Package 'text'

April 23, 2024

Type Package

Title Analyses of Text using Transformers Models from HuggingFace, Natural Language Processing and Machine Learning

Version 1.2.1

Description Link R with Transformers from Hugging Face to transform text variables to word embeddings; where the word embeddings are used to statistically test the mean difference between set of texts, compute semantic similarity scores between texts, predict numerical variables, and visual statistically significant words according to various dimensions etc. For more information see <https://www.r-text.org>.

License GPL-3

URL https://r-text.org/, https://github.com/OscarKjell/text/

BugReports https://github.com/OscarKjell/text/issues/

Encoding UTF-8

Archs x64

SystemRequirements Python (>= 3.6.0)

LazyData true

BuildVignettes true

Imports dplyr, tibble, stringi, tidyr, ggplot2, ggrepel, cowplot, rlang, purrr, magrittr, parsnip, recipes (>= 0.1.16), rsample, reticulate, tune, workflows, yardstick, future, furrr, overlapping

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat, rio, glmnet, randomForest, covr, xml2, ranger, utils, ggwordcloud, reactable

VignetteBuilder knitr

Depends R (>= 4.00)

NeedsCompilation no

Author Oscar Kjell [aut, cre] (<https://orcid.org/0000-0002-2728-6278>), Salvatore Giorgi [aut] (<https://orcid.org/0000-0001-7381-6295>), Andrew Schwartz [aut] (<https://orcid.org/0000-0002-6383-3339>)

Maintainer Oscar Kjell <oscar.kjell@psy.lu.se>

Repository CRAN

Date/Publication 2024-04-23 00:00:03 UTC

R topics documented:

`centrality_data_harmony` *Example data for plotting a Semantic Centrality Plot.*

Description

The dataset is a shortened version of the data sets of Study 1 from Kjell, et al., 2016.

Usage

```
centrality_data_harmony
```

Format

A data frame with 2,146 and 4 variables:

words unique words

n overall word frequency

central_semantic_similarity cosine semantic similarity to the aggregated word embedding

n_percent frequency in percent

Source

https://link.springer.com/article/10.1007/s11205-015-0903-z

DP_projections_HILS_SWLS_100 *Data for plotting a Dot Product Projection Plot.*

Description

Tibble is the output from textProjection. The dataset is a shortened version of the data sets of Study 3-5 from Kjell, Kjell, Garcia and Sikström 2018.

Usage

```
DP_projections_HILS_SWLS_100
```

Format

A data frame with 583 rows and 12 variables:

words unique words

dot.x dot product projection on the x-axes

p_values_dot.x p-value for the word in relation to the x-axes

n_g1.x frequency of the word in group 1 on the x-axes variable

n_g2.x frequency of the word in group 2 on the x-axes variable

dot.y dot product projection on the y-axes

p_values_dot.y p-value for the word in relation to the y-axes

n_g1.y frequency of the word in group 1 on the y-axes variable

n_g2.y frequency of the word in group 2 on the x-axes variable

n overall word frequency

n.percent frequency in percent

N_participant_responses number of participants (as this is needed in the analyses)

Source

https://osf.io/preprints/psyarxiv/er6t7/

`Language_based_assessment_data_3_100`

Example text and numeric data.

Description

The dataset is a shortened version of the data sets of Study 3-5 from Kjell, Kjell, Garcia and Sikström 2018.

Usage

```
Language_based_assessment_data_3_100
```

Format

A data frame with 100 rows and 4 variables:

harmonywords Word responses from the harmony in life word question

hilstotal total score of the Harmony In Life Scale

swlstotal total score of the Satisfaction With Life Scale

Source

https://osf.io/preprints/psyarxiv/er6t7/

`Language_based_assessment_data_8`

Text and numeric data for 10 participants.

Description

The dataset is a shortened version of the data sets of Study 3-5 from Kjell et al., (2018; https://psyarxiv.com/er6t7/).

Usage

```
Language_based_assessment_data_8
```

Format

A data frame with 40 participants and 8 variables:

harmonywords descriptive words where respondents describe their harmony in life

satisfactionwords descriptive words where respondents describe their satisfaction with life

harmonytexts text where respondents describe their harmony in life

satisfactiontexts text where respondents describe their satisfaction with life

hilstotal total score of the Harmony In Life Scale

swlstotal total score of the Satisfaction With Life Scale

age respondents age in years

gender respondents gender 1=male, 2=female

Source

https://osf.io/preprints/psyarxiv/er6t7/

PC_projections_satisfactionwords_40

Example data for plotting a Principle Component Projection Plot.

Description

The dataset is a shortened version of the data sets of Study 1 from Kjell, et al., 2016.

Usage

```
PC_projections_satisfactionwords_40
```

Format

A data frame.

words unique words

n overall word frequency

Dim_PC1 Principle component value for dimension 1

Dim_PC2 Principle component value for dimension 2

Source

https://link.springer.com/article/10.1007/s11205-015-0903-z

raw_embeddings_1 *Word embeddings from textEmbedRawLayers function*

Description

The dataset is a shortened version of the data sets of Study 3-5 from Kjell, Kjell, Garcia and Sikström 2018.

Usage

```
raw_embeddings_1
```

Format

A list with token-level word embeddings for harmony words.

tokens words

layer_number layer of the transformer model

Dim1:Dim8 Word embeddings dimensions

Source

https://osf.io/preprints/psyarxiv/er6t7/

textCentrality *Compute semantic similarity score between single words' word embeddings and the aggregated word embedding of all words.*

Description

Compute semantic similarity score between single words' word embeddings and the aggregated word embedding of all words.

Usage

```
textCentrality(
  words,
  word_embeddings,
  word_types_embeddings = word_types_embeddings_df,
  method = "cosine",
  aggregation = "mean",
  min_freq_words_test = 0
)
```

Arguments

words	(character) Word or text variable to be plotted.
word_embeddings	Word embeddings from textEmbed for the words to be plotted (i.e., the aggregated word embeddings for the "words" variable).
word_types_embeddings	Word embeddings from textEmbed for individual words (i.e., the decontextualized word embeddings).
method	(character) Character string describing type of measure to be computed. Default is "cosine" (see also "spearmen", "pearson" as well as measures from textDistance() (which here is computed as 1 - textDistance) including "euclidean", "maximum", "manhattan", "canberra", "binary" and "minkowski").
aggregation	(character) Method to aggregate the word embeddings (default = "mean"; see also "min", "max" or "[CLS]").
min_freq_words_test	(numeric) Option to select words that have at least occurred a specified number of times (default = 0); when creating the semantic similarity scores.

Value

A dataframe with variables (e.g., including semantic similarity, frequencies) for the individual words that are used as input for the plotting in the textCentralityPlot function.

See Also

See textCentralityPlot and textProjection.

Examples

```
# Computes the semantic similarity between the individual word embeddings (Iwe)
# in the "harmonywords" column of the pre-installed dataset: Language_based_assessment_data_8,
# and the aggregated word embedding (Awe).
# The Awe can be interpreted the latent meaning of the text.

## Not run:
df_for_plotting <- textCentrality(
  words = Language_based_assessment_data_8["harmonywords"],
  word_embeddings = word_embeddings_4$texts$harmonywords,
  word_types_embeddings = word_embeddings_4$word_types
)

# df_for_plotting contain variables (e.g., semantic similarity, frequencies) for
# the individual words that are used for plotting by the textCentralityPlot function.

## End(Not run)
```

`textCentralityPlot` *Plot words according to semantic similarity to the aggregated word embedding.*

Description

Plot words according to semantic similarity to the aggregated word embedding.

Usage

```
textCentralityPlot(
  word_data,
  min_freq_words_test = 1,
  plot_n_word_extreme = 10,
  plot_n_word_frequency = 10,
  plot_n_words_middle = 10,
  titles_color = "#61605e",
  x_axes = "central_semantic_similarity",
  title_top = "Semantic Centrality Plot",
  x_axes_label = "Semantic Centrality",
  scale_x_axes_lim = NULL,
  scale_y_axes_lim = NULL,
  word_font = NULL,
  centrality_color_codes = c("#EAEAEA", "#85DB8E", "#398CF9", "#9e9d9d"),
  word_size_range = c(3, 8),
  position_jitter_hight = 0,
  position_jitter_width = 0.03,
  point_size = 0.5,
  arrow_transparency = 0.1,
  points_without_words_size = 0.5,
  points_without_words_alpha = 0.5,
  legend_title = "SC",
  legend_x_axes_label = "x",
  legend_x_position = 0.02,
  legend_y_position = 0.02,
  legend_h_size = 0.2,
  legend_w_size = 0.2,
  legend_title_size = 7,
  legend_number_size = 2,
  seed = 1007
)
```

Arguments

`word_data`	Tibble from the textPlot function.

`min_freq_words_test`
: Select words to significance test that have occurred at least min_freq_words_test (default = 1).

`plot_n_word_extreme`
: Number of words per dimension to plot with extreme Supervised Dimension Projection value (default = 10). (i.e., even if not significant; duplicates are removed).

`plot_n_word_frequency`
: Number of words to plot according to their frequency (default = 10). (i.e., even if not significant).

`plot_n_words_middle`
: Number of words to plot that are in the middle in Supervised Dimension Projection score (default = 10). (i.e., even if not significant; duplicates are removed).

`titles_color`
: Color for all the titles (default: "#61605e").

`x_axes`
: Variable to be plotted on the x-axes (default: "central_semantic_similarity", could also select "n", "n_percent").

`title_top`
: Title (default: " ").

`x_axes_label`
: Label on the x-axes (default: "Semantic Centrality").

`scale_x_axes_lim`
: Length of the x-axes (default: NULL, which uses c(min(word_data$central_semantic_similarity)-0.05, max(word_data$central_semantic_similarity)+0.05); change this by e.g., try c(-5, 5)).

`scale_y_axes_lim`
: Length of the y-axes (default: NULL, which uses c(-1, 1); change e.g., by trying c(-5, 5)).

`word_font`
: Type of font (default: NULL).

`centrality_color_codes`
: (HTML color codes. type = character) Colors of the words selected as plot_n_word_extreme (minimum values), plot_n_words_middle, plot_n_word_extreme (maximum values) and plot_n_word_frequency; the default is c("#EAEAEA", "#85DB8E", "#398CF9", "#9e9d9d", respectively.

`word_size_range`
: Vector with minimum and maximum font size (default: c(3, 8)).

`position_jitter_hight`
: Jitter height (default: .0).

`position_jitter_width`
: Jitter width (default: .03).

`point_size`
: Size of the points indicating the words' position (default: 0.5).

`arrow_transparency`
: Transparency of the lines between each word and point (default: 0.1).

`points_without_words_size`
: Size of the points not linked to a word (default is to not show the point; , i.e., 0).

`points_without_words_alpha`
: Transparency of the points that are not linked to a word (default is to not show it; i.e., 0).

legend_title Title of the color legend (default: "SCP").

legend_x_axes_label
Label on the color legend (default: "x").

legend_x_position
Position on the x coordinates of the color legend (default = 0.02).

legend_y_position
Position on the y coordinates of the color legend (default = 0.05).

legend_h_size Height of the color legend (default = 0.15).

legend_w_size Width of the color legend (default = 0.15).

legend_title_size
Font size of the title (default = 7).

legend_number_size
Font size of the values in the legend (default = 2).

seed Set different seed (default = 1007).

Value

A 1-dimensional word plot based on similarity to the aggregated word embedding, as well as tibble with processed data used to plot.

See Also

See textCentrality and textProjection.

Examples

```
# Plot a centrality plot from the dataframe df_for_plotting
# that is returned by the textCentrality function.
## Not run:
textCentralityPlot(
  df_for_plotting,
  min_freq_words_test = 1,
  plot_n_word_extreme = 10,
  plot_n_word_frequency = 10,
  plot_n_words_middle = 10,
  titles_color = "#61605e",
  x_axes = "central_semantic_similarity",
  title_top = "Semantic Centrality Plot",
  x_axes_label = "Semantic Centrality",
  scale_x_axes_lim = NULL,
  scale_y_axes_lim = NULL,
  word_font = NULL,
  centrality_color_codes = c("#EAEAEA", "#85DB8E", "#398CF9", "#9e9d9d"),
  word_size_range = c(3, 8),
  position_jitter_hight = 0,
  position_jitter_width = 0.03,
  point_size = 0.5,
  arrow_transparency = 0.1,
  points_without_words_size = 0.5,
```

```
  points_without_words_alpha = 0.5,
  legend_title = "SC",
  legend_x_axes_label = "x",
  legend_x_position = 0.02,
  legend_y_position = 0.02,
  legend_h_size = 0.2,
  legend_w_size = 0.2,
  legend_title_size = 7,
  legend_number_size = 2,
  seed = 1007
)

## End(Not run)
```

`textClassify` *Predict label and probability of a text using a pretrained classifier language model. (experimental)*

Description

Predict label and probability of a text using a pretrained classifier language model. (experimental)

Usage

```
textClassify(
  x,
  model = "distilbert-base-uncased-finetuned-sst-2-english",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "error",
  return_incorrect_results = FALSE,
  return_all_scores = FALSE,
  function_to_apply = "none",
  set_seed = 202208
)
```

Arguments

`x`	(string) A character variable or a tibble/dataframe with at least one character variable.
`model`	(string) Specification of a pre-trained classifier language model. For full list of options see pretrained classifier models at HuggingFace. For example use "cardiffnlp/twitter-roberta-base-sentiment", "distilbert-base-uncased-finetuned-sst-2-english".
`device`	(string) Device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number.

