize
`...`	Additional arguments

`testAssessment-class` *testAssessment Class*

Description

testAssessment Class

Slots

`call` The function called to generate this model, with all arguments specified by the user

`data_text` The text input to test the model

`data_outcome` The outcome variable input to test the model

`type` Model type, "binary" or "continuous"

`text` The name of the column in the test dataframe containing the data_text

`outcome` The name of the column in the test dataframe containing the data_outcome

`ngrams` The ngrams used to generate the tokens

`dfmWeightScheme` The weight scheme used to create the document-frequency matrix

`x` The document-frequency matrix

`y` The dependent (outcome) variable

`predicted_y` The predicted outcomes based on the model and test data

`predicted_probabilities` (If binary) The predicted probabilities of the outcomes based on the model and test data

`roc` (If binary) The ROC calculated using the predicted_y

`roc_ci` (If binary) The boostrapped confidence interval calculated for the ROC

`corr` (If continuous) The correlation using the predicted_y

`level0` The bottom/first level of a binary variable, or the lowest value of a continuous variable

`level1` The top/second level of a binary variable, or the highest value of a continuous variable

`trainedModel` The name of the model used for the test

`original_predictive_ngrams` The list of ngram predictors from the model

`ngrams_present` The number of original_predictive_ngrams that appear in the test lanuage sample

`test_language_model` *Test Language Model*

Description

This function tests a model created by the `language_model` function on a new dataset

Usage

```
test_language_model(
  input,
  outcome,
  text,
  trainedModel,
  ngrams = "1",
  dfmWeightScheme = "count",
  progressBar = TRUE
)
```

Arguments

`input`	A dataframe containing a column with text data (character strings) and an outcome variable (numeric or two-level factor)
`outcome`	A string consisting of the column name for the outcome variable in `inputDataframe`
`text`	A string consisting of the column name for the text data in `inputDataframe`
`trainedModel`	A trained model created by the `language_model` function
`ngrams`	A string defining the ngrams to serve as predictors in the model. Defaults to "1". For more information, see the `okens_ngrams` function in the `quanteda` package
`dfmWeightScheme`	A string defining the weight scheme you wish to use for constructing a document-frequency matrix. Default is "count". For more information, see the `dfm_weight` function in the `quanteda` package
`progressBar`	Show a progress bar. Defaults to TRUE.

Details

This function is effectively a special version of the `language_model` function. Instead of creating a new model, the outputs are based on the results of testing a new, independent dataset using an existing model. This allows for assessing how well a trained language model generalizes to other inputs - this function allows for comparisons between the models using many of the same functions that can be used with `language_model`.

Value

An object of the type "testAssessment"

See Also

`language_model`

Examples

```
## Not run:
movie_review_data1$cleanText = clean_text(movie_review_data1$text)
movie_review_data2$cleanText = clean_text(movie_review_data2$text)

# Train a model on the \code{movie_review_data1} dataset
# Using language to predict "Positive" vs. "Negative" reviews
movie_model_valence = language_model(movie_review_data1,
                                     outcome = "valence",
                                     outcomeType = "binary",
                                     text = "cleanText")

# Test the model on the \code{movie_review_data2} dataset
movie_model_valence_test = test_language_model(movie_review_data2,
                                    outcome = "valence",
                                    text = "cleanText",
                                    trainedModel = movie_model_valence)
summary(movie_model_valence_test)

## End(Not run)
```

Index