`tokenizer_parallelism`
: (boolean) If TRUE this will turn on tokenizer parallelism.

`logging_level`
: (string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug

`return_incorrect_results`
: (boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").

`return_all_scores`
: (boolean) Whether to return all prediction scores or just the one of the predicted class.

`function_to_apply`
: (string) The function to apply to the model outputs to retrieve the scores.

`set_seed`
: (Integer) Set seed. There are four different values: "default": if the model has a single label, will apply the sigmoid function on the output. If the model has several labels, the softmax function will be applied on the output. "sigmoid": Applies the sigmoid function on the output. "softmax": Applies the softmax function on the output. "none": Does not apply any function on the output.

Value

A tibble with predicted labels and scores for each text variable. The comment of the object show the model-name and computation time.

See Also

see `textGeneration`, `textNER`, `textSum`, `textQA`, `textTranslate`

Examples

```
# classifications <- textClassify(x = Language_based_assessment_data_8[1:2, 1:2])
# classifications
# comment(classifications)
```

---

textDescriptives *Compute descriptive statistics of character variables.*

Description

Compute descriptive statistics of character variables.

Usage

```
textDescriptives(
  words,
  compute_total = TRUE,
  entropy_unit = "log2",
  na.rm = TRUE,
  locale = "en_US"
)
```

Arguments

words	One or several character variables; if its a tibble or dataframe, all the character variables will be selected.
compute_total	Boolean. If the input (words) is a tibble/dataframe with several character variables, a total variable is computed.
entropy_unit	The unit entropy is measured in. The default is to used bits (i.e., log2; see also, "log", "log10"). If a total score for several varaibles is computed,the text columns are combined using the dplyr unite function. For more information about the entropy see the entropy package and specifically its entropy.plugin function.
na.rm	Option to remove NAs when computing mean, median etc (see under return).
locale	(character string) Locale Identifiers for example in US-English ('en_US') and Australian-English ('en_AU'); see help(about_locale) in the stringi package

Value

A tibble with descriptive statistics, including variable = the variable names of input "words"; w_total = total number of words in the variable; w_mean = mean number of words in each row of the variable; w_median = median number of words in each row of the variable; w_range_min = smallest number of words of all rows; w_range_max = largest number of words of all rows; w_sd = the standard deviation of the number of words of all rows; unique_tokens = the unique number of tokens (using the word_tokenize function from python package nltk) n_token = number of tokens in the variable (using the word_tokenize function from python package nltk) entropy = the entropy of the variable. It is computed as the Shannon entropy H of a discrete random variable from the specified bin frequencies. (see library entropy and specifically the entropy.plugin function)

See Also

see textEmbed

Examples

```
## Not run:
textDescriptives(Language_based_assessment_data_8[1:2])

## End(Not run)
```

textDimName *Change the names of the dimensions in the word embeddings.*

Description

Change the names of the dimensions in the word embeddings.

Usage

```
textDimName(word_embeddings, dim_names = TRUE)
```

Arguments

word_embeddings	List of word embeddings
dim_names	(boolean) If TRUE the word embedding name will be attached to the name of each dimension; is FALSE, the attached part of the name will be removed.

Value

Word embeddings with changed names.

See Also

see textEmbed

Examples

```
# Note that dimensions are called Dim1_harmonytexts etc.
word_embeddings_4$texts$harmonytexts
# Here they are changed to just Dim
w_e_T <- textDimName(word_embeddings_4$texts["harmonytexts"],
  dim_names = FALSE
)
# Here they are changed back
w_e_F <- textDimName(w_e_T, dim_names = TRUE)
```

`textDistance` *Compute the semantic distance between two text variables.*

Description

Compute the semantic distance between two text variables.

Usage

```
textDistance(x, y, method = "euclidean", center = FALSE, scale = FALSE)
```

Arguments

`x`	Word embeddings (from textEmbed).
`y`	Word embeddings (from textEmbed).
`method`	(character) Character string describing type of measure to be computed; default is "euclidean" (see also measures from stats:dist() including "maximum", "manhattan", "canberra", "binary" and "minkowski". It is also possible to use "cosine", which computes the cosine distance (i.e., 1 - cosine(x, y)).
`center`	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the embedding mean (omitting NAs) of x from each of its dimension, and if center is FALSE, no centering is done.
`scale`	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) embedding dimensions by the standard deviation of the embedding if center is TRUE, and the root mean square otherwise.

Value

A vector comprising semantic distance scores.

See Also

See `textSimilarity` and `textSimilarityNorm`.

Examples

```
# Compute the semantic distance score between the embeddings
# from "harmonytext" and "satisfactiontext".

## Not run:
distance_scores <- textDistance(
  x = word_embeddings_4$texts$harmonytext,
  y = word_embeddings_4$texts$satisfactiontext
)

# Show information about how distance_scores were constructed.
```

```
comment(distance_scores)

## End(Not run)
```

`textDistanceMatrix` *Compute semantic distance scores between all combinations in a word embedding*

Description

Compute semantic distance scores between all combinations in a word embedding

Usage

```
textDistanceMatrix(x, method = "euclidean", center = FALSE, scale = FALSE)
```

Arguments

`x`	Word embeddings (from textEmbed).
`method`	(character) Character string describing type of measure to be computed; default is "euclidean" (see also measures from stats:dist() including "maximum", "manhattan", "canberra", "binary" and "minkowski". It is also possible to use "cosine", which computes the cosine distance (i.e., 1 - cosine(x, y)).
`center`	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the embedding mean (omitting NAs) of x from each of its dimension, and if center is FALSE, no centering is done.
`scale`	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) embedding dimensions by the standard deviation of the embedding if center is TRUE, and the root mean square otherwise.

Value

A matrix of semantic distance scores

See Also

see `textDistanceNorm`

Examples

```
distance_scores <- textDistanceMatrix(word_embeddings_4$texts$harmonytext[1:3, ])
round(distance_scores, 3)
```

`textDistanceNorm` *Compute the semantic distance between a text variable and a word norm (i.e., a text represented by one word embedding that represent a construct/concept).*

Description

Compute the semantic distance between a text variable and a word norm (i.e., a text represented by one word embedding that represent a construct/concept).

Usage

```
textDistanceNorm(x, y, method = "euclidean", center = FALSE, scale = FALSE)
```

Arguments

`x`	Word embeddings (from textEmbed).
`y`	Word embedding from textEmbed (from only one text).
`method`	(character) Character string describing type of measure to be computed; default is "euclidean" (see also measures from stats:dist() including "maximum", "manhattan", "canberra", "binary" and "minkowski". It is also possible to use "cosine", which computes the cosine distance (i.e., 1 - cosine(x, y)).
`center`	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the embedding mean (omitting NAs) of x from each of its dimension, and if center is FALSE, no centering is done.
`scale`	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) embedding dimensions by the standard deviation of the embedding if center is TRUE, and the root mean square otherwise.

Value

A vector comprising semantic distance scores.

See Also

see `textDistance`

Examples

```
## Not run:
library(dplyr)
library(tibble)
harmonynorm <- c("harmony peace ")
satisfactionnorm <- c("satisfaction achievement")

norms <- tibble::tibble(harmonynorm, satisfactionnorm)
word_embeddings <- word_embeddings_4$texts
```

```
word_embeddings_wordnorm <- textEmbed(norms)
similarity_scores <- textDistanceNorm(
  word_embeddings$harmonytext,
  word_embeddings_wordnorm$harmonynorm
)

## End(Not run)
```

`textEmbed` *Extract layers and aggregate them to word embeddings, for all character variables in a given dataframe.*

Description

Extract layers and aggregate them to word embeddings, for all character variables in a given dataframe.

Usage

```
textEmbed(
  texts,
  model = "bert-base-uncased",
  layers = -2,
  dim_name = TRUE,
  aggregation_from_layers_to_tokens = "concatenate",
  aggregation_from_tokens_to_texts = "mean",
  aggregation_from_tokens_to_word_types = NULL,
  keep_token_embeddings = TRUE,
  tokens_select = NULL,
  tokens_deselect = NULL,
  decontextualize = FALSE,
  model_max_length = NULL,
  max_token_to_sentence = 4,
  tokenizer_parallelism = FALSE,
  device = "cpu",
  hg_gated = FALSE,
  hg_token = Sys.getenv("HUGGINGFACE_TOKEN", unset = ""),
  logging_level = "error",
  ...
)
```

Arguments

`texts`	A character variable or a tibble/dataframe with at least one character variable.
`model`	Character string specifying pre-trained language model (default 'bert-base-uncased'). For full list of options see pretrained models at HuggingFace. For example use "bert-base-multilingual-cased", "openai-gpt", "gpt2", "ctrl", "transfo-xl-wt103", "xlnet-base-cased", "xlm-mlm-enfr-1024", "distilbert-base-cased", "roberta-base",

or "xlm-roberta-base". Only load models that you trust from HuggingFace; loading a malicious model can execute arbitrary code on your computer).

`layers` (string or numeric) Specify the layers that should be extracted (default -2 which give the second to last layer). It is more efficient to only extract the layers that you need (e.g., 11). You can also extract several (e.g., 11:12), or all by setting this parameter to "all". Layer 0 is the decontextualized input layer (i.e., not comprising hidden states) and thus should normally not be used. These layers can then be aggregated in the textEmbedLayerAggregation function.

`dim_name` (boolean) If TRUE append the variable name after all variable-names in the output. (This differentiates between word embedding dimension names; e.g., Dim1_text_variable_name). see `textDimName` to change names back and forth.

`aggregation_from_layers_to_tokens`
(string) Aggregated layers of each token. Method to aggregate the contextualized layers (e.g., "mean", "min" or "max, which takes the minimum, maximum or mean, respectively, across each column; or "concatenate", which links together each word embedding layer to one long row.

`aggregation_from_tokens_to_texts`
(string) Method to carry out the aggregation among the word embeddings for the words/tokens, including "min", "max" and "mean" which takes the minimum, maximum or mean across each column; or "concatenate", which links together each layer of the word embedding to one long row (default = "mean"). If set to NULL, embeddings are not aggregated.

`aggregation_from_tokens_to_word_types`
(string) Aggregates to the word type (i.e., the individual words) rather than texts. If set to "individually", then duplicate words are not aggregated, (i.e, the context of individual is preserved). (default = NULL).

`keep_token_embeddings`
(boolean) Whether to also keep token embeddings when using texts or word types aggregation.

`tokens_select` Option to select word embeddings linked to specific tokens such as [CLS] and [SEP] for the context embeddings.

`tokens_deselect`
Option to deselect embeddings linked to specific tokens such as [CLS] and [SEP] for the context embeddings.

`decontextualize`
(boolean) Provide word embeddings of single words as input to the model (these embeddings are, e.g., used for plotting; default is to use). If using this, then set single_context_embeddings to FALSE.

`model_max_length`
The maximum length (in number of tokens) for the inputs to the transformer model (default the value stored for the associated model).

`max_token_to_sentence`
(numeric) Maximum number of tokens in a string to handle before switching to embedding text sentence by sentence.

`tokenizer_parallelism`
(boolean) If TRUE this will turn on tokenizer parallelism. Default FALSE.

device	Name of device to use: 'cpu', 'gpu', 'gpu:k' or 'mps'/'mps:k' for MacOS, where k is a specific device number such as 'mps:1'.
hg_gated	Set to TRUE if the accessed model is gated.
hg_token	The token needed to access the gated model. Create a token from the ['Settings' page](https://huggingface.co/settings/tokens) of the Hugging Face website. An an environment variable HUGGINGFACE_TOKEN can be set to avoid the need to enter the token each time.
logging_level	Set the logging level. Default: "warning". Options (ordered from less logging to more logging): critical, error, warning, info, debug
...	settings from textEmbedRawLayers().

Value

A tibble with tokens, a column for layer identifier and word embeddings. Note that layer 0 is the input embedding to the transformer.

See Also

See textEmbedLayerAggregation, textEmbedRawLayers and textDimName.

Examples

```
# Automatically transforms the characters in the example dataset:
# Language_based_assessment_data_8 (included in text-package), to embeddings.
## Not run:
word_embeddings <- textEmbed(Language_based_assessment_data_8[1:2, 1:2],
  layers = 10:11,
  aggregation_from_layers_to_tokens = "concatenate",
  aggregation_from_tokens_to_texts = "mean",
  aggregation_from_tokens_to_word_types = "mean"
)

# Show information about how the embeddings were constructed.
comment(word_embeddings$texts$satisfactiontexts)
comment(word_embeddings$word_types)
comment(word_embeddings$tokens$satisfactiontexts)

# See how the word embeddings are structured.
word_embeddings

# Save the word embeddings to avoid having to embed the text again.
saveRDS(word_embeddings, "word_embeddings.rds")

# Retrieve the saved word embeddings.
word_embeddings <- readRDS("word_embeddings.rds")

## End(Not run)
```

`textEmbedLayerAggregation` *Select and aggregate layers of hidden states to form a word embedding.*

Description

Select and aggregate layers of hidden states to form a word embedding.

Usage

```
textEmbedLayerAggregation(
  word_embeddings_layers,
  layers = "all",
  aggregation_from_layers_to_tokens = "concatenate",
  aggregation_from_tokens_to_texts = "mean",
  return_tokens = FALSE,
  tokens_select = NULL,
  tokens_deselect = NULL
)
```

Arguments

`word_embeddings_layers`	Layers returned by the textEmbedRawLayers function.
`layers`	(character or numeric) The numbers of the layers to be aggregated (e.g., c(11:12) to aggregate the eleventh and twelfth). Note that layer 0 is the input embedding to the transformer, and should normally not be used. Selecting 'all' thus removes layer 0 (default = "all")
`aggregation_from_layers_to_tokens`	(character) Method to carry out the aggregation among the layers for each word/token, including "min", "max" and "mean" which takes the minimum, maximum or mean across each column; or "concatenate", which links together each layer of the word embedding to one long row (default = "concatenate").
`aggregation_from_tokens_to_texts`	(character) Method to carry out the aggregation among the word embeddings for the words/tokens, including "min", "max" and "mean" which takes the minimum, maximum or mean across each column; or "concatenate", which links together each layer of the word embedding to one long row (default = "mean").
`return_tokens`	(boolean) If TRUE, provide the tokens used in the specified transformer model (default = FALSE).
`tokens_select`	(character) Option to only select embeddings linked to specific tokens in the textEmbedLayerAggregation() phase such as "[CLS]" and "[SEP]" (default NULL).
`tokens_deselect`	(character) Option to deselect embeddings linked to specific tokens in the textEmbedLayerAggregation() phase such as "[CLS]" and "[SEP]" (default NULL).

Value

A tibble with word embeddings. Note that layer 0 is the input embedding to the transformer, which is normally not used.

See Also

See textEmbedRawLayers and textEmbed.

Examples

```
# Aggregate the hidden states from textEmbedRawLayers
# to create a word embedding representing the entire text.
# This is achieved by concatenating layer 11 and 12.
## Not run:
word_embedding <- textEmbedLayerAggregation(
  imf_embeddings_11_12$context_tokens,
  layers = 11:12,
  aggregation_from_layers_to_tokens = "concatenate",
  aggregation_from_tokens_to_texts = "mean"
)

# Examine word_embedding
word_embedding

## End(Not run)
```

---

textEmbedRawLayers *Extract layers of hidden states (word embeddings) for all character variables in a given dataframe.*

Description

Extract layers of hidden states (word embeddings) for all character variables in a given dataframe.

Usage

```
textEmbedRawLayers(
  texts,
  model = "bert-base-uncased",
  layers = -2,
  return_tokens = TRUE,
  word_type_embeddings = FALSE,
  decontextualize = FALSE,
  keep_token_embeddings = TRUE,
  device = "cpu",
  tokenizer_parallelism = FALSE,
  model_max_length = NULL,
  max_token_to_sentence = 4,
```

```
  hg_gated = FALSE,
  hg_token = Sys.getenv("HUGGINGFACE_TOKEN", unset = ""),
  logging_level = "error",
  sort = TRUE
)
```

Arguments

texts	A character variable or a tibble with at least one character variable.
model	(character) Character string specifying pre-trained language model (default = 'bert-base-uncased'). For full list of options see pretrained models at Hugging-Face. For example use "bert-base-multilingual-cased", "openai-gpt", "gpt2", "ctrl", "transfo-xl-wt103", "xlnet-base-cased", "xlm-mlm-enfr-1024", "distilbert-base-cased", "roberta-base", or "xlm-roberta-base". Only load models that you trust from HuggingFace; loading a malicious model can execute arbitrary code on your computer).
layers	(character or numeric) Specify the layers that should be extracted (default -2, which give the second to last layer). It is more efficient to only extract the layers that you need (e.g., 11). You can also extract several (e.g., 11:12), or all by setting this parameter to "all". Layer 0 is the decontextualized input layer (i.e., not comprising hidden states) and thus should normally not be used. These layers can then be aggregated in the textEmbedLayerAggregation function.
return_tokens	(boolean) If TRUE, provide the tokens used in the specified transformer model. (default = TRUE)
word_type_embeddings	(boolean) Wether to provide embeddings for each word/token type. (default = FALSE)
decontextualize	(boolean) Wether to dectonextualise embeddings (i.e., embedding one word at a time). (default = TRUE)
keep_token_embeddings	(boolean) Whether to keep token level embeddings in the output (when using word_types aggregation). (default= TRUE)
device	(character) Name of device to use: 'cpu', 'gpu', 'gpu:k' or 'mps'/'mps:k' for MacOS, where k is a specific device number. (default = "cpu")
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism. (default = FALSE).
model_max_length	The maximum length (in number of tokens) for the inputs to the transformer model (default the value stored for the associated model).
max_token_to_sentence	(numeric) Maximum number of tokens in a string to handle before switching to embedding text sentence by sentence. (default= 4)
hg_gated	Set to TRUE if the accessed model is gated.
hg_token	The token needed to access the gated model. Create a token from the ['Settings' page](https://huggingface.co/settings/tokens) of the Hugging Face website. An

	an environment variable HUGGINGFACE_TOKEN can be set to avoid the need to enter the token each time.
logging_level	(character) Set the logging level. (default ="error") Options (ordered from less logging to more logging): critical, error, warning, info, debug
sort	(boolean) If TRUE sort the output to tidy format. (default = TRUE)

Value

The textEmbedRawLayers() takes text as input, and returns the hidden states for each token of the text, including the [CLS] and the [SEP]. Note that layer 0 is the input embedding to the transformer, and should normally not be used.

See Also

See textEmbedLayerAggregation and textEmbed.

Examples

```
# Get hidden states of layer 11 and 12 for "I am fine".
## Not run:
imf_embeddings_11_12 <- textEmbedRawLayers(
  "I am fine",
  layers = 11:12
)

# Show hidden states of layer 11 and 12.
imf_embeddings_11_12

## End(Not run)
```

---

textEmbedReduce *Pre-trained dimension reduction (experimental)*

Description

Pre-trained dimension reduction (experimental)

Usage

```
textEmbedReduce(
  embeddings,
  n_dim = NULL,
  scalar = "fb20/scalar.csv",
  pca = "fb20/rpca_roberta_768_D_20.csv"
)
```

Arguments

embeddings	(list) Embedding(s) - including, tokens, texts and/or word_types.
n_dim	(numeric) Number of dimensions to reduce to.
scalar	(string or matrix) Name or URL to scalar for standardizing the embeddings. If a URL, the function first examines whether it has been downloaded before. The string should be to a csv file containing a matrix with the pca weights for matrix multiplication. For more information see reference below.
pca	(string or matrix) Name or URL to pca weights. If a URL, the function first examines whether it has been downlaoded before. The string should be to a csv file containing a matrix. For more information see reference below.

Details

To use this method please see and cite:
Ganesan, A. V., Matero, M., Ravula, A. R., Vu, H., & Schwartz, H. A. (2021, June). Empirical evaluation of pre-trained transformers for human-level nlp: The role of sample size and dimensionality. In Proceedings of the conference. Association for Computational Linguistics. North American Chapter. Meeting (Vol. 2021, p. 4515). NIH Public Access.

See Git-Hub Empirical-Evaluation

Value

Returns embeddings with reduced number of dimensions.

See Also

textEmbed

Examples

```
## Not run:
embeddings <- textEmbedReduce(word_embeddings_4$texts)

## End(Not run)
```

---

textEmbedStatic *Applies word embeddings from a given decontextualized static space (such as from Latent Semantic Analyses) to all character variables*

Description

Applies word embeddings from a given decontextualized static space (such as from Latent Semantic Analyses) to all character variables

Usage

```
textEmbedStatic(
  df,
  space,
  tk_df = "null",
  aggregation_from_tokens_to_texts = "mean",
  dim_name = FALSE,
  tolower = FALSE
)
```

Arguments

df	dataframe that at least contains one character column.
space	decontextualized/static space with a column called "words" and the semantic representations are in columns called Dim1, Dim2 (or V1, V2, ...) and so on (from textSpace, which is not included in the current text package).
tk_df	default "null"; option to use either the "tk" of "df" space (if using textSpace, which has not been implemented yet).
aggregation_from_tokens_to_texts	method to aggregate semantic representation when their are more than a single word. (default is "mean"; see also "min" and "max", "concatenate" and "normalize")
dim_name	Boolean, if TRUE append the variable name after all variable-names in the output. (This differentiates between word embedding dimension names; e.g., Dim1_text_variable_name)
tolower	(boolean) Lower case input.

Value

A list with tibbles for each character variable. Each tibble comprises a column with the text, followed by columns representing the semantic representations of the text. The tibbles are called the same as the original variable.

See Also

see textEmbed

---

textFineTuneDomain *Domain Adapted Pre-Training (EXPERIMENTAL - under development)*

Description

Domain Adapted Pre-Training (EXPERIMENTAL - under development)

Usage

```
textFineTuneDomain(
  text_data,
  model_name_or_path = "bert-base-uncased",
  output_dir = "./runs",
  validation_proportion = 0.1,
  evaluation_proportion = 0.1,
  config_name = NULL,
  tokenizer_name = NULL,
  max_seq_length = 128L,
  evaluation_strategy = "epoch",
  eval_accumulation_steps = NULL,
  num_train_epochs = 3,
  past_index = -1,
  set_seed = 2022,
  ...
)
```

Arguments

text_data	A dataframe, where the first column contain text data, and the second column the to-be-predicted variable (numeric or categorical).
model_name_or_path	(string) Path to foundation/pretrained model or model identifier from huggingface.co/models
output_dir	(string) Path to the output directory.
validation_proportion	(Numeric) Proportion of the text_data to be used for validation.
evaluation_proportion	(Numeric) Proportion of the text_data to be used for evaluation.
config_name	(String) Pretrained config name or path if not the same as model_name.
tokenizer_name	(String) Pretrained tokenizer name or path if not the same as model_name
max_seq_length	(Numeric) The maximum total input sequence length after tokenization. Sequences longer than this will be truncated, sequences shorter will be padded.
evaluation_strategy	(String or IntervalStrategy) — The evaluation strategy to adopt during training. Possible values are: "no": No evaluation is done during training. "steps": Evaluation is done (and logged) every eval_steps. "epoch": Evaluation is done at the end of each epoch.
eval_accumulation_steps	(Integer) Number of predictions steps to accumulate the output tensors for, before moving the results to the CPU. If left unset, the whole predictions are accumulated on GPU/TPU before being moved to the CPU (faster but requires more memory).
num_train_epochs	(Numeric) Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).

past_index	(Numeric, defaults to -1) Some models like TransformerXL or XLNet can make use of the past hidden states for their predictions. If this argument is set to a positive int, the Trainer will use the corresponding output (usually index 2) as the past state and feed it to the model at the next training step under the keyword argument mems.
set_seed	(Numeric) Set the seed
...	Parameters related to the fine tuning, which can be seen in the text-package file inst/python/arg2.json.

Details

Information about more parameters see inst/python/args2.json (https://github.com/OscarKjell/text/tree/master/inst/python/arg
Descriptions of settings can be found in inst/python/task_finetune.py under "class ModelArguments" and "class DataTrainingArguments" as well as online at https://huggingface.co/docs/transformers/main_classes/traine

Value

A folder containing the pretrained model and output data. The model can then be used, for example, by textEmbed() by providing the model parameter with a the path to the output folder.

See Also

see textEmbed, textEmbed

Examples

```
## Not run:
textFineTuneDomain(text_data)

## End(Not run)
```

---

textFineTuneTask *Task Adapted Pre-Training (EXPERIMENTAL - under development)*

Description

Task Adapted Pre-Training (EXPERIMENTAL - under development)

Usage

```
textFineTuneTask(
  text_outcome_data,
  model_name_or_path = "bert-base-uncased",
  output_dir = "./runs",
  validation_proportion = 0.1,
  evaluation_proportion = 0.1,
  is_regression = TRUE,
```

```
  config_name = NULL,
  tokenizer_name = NULL,
  max_seq_length = 128L,
  evaluation_strategy = "epoch",
  eval_accumulation_steps = NULL,
  num_train_epochs = 3,
  past_index = -1,
  set_seed = 2022,
  label_names = NULL,
  pytorch_mps_high_watermark_ratio = FALSE,
  tokenizer_parallelism = FALSE,
  ...
)
```

Arguments

`text_outcome_data`
: A dataframe, where the first column contain text data, and the second column the to-be-predicted variable (numeric or categorical).

`model_name_or_path`
: (string) Path to foundation/pretrained model or model identifier from huggingface.co/models

`output_dir`
: (string) Path to the output directory.

`validation_proportion`
: (Numeric) Proportion of the text_outcome_data to be used for validation.

`evaluation_proportion`
: (Numeric) Proportion of the text_outcome_data to be used for evaluation.

`is_regression`
: (Boolean) TRUE for regression tasks, FALSE for classification.

`config_name`
: (String) Pretrained config name or path if not the same as model_name.

`tokenizer_name`
: (String) Pretrained tokenizer name or path if not the same as model_name

`max_seq_length`
: (Numeric) The maximum total input sequence length after tokenization. Sequences longer than this will be truncated, sequences shorter will be padded.

`evaluation_strategy`
: (String or IntervalStrategy) — The evaluation strategy to adopt during training. Possible values are: "no": No evaluation is done during training. "steps": Evaluation is done (and logged) every eval_steps. "epoch": Evaluation is done at the end of each epoch.

`eval_accumulation_steps`
: (Integer) Number of predictions steps to accumulate the output tensors for, before moving the results to the CPU. If left unset, the whole predictions are accumulated on GPU/TPU before being moved to the CPU (faster but requires more memory).

`num_train_epochs`
: (Numeric) Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).

past_index	(Numeric, defaults to -1) Some models like TransformerXL or XLNet can make use of the past hidden states for their predictions. If this argument is set to a positive int, the Trainer will use the corresponding output (usually index 2) as the past state and feed it to the model at the next training step under the keyword argument mems.
set_seed	(Numeric) Set the seed
label_names	label name in case of classification; e.g., label_names = c("female", "male").
pytorch_mps_high_watermark_ratio	Set to TRUE to solve error RuntimeError: MPS backend out of memory.Use PYTORCH_MPS_HIGH_WATERMARK_RATIO=0.0 to disable upper limit for memory allocations (may cause system failure). Monitor System Resources: If you decide to adjust this setting, closely monitor your system's resource usage to ensure it does not become unstable.
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism. Default FALSE.
...	Parameters related to the fine tuning, which can be seen in the text-package file inst/python/arg2.json.

Details

Information about more parameters see inst/python/args2.json (https://github.com/OscarKjell/text/tree/master/inst/python/arg
Descriptions of settings can be found in inst/python/task_finetune.py under "class ModelArguments" and "class DataTrainingArguments" as well as online at https://huggingface.co/docs/transformers/main_classes/traine

Value

A folder containing the pretrained model and output data. The model can then be used, for example, by textEmbed() by providing the model parameter with a the path to the output folder.

See Also

see textEmbed, textEmbed

Examples

```
## Not run:
textFineTuneTask(text_outcome_data)

## End(Not run)
```

---

textGeneration *Predicts the words that will follow a specified text prompt. (experimental)*

Description

Predicts the words that will follow a specified text prompt. (experimental)

Usage

```
textGeneration(
  x,
  model = "gpt2",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "warning",
  return_incorrect_results = FALSE,
  return_tensors = FALSE,
  return_full_text = TRUE,
  clean_up_tokenization_spaces = FALSE,
  prefix = "",
  handle_long_generation = NULL,
  set_seed = 202208L
)
```

Arguments

x	(string) A variable or a tibble/dataframe with at least one character variable.
model	(string) Specification of a pre-trained language model that have been trained with an autoregressive language modeling objective, which includes the uni-directional models (e.g., gpt2).
device	(string) Device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism.
logging_level	(string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug
return_incorrect_results	(boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").
return_tensors	(boolean) Whether or not the output should include the prediction tensors (as token indices).
return_full_text	(boolean) If FALSE only the added text is returned, otherwise the full text is returned. (This setting is only meaningful if return_text is set to TRUE)
clean_up_tokenization_spaces	(boolean) Option to clean up the potential extra spaces in the returned text.
prefix	(string) Option to add a prefix to prompt.
handle_long_generation	By default, this function does not handle long generation (those that exceed the model maximum length).

set_seed	(Integer) Set seed. (more info :https://github.com/huggingface/transformers/issues/14033#issuecomment-948385227). This setting provides some ways to work around the problem: None: default way, where no particular strategy is applied. "hole": Truncates left of input, and leaves a gap that is wide enough to let generation happen. (this might truncate a lot of the prompt and not suitable when generation exceed the model capacity)

Value

A tibble with generated text.

See Also

see textClassify, textNER, textSum, textQA, textTranslate

Examples

```
# generated_text <- textGeneration("The meaning of life is")
# generated_text
```

---

textModelLayers *Get the number of layers in a given model.*

Description

Get the number of layers in a given model.

Usage

```
textModelLayers(
  target_model,
  hg_gated = FALSE,
  hg_token = Sys.getenv("HUGGINGFACE_TOKEN", unset = "")
)
```

Arguments

target_model	(string) The name of the model to know the number of layers of.
hg_gated	Set to TRUE if the accessed model is gated.
hg_token	The token needed to access the gated model. Create a token from the ['Settings' page](https://huggingface.co/settings/tokens) of the Hugging Face website. An an environment variable HUGGINGFACE_TOKEN can be set to avoid the need to enter the token each time.

Value

Number of layers.

See Also

see textModels

Examples

```
## Not run:
textModelLayers(target_model = "bert-base-uncased")

## End(Not run)
```

---

textModels *Check downloaded, available models.*

Description

Check downloaded, available models.

Usage

```
textModels()
```

Value

List of names of models and tokenizers

See Also

see textModelsRemove

Examples

```
## Not run:
textModels()

## End(Not run)
```

textModelsRemove *Delete a specified model and model associated files.*

Description

Delete a specified model and model associated files.

Usage

```
textModelsRemove(target_model)
```

Arguments

target_model	(string) The name of the model to be deleted.

Value

Confirmation whether the model has been deleted.

See Also

see textModels

Examples

```
## Not run:
textModelsRemove("name-of-model-to-delete")

## End(Not run)
```

textNER *Named Entity Recognition. (experimental)*

Description

Named Entity Recognition. (experimental)

Usage

```
textNER(
  x,
  model = "dslim/bert-base-NER",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "error",
  return_incorrect_results = FALSE,
  set_seed = 202208L
)
```

Arguments

x	(string) A variable or a tibble/dataframe with at least one character variable.
model	(string) Specification of a pre-trained language model for token classification that have been fine-tuned on a NER task (e.g., see "dslim/bert-base-NER"). Use for predicting the classes of tokens in a sequence: person, organisation, location or miscellaneous).
device	(string) Device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism.
logging_level	(string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug
return_incorrect_results	(boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").
set_seed	(Integer) Set seed.

Value

A list with tibble(s) with NER classifications for each column.

See Also

see textClassify, textGeneration, textNER, textSum, textQA, textTranslate

Examples

```
# ner_example <- textNER("Arnes plays football with Daniel")
# ner_example
```

---

textPCA *Compute 2 PCA dimensions of the word embeddings for individual words.*

Description

Compute 2 PCA dimensions of the word embeddings for individual words.

Usage

```
textPCA(
  words,
  word_types_embeddings = word_types_embeddings_df,
  to_lower_case = TRUE,
  seed = 1010
)
```

Arguments

words	Word or text variable to be plotted.
word_types_embeddings	Word embeddings from textEmbed for individual words (i.e., decontextualized embeddings).
to_lower_case	Lower case words
seed	Set different seed.

Value

A dataframe with words, their frquency and two PCA dimensions from the word_embeddings for the individual words that is used for the plotting in the textPCAPlot function.

See Also

see textPCAPlot

Examples

```
## Not run:
# Data
df_for_plotting2d <- textPCA(
  words = Language_based_assessment_data_8$harmonywords,
  word_types_embeddings = word_embeddings_4$word_types
)
df_for_plotting2d

## End(Not run)
```

---

textPCAPlot *Plot words according to 2-D plot from 2 PCA components.*

Description

Plot words according to 2-D plot from 2 PCA components.

Usage

```
textPCAPlot(
  word_data,
  min_freq_words_test = 1,
  plot_n_word_extreme = 5,
  plot_n_word_frequency = 5,
  plot_n_words_middle = 5,
  titles_color = "#61605e",
  title_top = "Principal Component (PC) Plot",
  x_axes_label = "PC1",
  y_axes_label = "PC2",
  scale_x_axes_lim = NULL,
  scale_y_axes_lim = NULL,
  word_font = NULL,
 bivariate_color_codes = c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA",
    "#40DD52", "#FF0000", "#EA7467", "#85DB8E"),
  word_size_range = c(3, 8),
  position_jitter_hight = 0,
  position_jitter_width = 0.03,
  point_size = 0.5,
  arrow_transparency = 0.1,
  points_without_words_size = 0.2,
  points_without_words_alpha = 0.2,
  legend_title = "PC",
  legend_x_axes_label = "PC1",
  legend_y_axes_label = "PC2",
  legend_x_position = 0.02,
  legend_y_position = 0.02,
  legend_h_size = 0.2,
  legend_w_size = 0.2,
  legend_title_size = 7,
  legend_number_size = 2,
  seed = 1002
)
```

Arguments

word_data	Dataframe from textPCA
min_freq_words_test	Select words to significance test that have occurred at least min_freq_words_test (default = 1).
plot_n_word_extreme	Number of words that are extreme on Supervised Dimension Projection per dimension. (i.e., even if not significant; per dimensions, where duplicates are removed).
plot_n_word_frequency	Number of words based on being most frequent. (i.e., even if not significant).

plot_n_words_middle
: Number of words plotted that are in the middle in Supervised Dimension Projection score (i.e., even if not significant; per dimensions, where duplicates are removed).

titles_color
: Color for all the titles (default: "#61605e")

title_top
: Title (default " ")

x_axes_label
: Label on the x-axes.

y_axes_label
: Label on the y-axes.

scale_x_axes_lim
: Manually set the length of the x-axes (default = NULL, which uses ggplot2::scale_x_continuous(limits = scale_x_axes_lim); change e.g., by trying c(-5, 5)).

scale_y_axes_lim
: Manually set the length of the y-axes (default = NULL; which uses ggplot2::scale_y_continuous(limits = scale_y_axes_lim); change e.g., by trying c(-5, 5)).

word_font
: Font type (default: NULL).

bivariate_color_codes
: The different colors of the words (default: c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA", "#40DD52", "#FF0000", "#EA7467", "#85DB8E")).

word_size_range
: Vector with minimum and maximum font size (default: c(3, 8)).

position_jitter_hight
: Jitter height (default: .0).

position_jitter_width
: Jitter width (default: .03).

point_size
: Size of the points indicating the words' position (default: 0.5).

arrow_transparency
: Transparency of the lines between each word and point (default: 0.1).

points_without_words_size
: Size of the points not linked with a words (default is to not show it, i.e., 0).

points_without_words_alpha
: Transparency of the points not linked with a words (default is to not show it, i.e., 0).

legend_title
: Title on the color legend (default: "(PCA)".

legend_x_axes_label
: Label on the color legend (default: "(x)".

legend_y_axes_label
: Label on the color legend (default: "(y)".

legend_x_position
: Position on the x coordinates of the color legend (default: 0.02).

legend_y_position
: Position on the y coordinates of the color legend (default: 0.05).

legend_h_size
: Height of the color legend (default 0.15).

legend_w_size
: Width of the color legend (default 0.15).

legend_title_size
: Font size (default: 7).

legend_number_size
: Font size of the values in the legend (default: 2).

seed
: Set different seed.

Value

A 1- or 2-dimensional word plot, as well as tibble with processed data used to plot..

See Also

see textPCA

Examples

```
# The test-data included in the package is called: DP_projections_HILS_SWLS_100

# Supervised Dimension Projection Plot
principle_component_plot_projection <- textPCAPlot(PC_projections_satisfactionwords_40)
principle_component_plot_projection

names(DP_projections_HILS_SWLS_100)
```

---

textPlot *Plot words from textProjection() or textWordPrediction().*

Description

Plot words from textProjection() or textWordPrediction().

Usage

```
textPlot(
  word_data,
  k_n_words_to_test = FALSE,
  min_freq_words_test = 1,
  min_freq_words_plot = 1,
  plot_n_words_square = 3,
  plot_n_words_p = 5,
  plot_n_word_extreme = 5,
  plot_n_word_frequency = 5,
  plot_n_words_middle = 5,
  titles_color = "#61605e",
  y_axes = FALSE,
  p_alpha = 0.05,
  overlapping = TRUE,
  p_adjust_method = "none",
```

```
  title_top = "Supervised Dimension Projection",
  x_axes_label = "Supervised Dimension Projection (SDP)",
  y_axes_label = "Supervised Dimension Projection (SDP)",
  scale_x_axes_lim = NULL,
  scale_y_axes_lim = NULL,
  word_font = NULL,
 bivariate_color_codes = c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA",
    "#40DD52", "#FF0000", "#EA7467", "#85DB8E"),
  word_size_range = c(3, 8),
  position_jitter_hight = 0,
  position_jitter_width = 0.03,
  point_size = 0.5,
  arrow_transparency = 0.1,
  points_without_words_size = 0.2,
  points_without_words_alpha = 0.2,
  legend_title = "SDP",
  legend_x_axes_label = "x",
  legend_y_axes_label = "y",
  legend_x_position = 0.02,
  legend_y_position = 0.02,
  legend_h_size = 0.2,
  legend_w_size = 0.2,
  legend_title_size = 7,
  legend_number_size = 2,
  group_embeddings1 = FALSE,
  group_embeddings2 = FALSE,
  projection_embedding = FALSE,
  aggregated_point_size = 0.8,
  aggregated_shape = 8,
  aggregated_color_G1 = "black",
  aggregated_color_G2 = "black",
  projection_color = "blue",
  seed = 1005,
  explore_words = NULL,
  explore_words_color = "#ad42f5",
  explore_words_point = "ALL_1",
  explore_words_aggregation = "mean",
  remove_words = NULL,
  n_contrast_group_color = NULL,
  n_contrast_group_remove = FALSE,
  space = NULL,
  scaling = FALSE
)
```

Arguments

word_data	Dataframe from textProjection.
k_n_words_to_test	Select the k most frequent words to significance test (k = sqrt(100*N); N =

number of participant responses) (default = TRUE).

`min_freq_words_test`
: Select words to significance test that have occurred at least min_freq_words_test (default = 1).

`min_freq_words_plot`
: Select words to plot that has occurred at least min_freq_words_plot times (default = 1).

`plot_n_words_square`
: Select number of significant words in each square of the figure to plot. The significant words, in each square is selected according to most frequent words (default = 3).

`plot_n_words_p`
: Number of significant words to plot on each (positive and negative) side of the x-axes and y-axes, (where duplicates are removed); selects first according to lowest p-value and then according to frequency (default = 5). Hence, on a two dimensional plot it is possible that plot_n_words_p = 1 yield 4 words.

`plot_n_word_extreme`
: Number of words that are extreme on Supervised Dimension Projection per dimension (default = 5). (i.e., even if not significant; per dimensions, where duplicates are removed).

`plot_n_word_frequency`
: Number of words based on being most frequent (default = 5). (i.e., even if not significant).

`plot_n_words_middle`
: Number of words plotted that are in the middle in Supervised Dimension Projection score (default = 5). (i.e., even if not significant; per dimensions, where duplicates are removed).

`titles_color`
: Color for all the titles (default: "#61605e").

`y_axes`
: (boolean) If TRUE, also plotting on the y-axes (default = FALSE, i.e, a 1-dimensional plot is generated). Also plotting on y-axes produces a two dimension 2-dimensional plot, but the textProjection function has to have had a variable on the y-axes.

`p_alpha`
: Alpha (default = .05).

`overlapping`
: (boolean) Allow overlapping (TRUE) or disallow (FALSE) (default = TRUE).

`p_adjust_method`
: (character) Method to adjust/correct p-values for multiple comparisons (default = "none"; see also "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr").

`title_top`
: Title (default " ").

`x_axes_label`
: (character) Label on the x-axes (default = "Supervised Dimension Projection (SDP)").

`y_axes_label`
: (character) Label on the y-axes (default = "Supervised Dimension Projection (SDP)").

`scale_x_axes_lim`
: Manually set the length of the x-axes (default = NULL, which uses ggplot2::scale_x_continuous(limits = scale_x_axes_lim); change e.g., by trying c(-5, 5)).

scale_y_axes_lim
: Manually set the length of the y-axes (default = NULL; which uses ggplot2::scale_y_continuous(limits = scale_y_axes_lim); change e.g., by trying c(-5, 5)).

word_font
: Font type (default = NULL).

bivariate_color_codes
: (HTML color codes. Type = character) The different colors of the words. Note that, at the moment, two squares should not have the exact same colour-code because the numbers within the squares of the legend will then be aggregated (and show the same, incorrect value). (default: c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA", "#40DD52", "#FF0000", "#EA7467", "#85DB8E")).

word_size_range
: Vector with minimum and maximum font size (default: c(3, 8)).

position_jitter_hight
: Jitter height (default: .0).

position_jitter_width
: Jitter width (default: .03).

point_size
: Size of the points indicating the words' position (default: 0.5).

arrow_transparency
: Transparency of the lines between each word and point (default: 0.1).

points_without_words_size
: Size of the points not linked with a words (default is to not show it, i.e., 0).

points_without_words_alpha
: Transparency of the points not linked with a words (default is to not show it, i.e., 0).

legend_title
: Title on the color legend (default: "SDP").

legend_x_axes_label
: Label on the color legend (default: "x").

legend_y_axes_label
: Label on the color legend (default: "y").

legend_x_position
: Position on the x coordinates of the color legend (default: 0.02).

legend_y_position
: Position on the y coordinates of the color legend (default: 0.05).

legend_h_size
: Height of the color legend (default 0.15).

legend_w_size
: Width of the color legend (default 0.15).

legend_title_size
: Font size (default: 7).

legend_number_size
: Font size of the values in the legend (default: 2).

group_embeddings1
: (boolean) Shows a point representing the aggregated word embedding for group 1 (default = FALSE).

group_embeddings2
: (boolean) Shows a point representing the aggregated word embedding for group 2 (default = FALSE).

`projection_embedding`
: (boolean) Shows a point representing the aggregated direction embedding (default = FALSE).

`aggregated_point_size`
: Size of the points representing the group_embeddings1, group_embeddings2 and projection_embedding (default = 0.8).

`aggregated_shape`
: Shape type of the points representing the group_embeddings1, group_embeddings2 and projection_embedding (default = 8).

`aggregated_color_G1`
: Color (default = "black").

`aggregated_color_G2`
: Color (default = "black").

`projection_color`
: Color (default = "blue").

`seed`
: (numeric) Set different seed (default = 1005)..

`explore_words`
: Explore where specific words are positioned in the embedding space. For example, c("happy content", "sad down") (default = NULL).

`explore_words_color`
: Specify the color(s) of the words being explored. For example c("#ad42f5", "green") (default = "#ad42f5").

`explore_words_point`
: Specify the names of the point for the aggregated word embeddings of all the explored words (default = "ALL_1").

`explore_words_aggregation`
: Specify how to aggregate the word embeddings of the explored words (default = "mean").

`remove_words`
: Manually remove words from the plot (which is done just before the words are plotted so that the remove_words are part of previous counts/analyses) (default = NULL).

`n_contrast_group_color`
: Set color to words that have higher frequency (N) on the other opposite side of its dot product projection (default = NULL).

`n_contrast_group_remove`
: Remove words that have higher frequency (N) on the other opposite side of its dot product projection (default = FALSE).

`space`
: Provide a semantic space if using static embeddings and wanting to explore words (default = NULL).

`scaling`
: Scaling word embeddings before aggregation (default = FALSE).

Value

A 1- or 2-dimensional word plot, as well as tibble with processed data used to plot.

See Also

See `textProjection`.

Examples

```
# The test-data included in the package is called: DP_projections_HILS_SWLS_100

# Supervised Dimension Projection Plot
plot_projection <- textPlot(
  word_data = DP_projections_HILS_SWLS_100,
  k_n_words_to_test = FALSE,
  min_freq_words_test = 1,
  plot_n_words_square = 3,
  plot_n_words_p = 3,
  plot_n_word_extreme = 1,
  plot_n_word_frequency = 1,
  plot_n_words_middle = 1,
  y_axes = FALSE,
  p_alpha = 0.05,
  title_top = "Supervised Dimension Projection (SDP)",
  x_axes_label = "Low vs. High HILS score",
  y_axes_label = "Low vs. High SWLS score",
  p_adjust_method = "bonferroni",
  scale_y_axes_lim = NULL
)
plot_projection

names(DP_projections_HILS_SWLS_100)
```

---

textPredict

Trained models created by e.g., textTrain() or stored on e.g., github can be used to predict new scores or classes from embeddings or text using textPredict.

Description

Trained models created by e.g., textTrain() or stored on e.g., github can be used to predict new scores or classes from embeddings or text using textPredict.

Usage

```
textPredict(
  model_info = NULL,
  word_embeddings = NULL,
  texts = NULL,
  x_append = NULL,
  type = NULL,
  dim_names = TRUE,
  save_model = TRUE,
  threshold = NULL,
  show_texts = FALSE,
  device = "cpu",
```

```
  participant_id = NULL,
  save_embeddings = TRUE,
  save_dir = "wd",
  save_name = "textPredict",
  story_id = NULL,
  dataset_to_merge_predictions = NULL,
  previous_sentence = FALSE,
  ...
)
```

Arguments

model_info	(character or r-object) model_info has three options. 1: R model object (e.g, saved output from textTrain). 2:link to github-model (e.g, "https://github.com/CarlViggo/pretrained_swls_ 3: Path to a model stored locally (e.g, "path/to/your/model"). Information about accessble models can be found at: r-text.org.
word_embeddings	(tibble) Embeddings from e.g., textEmbed(). If you're using a pre-trained model, then texts and embeddings cannot be submitted simultaneously (default = NULL).
texts	(character) Text to predict. If this argument is specified, then arguments "word_embeddings" and "premade embeddings" cannot be defined (default = NULL).
x_append	(tibble) Variables to be appended after the word embeddings (x).
type	(character) Defines what output to give after logistic regression prediction. Either probabilities, classifications or both are returned (default = "class". For probabilities use "prob". For both use "class_prob").
dim_names	(boolean) Account for specific dimension names from textEmbed() (rather than generic names including Dim1, Dim2 etc.). If FALSE the models need to have been trained on word embeddings created with dim_names FALSE, so that embeddings were only called Dim1, Dim2 etc.
save_model	(boolean) The model will by default be saved in your work-directory (default = TRUE). If the model already exists in your work-directory, it will automatically be loaded from there.
threshold	(numeric) Determine threshold if you are using a logistic model (default = 0.5).
show_texts	(boolean) Show texts together with predictions (default = FALSE).
device	Name of device to use: 'cpu', 'gpu', 'gpu:k' or 'mps'/'mps:k' for MacOS, where k is a specific device number such as 'mps:1'.
participant_id	(list) Vector of participant-ids. Specify this for getting person level scores (i.e., summed sentence probabilities to the person level corrected for word count). (default = NULL)
save_embeddings	(boolean) If set to TRUE, embeddings will be saved with a unique identifier, and will be automatically opened next time textPredict is run with the same text. (default = TRUE)
save_dir	(character) Directory to save embeddings. (default = "wd" (i.e, work-directory))

save_name (character) Name of the saved embeddings (will be combined with a unique identifier). (default = ""). Obs: If no save_name is provided, and model_info is a character, then save_name will be set to model_info.

story_id (vector) Vector of story-ids. Specify this to get story level scores (i.e., summed sentence probabilities corrected for word count). When there is both story_id and participant_id indicated, the function returns a list including both story level and person level prediction corrected for word count. (default = NULL)

dataset_to_merge_predictions
(R-object, tibble) Insert your data here to integrate predictions to your dataset, (default = NULL).

previous_sentence
If set to TRUE, word-embeddings will be averaged over the current and previous sentence per story-id. For this, both participant-id and story-id must be specified.

... Setting from stats::predict can be called.

Value

Predictions from word-embedding or text input.

See Also

See textTrain, textTrainLists and textTrainRandomForest.

Examples

```
## Not run:

# Text data from Language_based_assessment_data_8
text_to_predict <- "I am not in harmony in my life as much as I would like to be."

# Example 1: (predict using pre-made embeddings and an R model-object)
prediction1 <- textPredict(
  model_info = trained_model,
  word_embeddings_4$texts$satisfactiontexts
)

# Example 2: (predict using a pretrained github model)
prediction3 <- textPredict(
  texts = text_to_predict,
 model_info = "https://github.com/CarlViggo/pretrained-models/raw/main/trained_hils_model.RDS"
)

# Example 3: (predict using a pretrained logistic github model and return
# probabilities and classifications)
prediction4 <- textPredict(
  texts = text_to_predict,
  model_info = "https://github.com/CarlViggo/pretrained-models/raw/main/
  trained_github_model_logistic.RDS",
  type = "class_prob",
  threshold = 0.7
```

```
)

##### Automatic implicit motive coding section ######

# Create example dataset
implicit_motive_data <- dplyr::mutate(.data = Language_based_assessment_data_8,
participant_id = dplyr::row_number())

# Code implicit motives.
implicit_motives <- textPredict(
  texts = implicit_motive_data$satisfactiontexts,
  model_info = "implicit_power_roberta_large_L23_v1",
  participant_id = implicit_motive_data$participant_id,
  dataset_to_merge_predictions = implicit_motive_data
)

# Examine results
implicit_motives$sentence_predictions
implicit_motives$person_predictions

## End(Not run)

## Not run:
# Examine the correlation between the predicted values and
# the Satisfaction with life scale score (pre-included in text).

psych::corr.test(
  predictions1$word_embeddings__ypred,
  Language_based_assessment_data_8$swlstotal
)

## End(Not run)
```

`textPredictAll` *Predict from several models, selecting the correct input*

Description

Predict from several models, selecting the correct input

Usage

```
textPredictAll(models, word_embeddings, x_append = NULL, ...)
```

Arguments

`models`	Object containing several models.
`word_embeddings`	List of word embeddings (if using word embeddings from more than one text-variable use dim_names = TRUE throughout the pipeline).

x_append	A tibble/dataframe with additional variables used in the training of the models (optional).
...	Settings from textPredict.

Value

A tibble with predictions.

See Also

see textPredict and textTrain

Examples

```
# x <- Language_based_assessment_data_8[1:2, 1:2]
# word_embeddings_with_layers <- textEmbedLayersOutput(x, layers = 11:12)
```

---

textPredictTest *Significance testing correlations If only y1 is provided a t-test is computed, between the absolute error from yhat1-y1 and yhat2-y1.*

Description

If y2 is provided a bootstrapped procedure is used to compare the correlations between y1 and yhat1 versus y2 and yhat2. This is achieved by creating two distributions of correlations using bootstrapping; and then finally compute the distributions overlap.

Usage

```
textPredictTest(
  y1,
  y2,
  yhat1,
  yhat2,
  method = "t-test",
  statistic = "correlation",
  paired = TRUE,
  event_level = "first",
  bootstraps_times = 1000,
  seed = 6134,
  ...
)
```

Arguments

y1	The observed scores (i.e., what was used to predict when training a model).
y2	The second observed scores (default = NULL; i.e., for when comparing models that are predicting different outcomes. In this case a bootstrap procedure is used to create two distributions of correlations that are compared (see description above).
yhat1	The predicted scores from model 1.
yhat2	The predicted scores from model 2 that will be compared with model 1.
method	Set "t-test" if comparing predictions from models that predict the SAME outcome. Set "bootstrap" if comparing predictions from models that predict DIFFERENT outcomes or comparison from logistic regression computing AUC distributions.
statistic	Character ("correlation", "auc") describing statistic to be compared in bootstrapping.
paired	Paired test or not in stats::t.test (default TRUE).
event_level	Character "first" or "second" for computing the auc in the bootstrap.
bootstraps_times	Number of bootstraps (when providing y2).
seed	Set seed.
...	Settings from stats::t.test or overlapping::overlap (e.g., plot = TRUE).

Value

Comparison of correlations either a t-test or the overlap of a bootstrapped procedure (see $OV).

See Also

see textTrain textPredict

Examples

```
# Example random data
y1 <- runif(10)
yhat1 <- runif(10)
y2 <- runif(10)
yhat2 <- runif(10)

boot_test <- textPredictTest(y1, y2, yhat1, yhat2)
```

`textProjection` *Compute Supervised Dimension Projection and related variables for plotting words.*

Description

Compute Supervised Dimension Projection and related variables for plotting words.

Usage

```
textProjection(
  words,
  word_embeddings,
  word_types_embeddings,
  x,
  y = NULL,
  pca = NULL,
  aggregation = "mean",
  split = "quartile",
  word_weight_power = 1,
  min_freq_words_test = 0,
  mean_centering = FALSE,
  mean_centering2 = FALSE,
  Npermutations = 10000,
  n_per_split = 50000,
  seed = 1003
)
```

Arguments

`words`	(character) Word or text variable to be plotted.
`word_embeddings`	Word embeddings from textEmbed for the words to be plotted (i.e., the aggregated word embeddings for the "words" parameter).
`word_types_embeddings`	Word embeddings from textEmbed for individual words (i.e., decontextualized embeddings).
`x`	Numeric variable that the words should be plotted according to on the x-axes.
`y`	Numeric variable that the words should be plotted according to on the y-axes (default = NULL, i.e., a 1-dimensional plot is created).
`pca`	Number of PCA dimensions applied to the word embeddings in the beginning of the function (default = NULL). A number below 1 takes out % of variance; An integer specify number of components to extract. (default is NULL as this setting has not yet been evaluated).
`aggregation`	(character) Method to aggregate the word embeddings (default = "mean"; see also "min", "max", and "[CLS]").

split (character) Method to split the axes (default = "quartile" involving selecting lower and upper quartile; see also "mean"). However, if the variable is only containing two different values (i.e., being dichotomous) mean split is used.

word_weight_power
Compute the power of the frequency of the words and multiply the word embeddings with this in the computation of aggregated word embeddings for group low (1) and group high (2). This increases the weight of more frequent words.

min_freq_words_test
(numeric) Option to select words that have occurred a specified number of times (default = 0); when creating the Supervised Dimension Projection line (i.e., single words receive Supervised Dimension Projection and p-value).

mean_centering (boolean) Separately mean centering the Group 1 split aggregation embedding, and the Group 2 split aggregation embedding

mean_centering2
(boolean) Separately mean centering the G1 and G2 split aggregation embeddings

Npermutations (numeric) Number of permutations in the creation of the null distribution (default = 10000).

n_per_split (numeric) Setting to split Npermutations to avoid reaching computer memory limits; set it lower than Npermutations <- and the higher it is set the faster the computation completes, but too high may lead to abortion (default = 50000).

seed (numeric) Set different seed (default = 1003).

Value

A dataframe with variables (e.g., including Supervised Dimension Projection, frequencies, p-values) for the individual words that is used for the plotting in the textProjectionPlot function.

See Also

See textProjectionPlot.

Examples

```
# Pre-processing data for plotting.
## Not run:
df_for_plotting <- textProjection(
  words = Language_based_assessment_data_8$harmonywords,
  word_embeddings = word_embeddings_4$texts$harmonywords,
  word_types_embeddings = word_embeddings_4$word_types,
  x = Language_based_assessment_data_8$hilstotal,
  split = "mean",
  Npermutations = 10,
  n_per_split = 1
)
# Run df_for_plotting to examine result.
df_for_plotting

## End(Not run)
```

`textProjectionPlot` *Plot words according to Supervised Dimension Projection.*

Description

Plot words according to Supervised Dimension Projection.

Usage

```
textProjectionPlot(
  word_data,
  k_n_words_to_test = FALSE,
  min_freq_words_test = 1,
  min_freq_words_plot = 1,
  plot_n_words_square = 3,
  plot_n_words_p = 5,
  plot_n_word_extreme = 5,
  plot_n_word_frequency = 5,
  plot_n_words_middle = 5,
  titles_color = "#61605e",
  y_axes = FALSE,
  p_alpha = 0.05,
  overlapping = TRUE,
  p_adjust_method = "none",
  title_top = "Supervised Dimension Projection",
  x_axes_label = "Supervised Dimension Projection (SDP)",
  y_axes_label = "Supervised Dimension Projection (SDP)",
  scale_x_axes_lim = NULL,
  scale_y_axes_lim = NULL,
  word_font = NULL,
 bivariate_color_codes = c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA",
    "#40DD52", "#FF0000", "#EA7467", "#85DB8E"),
 word_size_range = c(3, 8),
 position_jitter_hight = 0,
 position_jitter_width = 0.03,
 point_size = 0.5,
 arrow_transparency = 0.1,
 points_without_words_size = 0.2,
 points_without_words_alpha = 0.2,
 legend_title = "SDP",
 legend_x_axes_label = "x",
 legend_y_axes_label = "y",
 legend_x_position = 0.02,
 legend_y_position = 0.02,
 legend_h_size = 0.2,
 legend_w_size = 0.2,
 legend_title_size = 7,
```

```
  legend_number_size = 2,
  group_embeddings1 = FALSE,
  group_embeddings2 = FALSE,
  projection_embedding = FALSE,
  aggregated_point_size = 0.8,
  aggregated_shape = 8,
  aggregated_color_G1 = "black",
  aggregated_color_G2 = "black",
  projection_color = "blue",
  seed = 1005,
  explore_words = NULL,
  explore_words_color = "#ad42f5",
  explore_words_point = "ALL_1",
  explore_words_aggregation = "mean",
  remove_words = NULL,
  n_contrast_group_color = NULL,
  n_contrast_group_remove = FALSE,
  space = NULL,
  scaling = FALSE
)
```

Arguments

word_data	Dataframe from textProjection
k_n_words_to_test	Select the k most frequent words to significance test (k = sqrt(100*N); N = number of participant responses). Default = TRUE.
min_freq_words_test	Select words to significance test that have occurred at least min_freq_words_test (default = 1).
min_freq_words_plot	Select words to plot that has occurred at least min_freq_words_plot times.
plot_n_words_square	Select number of significant words in each square of the figure to plot. The significant words, in each square is selected according to most frequent words.
plot_n_words_p	Number of significant words to plot on each(positive and negative) side of the x-axes and y-axes, (where duplicates are removed); selects first according to lowest p-value and then according to frequency. Hence, on a two dimensional plot it is possible that plot_n_words_p = 1 yield 4 words.
plot_n_word_extreme	Number of words that are extreme on Supervised Dimension Projection per dimension. (i.e., even if not significant; per dimensions, where duplicates are removed).
plot_n_word_frequency	Number of words based on being most frequent. (i.e., even if not significant).
plot_n_words_middle	Number of words plotted that are in the middle in Supervised Dimension Projection score (i.e., even if not significant; per dimensions, where duplicates are

removed).

titles_color — Color for all the titles (default: "#61605e")

y_axes — If TRUE, also plotting on the y-axes (default is FALSE). Also plotting on y-axes produces a two dimension 2-dimensional plot, but the textProjection function has to have had a variable on the y-axes.

p_alpha — Alpha (default = .05).

overlapping — (boolean) Allow overlapping (TRUE) or disallow (FALSE) (default = TRUE).

p_adjust_method — Method to adjust/correct p-values for multiple comparisons (default = "holm"; see also "none", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr").

title_top — Title (default " ")

x_axes_label — Label on the x-axes.

y_axes_label — Label on the y-axes.

scale_x_axes_lim — Manually set the length of the x-axes (default = NULL, which uses ggplot2::scale_x_continuous(limits = scale_x_axes_lim); change e.g., by trying c(-5, 5)).

scale_y_axes_lim — Manually set the length of the y-axes (default = NULL; which uses ggplot2::scale_y_continuous(limits = scale_y_axes_lim); change e.g., by trying c(-5, 5)).

word_font — Font type (default: NULL).

bivariate_color_codes — The different colors of the words. Note that, at the moment, two squares should not have the exact same colour-code because the numbers within the squares of the legend will then be aggregated (and show the same, incorrect value). (default: c("#398CF9", "#60A1F7", "#5dc688", "#e07f6a", "#EAEAEA", "#40DD52", "#FF0000", "#EA7467", "#85DB8E")).

word_size_range — Vector with minimum and maximum font size (default: c(3, 8)).

position_jitter_hight — Jitter height (default: .0).

position_jitter_width — Jitter width (default: .03).

point_size — Size of the points indicating the words' position (default: 0.5).

arrow_transparency — Transparency of the lines between each word and point (default: 0.1).

points_without_words_size — Size of the points not linked with a words (default is to not show it, i.e., 0).

points_without_words_alpha — Transparency of the points not linked with a words (default is to not show it, i.e., 0).

legend_title — Title on the color legend (default: "(SDP)".

legend_x_axes_label — Label on the color legend (default: "(x)".

legend_y_axes_label
: Label on the color legend (default: "(y)".

legend_x_position
: Position on the x coordinates of the color legend (default: 0.02).

legend_y_position
: Position on the y coordinates of the color legend (default: 0.05).

legend_h_size
: Height of the color legend (default 0.15).

legend_w_size
: Width of the color legend (default 0.15).

legend_title_size
: Font size (default: 7).

legend_number_size
: Font size of the values in the legend (default: 2).

group_embeddings1
: Shows a point representing the aggregated word embedding for group 1 (default = FALSE).

group_embeddings2
: Shows a point representing the aggregated word embedding for group 2 (default = FALSE).

projection_embedding
: Shows a point representing the aggregated direction embedding (default = FALSE).

aggregated_point_size
: Size of the points representing the group_embeddings1, group_embeddings2 and projection_embedding

aggregated_shape
: Shape type of the points representing the group_embeddings1, group_embeddings2 and projection_embeddingd

aggregated_color_G1
: Color

aggregated_color_G2
: Color

projection_color
: Color

seed
: Set different seed.

explore_words
: Explore where specific words are positioned in the embedding space. For example, c("happy content", "sad down").

explore_words_color
: Specify the color(s) of the words being explored. For example c("#ad42f5", "green")

explore_words_point
: Specify the names of the point for the aggregated word embeddings of all the explored words.

explore_words_aggregation
: Specify how to aggregate the word embeddings of the explored words.

remove_words
: manually remove words from the plot (which is done just before the words are plotted so that the remove_words are part of previous counts/analyses).

n_contrast_group_color
: Set color to words that have higher frequency (N) on the other opposite side of its dot product projection (default = NULL).

n_contrast_group_remove
: Remove words that have higher frequency (N) on the other opposite side of its dot product projection (default = FALSE).

space
: Provide a semantic space if using static embeddings and wanting to explore words.

scaling
: Scaling word embeddings before aggregation.

Value

A 1- or 2-dimensional word plot, as well as tibble with processed data used to plot.

See Also

See textProjection.

Examples

```
# The test-data included in the package is called: DP_projections_HILS_SWLS_100.
# The dataframe created by textProjection can also be used as input-data.

# Supervised Dimension Projection Plot
plot_projection <- textProjectionPlot(
  word_data = DP_projections_HILS_SWLS_100,
  k_n_words_to_test = FALSE,
  min_freq_words_test = 1,
  plot_n_words_square = 3,
  plot_n_words_p = 3,
  plot_n_word_extreme = 1,
  plot_n_word_frequency = 1,
  plot_n_words_middle = 1,
  y_axes = FALSE,
  p_alpha = 0.05,
  title_top = "Supervised Dimension Projection (SDP)",
  x_axes_label = "Low vs. High HILS score",
  y_axes_label = "Low vs. High SWLS score",
  p_adjust_method = "bonferroni",
  scale_y_axes_lim = NULL
)

plot_projection

# Investigate elements in DP_projections_HILS_SWLS_100.
names(DP_projections_HILS_SWLS_100)
```

`textQA` *Question Answering. (experimental)*

Description

Question Answering. (experimental)

Usage

```
textQA(
  question,
  context,
  model = "",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "warning",
  return_incorrect_results = FALSE,
  top_k = 1L,
  doc_stride = 128L,
  max_answer_len = 15L,
  max_seq_len = 384L,
  max_question_len = 64L,
  handle_impossible_answer = FALSE,
  set_seed = 202208L
)
```

Arguments

`question`	(string) A question
`context`	(string) The context(s) where the model will look for the answer.
`model`	(string) HuggingFace name of a pre-trained language model that have been fine-tuned on a question answering task.
`device`	(string) Device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number
`tokenizer_parallelism`	(boolean) If TRUE this will turn on tokenizer parallelism.
`logging_level`	(string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug
`return_incorrect_results`	(boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").
`top_k`	(integer) (int) Indicates number of possible answer span(s) to get from the model output.

doc_stride (integer) If the context is too long to fit with the question for the model, it will be split into overlapping chunks. This setting controls the overlap size.

max_answer_len (integer) Max answer size to be extracted from the model's output.

max_seq_len (integer) The max total sentence length (context + question) in tokens of each chunk passed to the model. If needed, the context is split in chunks (using doc_stride as overlap).

max_question_len (integer) The max question length after tokenization. It will be truncated if needed.

handle_impossible_answer (boolean) Whether or not impossible is accepted as an answer.

set_seed (Integer) Set seed.

Value

Answers.

See Also

see textClassify, textGeneration, textNER, textSum, textQA, textTranslate

Examples

```
#   qa_examples <- textQA(question = "Which colour have trees?",
#     context = "Trees typically have leaves, are mostly green and like water.")
```

---

textrpp_initialize *Initialize text required python packages*

Description

Initialize text required python packages to call from R.

Usage

```
textrpp_initialize(
  python_executable = NULL,
  virtualenv = NULL,
  condaenv = "textrpp_condaenv",
  ask = FALSE,
  refresh_settings = FALSE,
  save_profile = FALSE,
  check_env = TRUE,
  textEmbed_test = FALSE,
  prompt = TRUE
)
```

Arguments

python_executable	the full path to the Python executable, for which text required python packages is installed.
virtualenv	set a path to the Python virtual environment with text required python packages installed Example: virtualenv = "~/myenv"
condaenv	set a path to the anaconda virtual environment with text required python packages installed Example: condalenv = "myenv"
ask	logical; if FALSE, use the first text required python packages installation found; if TRUE, list available text required python packages installations and prompt the user for which to use. If another (e.g. python_executable) is set, then this value will always be treated as FALSE.
refresh_settings	logical; if TRUE, text will ignore the saved settings in the profile and initiate a search of new settings.
save_profile	logical; if TRUE, the current text required python packages setting will be saved for the future use.
check_env	logical; check whether conda/virtual environment generated by textrpp_install() exists
textEmbed_test	logical; Test whether function (textEmbed) that requires python packages works.
prompt	logical; asking whether user wants to set the environment as default.

---

textrpp_install *Install text required python packages in conda or virtualenv environment*

Description

Install text required python packages (rpp) in a self-contained environment. For macOS and Linux-based systems, this will also install Python itself via a "miniconda" environment, for textrpp_install. Alternatively, an existing conda installation may be used, by specifying its path. The default setting of "auto" will locate and use an existing installation automatically, or download and install one if none exists.

For Windows, automatic installation of miniconda installation is not currently available, so the user will need to install miniconda (or Anaconda) manually.

If you wish to install Python in a "virtualenv", use the textrpp_install_virtualenv function. It requires that you have a python version and path to it (such as "/usr/local/bin/python3.9" for Mac and Linux.).

Usage

```
textrpp_install(
  conda = "auto",
  update_conda = FALSE,
  force_conda = FALSE,
  rpp_version = "rpp_version_system_specific_defaults",
  python_version = "python_version_system_specific_defaults",
  envname = "textrpp_condaenv",
  pip = TRUE,
  python_path = NULL,
  prompt = TRUE
)

textrpp_install_virtualenv(
 rpp_version = c("torch==2.0.0", "transformers==4.19.2", "numpy", "pandas", "nltk"),
  python_path = NULL,
  pip_version = NULL,
  bin = "python3",
  envname = "textrpp_virtualenv",
  prompt = TRUE
)
```

Arguments

conda	character; path to conda executable. Default "auto" which automatically find the path
update_conda	Boolean; update to the latest version of Miniconda after install? (should be combined with force_conda = TRUE)
force_conda	Boolean; force re-installation if Miniconda is already installed at the requested path?
rpp_version	character; default is "rpp_version_system_specific_defaults", because diffent systems require different combinations of python version and packages. It is also possible to specify your own, such as c("torch==2.0.0", "transformers==4.19.2", "numpy", "pandas", "nltk", "scikit-learn", "datasets", "evaluate").
python_version	character; default is "python_version_system_specific_defaults". You can specify your Python version for the condaenv yourself. installation.
envname	character; name of the conda-environment to install text required python packages. Default is "textrpp_condaenv".
pip	TRUE to use pip for installing rpp If FALSE, conda package manager with conda-forge channel will be used for installing rpp.
python_path	character; path to Python only for virtualenvironment installation
prompt	logical; ask whether to proceed during the installation
pip_version	character;
bin	character; e.g., "python", only for virtualenvironment installation

Examples

```
## Not run:
# install text required python packages in a miniconda environment (macOS and Linux)
textrpp_install(prompt = FALSE)

# install text required python packages to an existing conda environment
textrpp_install(conda = "~/anaconda/bin/")

## End(Not run)
## Not run:
# install text required python packages in a virtual environment
textrpp_install_virtualenv()

## End(Not run)
```

`textrpp_uninstall` *Uninstall textrpp conda environment*

Description

Removes the conda environment created by textrpp_install()

Usage

```
textrpp_uninstall(conda = "auto", prompt = TRUE, envname = "textrpp_condaenv")
```

Arguments

conda	path to conda executable, default to "auto" which automatically finds the path
prompt	logical; ask whether to proceed during the installation
envname	character; name of conda environment to remove

`textSimilarity` *Compute the semantic similarity between two text variables.*

Description

Compute the semantic similarity between two text variables.

Usage

```
textSimilarity(x, y, method = "cosine", center = TRUE, scale = FALSE)
```

Arguments

x	Word embeddings from textEmbed.
y	Word embeddings from textEmbed.
method	(character) Character string describing type of measure to be computed. Default is "cosine" (see also "spearmen", "pearson" as well as measures from textDistance() (which here is computed as 1 - textDistance) including "euclidean", "maximum", "manhattan", "canberra", "binary" and "minkowski").
center	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.
scale	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise.

Value

A vector comprising semantic similarity scores. The closer the value is to 1 when using the default method, "cosine", the higher the semantic similarity.

See Also

See textDistance and textSimilarityNorm.

Examples

```
# Compute the semantic similarity between the embeddings from "harmonytext" and "satisfactiontext".
## Not run:
similarity_scores <- textSimilarity(
  x = word_embeddings_4$texts$harmonytext,
  y = word_embeddings_4$texts$satisfactiontext
)

# Show information about how similarity_scores were constructed.
comment(similarity_scores)

## End(Not run)
```

---

textSimilarityMatrix *Compute semantic similarity scores between all combinations in a word embedding*

Description

Compute semantic similarity scores between all combinations in a word embedding

Usage

```
textSimilarityMatrix(x, method = "cosine", center = TRUE, scale = FALSE)
```

Arguments

x	Word embeddings from textEmbed.
method	(character) Character string describing type of measure to be computed. Default is "cosine" (see also "spearmen", "pearson" as well as measures from textDistance() (which here is computed as 1 - textDistance) including "euclidean", "maximum", "manhattan", "canberra", "binary" and "minkowski").
center	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.
scale	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise.

Value

A matrix of semantic similarity scores

See Also

see textSimilarityNorm

Examples

```
similarity_scores <- textSimilarityMatrix(word_embeddings_4$texts$harmonytext[1:3, ])
round(similarity_scores, 3)
```

---

textSimilarityNorm *Compute the semantic similarity between a text variable and a word norm (i.e., a text represented by one word embedding that represent a construct).*

Description

Compute the semantic similarity between a text variable and a word norm (i.e., a text represented by one word embedding that represent a construct).

Usage

```
textSimilarityNorm(x, y, method = "cosine", center = TRUE, scale = FALSE)
```

Arguments

x	Word embeddings from textEmbed.
y	Word embedding from textEmbed (from only one text).
method	(character) Character string describing type of measure to be computed. Default is "cosine" (see also "spearmen", "pearson" as well as measures from textDistance() (which here is computed as 1 - textDistance) including "euclidean", "maximum", "manhattan", "canberra", "binary" and "minkowski").
center	(boolean; from base::scale) If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.
scale	(boolean; from base::scale) If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise.

Value

A vector comprising semantic similarity scores.

See Also

see textSimilarity

Examples

```
## Not run:
library(dplyr)
library(tibble)
harmonynorm <- c("harmony peace ")
satisfactionnorm <- c("satisfaction achievement")

norms <- tibble::tibble(harmonynorm, satisfactionnorm)
word_embeddings <- word_embeddings_4$texts
word_embeddings_wordnorm <- textEmbed(norms)
similarity_scores <- textSimilarityNorm(
  word_embeddings$harmonytext,
  word_embeddings_wordnorm$harmonynorm
)

## End(Not run)
```

---

textSum *Summarize texts. (experimental)*

Description

Summarize texts. (experimental)

Usage

```
textSum(
  x,
  min_length = 10L,
  max_length = 20L,
  model = "t5-small",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "warning",
  return_incorrect_results = FALSE,
  return_text = TRUE,
  return_tensors = FALSE,
  clean_up_tokenization_spaces = FALSE,
  set_seed = 202208L
)
```

Arguments

x	(string) A variable or a tibble/dataframe with at least one character variable.
min_length	(explicit integer; e.g., 10L) The minimum number of tokens in the summed output.
max_length	(explicit integer higher than min_length; e.g., 20L) The maximum number of tokens in the summed output.
model	(string) Specififcation of a pre-trained language model that have been fine-tuned on a summarization task, such as 'bart-large-cnn', 't5-small', 't5-base', 't5-large', 't5-3b', 't5-11b'.
device	(string) Device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number.
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism.
logging_level	(string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug
return_incorrect_results	(boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").
return_text	(boolean) Whether or not the outputs should include the decoded text.
return_tensors	(boolean) Whether or not the output should include the prediction tensors (as token indices).
clean_up_tokenization_spaces	(boolean) Option to clean up the potential extra spaces in the returned text.
set_seed	(Integer) Set seed.

Value

A tibble with summed text(s).

See Also

see textClassify, textGeneration, textNER, textSum, textQA, textTranslate

Examples

```
# sum_examples <- textSum(Language_based_assessment_data_8[1:2,1:2],
# min_length = 5L,
# max_length = 10L)
```

---

textTokenize *Tokenize according to different huggingface transformers*

Description

Tokenize according to different huggingface transformers

Usage

```
textTokenize(
  texts,
  model = "bert-base-uncased",
  max_token_to_sentence = 4,
  device = "cpu",
  tokenizer_parallelism = FALSE,
  model_max_length = NULL,
  logging_level = "error"
)
```

Arguments

texts	A character variable or a tibble/dataframe with at least one character variable.
model	Character string specifying pre-trained language model (default 'bert-base-uncased'). For full list of options see pretrained models at HuggingFace. For example use "bert-base-multilingual-cased", "openai-gpt", "gpt2", "ctrl", "transfo-xl-wt103", "xlnet-base-cased", "xlm-mlm-enfr-1024", "distilbert-base-cased", "roberta-base", or "xlm-roberta-base".
max_token_to_sentence	(numeric) Maximum number of tokens in a string to handle before switching to embedding text sentence by sentence.
device	Name of device to use: 'cpu', 'gpu', 'gpu:k' or 'mps'/'mps:k' for MacOS, where k is a specific device number.

tokenizer_parallelism
: If TRUE this will turn on tokenizer parallelism. Default FALSE.

model_max_length
: The maximum length (in number of tokens) for the inputs to the transformer model (default the value stored for the associated model).

logging_level
: Set the logging level. Default: "warning". Options (ordered from less logging to more logging): critical, error, warning, info, debug

Value

Returns tokens according to specified huggingface transformer.

See Also

see textEmbed

Examples

```
# tokens <- textTokenize("hello are you?")
```

---

textTopics *This function creates and trains a BERTopic model (based on bertopic python packaged) on a text-variable in a tibble/data.frame. (EXPERIMENTAL)*

Description

This function creates and trains a BERTopic model (based on bertopic python packaged) on a text-variable in a tibble/data.frame. (EXPERIMENTAL)

Usage

```
textTopics(
  data,
  variable_name,
  embedding_model = "distilroberta",
  umap_model = "default",
  hdbscan_model = "default",
  vectorizer_model = "default",
  representation_model = "mmr",
  num_top_words = 10,
  n_gram_range = c(1, 3),
  stopwords = "english",
  min_df = 5,
  bm25_weighting = FALSE,
  reduce_frequent_words = TRUE,
```

```
  set_seed = 8,
  save_dir = "./results"
)
```

Arguments

data	(tibble/data.frame) A tibble with a text-variable to be analysed, and optional numeric/categorical variables that you might want to use for later analyses testing the significance of topics in relation to these variables.
variable_name	(string) Name of the text-variable in the data tibble that you want to perform topic modeling on.
embedding_model	(string) Name of the embedding model to use such as "miniLM", "mpnet", "multi-mpnet", "distilroberta".
umap_model	(string) The dimension reduction algorithm, currently only "default" is supported.
hdbscan_model	(string) The clustering algorithm to use, currently only "default" is supported.
vectorizer_model	(string) Name of the vectorizer model, currently only "default" is supported.
representation_model	(string) Name of the representation model used for topics, including "keybert" or "mmr".
num_top_words	(integer) Determine the number of top words presented for each topic.
n_gram_range	(vector) Two-dimensional vector indicating the ngram range used for the vectorizer model.
stopwords	(string) Name of the stopword dictionary to use.
min_df	(integer) The minimum document frequency of terms.
bm25_weighting	(boolean) Determine whether bm25_weighting is used for ClassTfidfTransformer.
reduce_frequent_words	(boolean) Determine whether frequent words are reduced by ClassTfidfTransformer.
set_seed	(integer) The random seed for initialization of the umap model.
save_dir	(string) The directory for saving results.

Value

A folder containing the model, data, folder with terms and values for each topic, and the document-topic matrix. Moreover the model itself is returned formatted as a data.frame together with metdata. See textTopicsReduce textTopicsTest and textTopicsWordcloud.

`textTopicsReduce` *textTopicsReduce (EXPERIMENTAL)*

Description

textTopicsReduce (EXPERIMENTAL)

Usage

```
textTopicsReduce(
  data,
  data_var,
  n_topics = 10,
  load_path = "./results",
  save_dir = "./results_reduced",
  embedding_model = "default"
)
```

Arguments

`data`	(tibble/data.frame) A tibble with a text-variable to be analysed, and optional numeric/categorical variables that you might want to use for later analyses testing the significance of topics in relation to these variables.
`data_var`	(string) Name of the text-variable in the data tibble that you want to perform topic modeling on.
`n_topics`	(string) The dimension reduction algorithm, currently only "default" is supported.
`load_path`	(string) The clustering algorithm to use, currently only "default" is supported.
`save_dir`	(string) The directory for saving results.
`embedding_model`	(string) Name of the embedding model to use such as "miniLM", "mpnet", "multi-mpnet", "distilroberta".

Value

A folder containing the model, data, folder with terms and values for each topic, and the document-topic matrix. Moreover the model itself is returned formatted as a data.frame together with metdata.

See Also

See `textTopics` `textTopicsTest` and `textTopicsWordcloud`.

`textTopicsTest` *This function tests the relationship between a single topic or all topics and a variable of interest. Available tests include correlation, t-test, linear regression, binary regression, and ridge regression. (EXPERIMENTAL - under development)*

Description

This function tests the relationship between a single topic or all topics and a variable of interest. Available tests include correlation, t-test, linear regression, binary regression, and ridge regression. (EXPERIMENTAL - under development)

Usage

```
textTopicsTest(
  model,
  pred_var,
  group_var = NULL,
  control_vars = c(),
  test_method = "linear_regression",
  multiple_comparison = "fdr",
  load_dir = NULL
)
```

Arguments

`model`	(data.frame) The model returned from textTopics().
`pred_var`	(string) Variable of interest for linear or binary regression
`group_var`	(string) Grouping variable for t-test
`control_vars`	(list) Control variables for linear or binary regression
`test_method`	(string) Choose between "correlation", "t-test", "binary_regression", "linear_regression" or "ridge_regression"
`multiple_comparison`	Method for correction of multiple tests (e.g., "fdr", "bonferroni").
`load_dir`	(string) if specified, the function returns the precomputed analysis from the directory, otherwise leave blank

Value

Metadata and results of the test such as estimate, t-value, p-value, and variable name.

textTopicsTree *textTopicsTest (EXPERIMENTAL) to get the hierarchical topic tree*

Description

textTopicsTest (EXPERIMENTAL) to get the hierarchical topic tree

Usage

```
textTopicsTree(topic_model, data, data_var)
```

Arguments

topic_model	(list) The output from textTopics.
data	(tibble/data.frame) A tibble with the data
data_var	(string) The name of the text variable that the topic model was trained on

Value

prints a hierarchical topic tree on the console

textTopicsWordcloud *This functions plots wordclouds of topics from a Topic Model based on their significance determined by a linear or binary regression*

Description

This functions plots wordclouds of topics from a Topic Model based on their significance determined by a linear or binary regression

Usage

```
textTopicsWordcloud(
  model,
  test,
 color_negative_cor = ggplot2::scale_color_gradient(low = "darkred", high = "red"),
 color_positive_cor = ggplot2::scale_color_gradient(low = "darkgreen", high = "green"),
  scale_size = FALSE,
  plot_topics_idx = NULL,
  p_threshold = 0.05
)
```

Arguments

model	(data.frame) The model returned from textTopics().
test	(data.frame) the test returned from textTopicTest()
color_negative_cor	(ggplot2::scale_color_gradient()) color gradient of topic cloud with negative correlation
color_positive_cor	(ggplot2::scale_color_gradient) color gradient of topic cloud with positive correlation
scale_size	(bool) if True, then the size of the topic cloud is scaled by the prevalence of the topic
plot_topics_idx	(list) if specified, then only the specified topics are plotted
p_threshold	(float) set significance threshold which determines which topics are plotted

---

textTrain *Train word embeddings to a numeric (ridge regression) or categorical (random forest) variable.*

Description

Train word embeddings to a numeric (ridge regression) or categorical (random forest) variable.

Usage

```
textTrain(x, y, force_train_method = "automatic", ...)
```

Arguments

x	Word embeddings from textEmbed (or textEmbedLayerAggreation). Can analyze several variables at the same time; but if training to several outcomes at the same time use a tibble within the list as input rather than just a tibble input (i.e., keep the name of the wordembedding).
y	Numeric variable to predict. Can be several; although then make sure to have them within a tibble (this is required even if it is only one outcome but several word embeddings variables).
force_train_method	Default is "automatic", so if y is a factor random_forest is used, and if y is numeric ridge regression is used. This can be overridden using "regression" or "random_forest".
...	Arguments from textTrainRegression or textTrainRandomForest the textTrain function.

Value

A correlation between predicted and observed values; as well as a tibble of predicted values (t-value, degree of freedom (df), p-value, alternative-hypothesis, confidence interval, correlation coefficient).

See Also

See textTrainRegression, textTrainRandomForest and textTrainLists.

Examples

```
# Examines how well the embeddings from "harmonytext" can
# predict the numeric variable "hilstotal" in the pre-included
# dataset "Language_based_assessment_data_8".

## Not run:
trained_model <- textTrain(
  x = word_embeddings_4$texts$harmonytext,
  y = Language_based_assessment_data_8$hilstotal
)

# Examine results (t-value, degree of freedom (df), p-value,
# alternative-hypothesis, confidence interval, correlation coefficient).

trained_model$results

## End(Not run)
```

---

textTrainLists *Individually trains word embeddings from several text variables to several numeric or categorical variables.*

Description

Individually trains word embeddings from several text variables to several numeric or categorical variables.

Usage

```
textTrainLists(
  x,
  y,
  force_train_method = "automatic",
  save_output = "all",
  method_cor = "pearson",
  eval_measure = "rmse",
  p_adjust_method = "holm",
  ...
)
```

Arguments

x	Word embeddings from textEmbed (or textEmbedLayerAggreation). It is possible to have word embeddings from one text variable and several numeric/categorical variables; or vice verse, word embeddings from several text variables to one numeric/categorical variable. It is not possible to mix numeric and categorical variables.
y	Tibble with several numeric or categorical variables to predict. Please note that you cannot mix numeric and categorical variables.
force_train_method	(character) Default is "automatic"; see also "regression" and "random_forest".
save_output	(character) Option not to save all output; default "all". See also "only_results" and "only_results_predictions".
method_cor	(character) A character string describing type of correlation (default "Pearson").
eval_measure	(character) Type of evaluative measure to assess models on (default "rmse").
p_adjust_method	Method to adjust/correct p-values for multiple comparisons. (default = "holm"; see also "none", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr").
...	Arguments from textTrainRegression or textTrainRandomForest (the textTrain function).

Value

Correlations between predicted and observed values (t-value, degree of freedom (df), p-value, confidence interval, alternative hypothesis, correlation coefficient) stored in a dataframe.

See Also

See textTrain, textTrainRegression and textTrainRandomForest.

Examples

```
# Examines how well the embeddings from Language_based_assessment_data_8 can
# predict the numerical numerical variables in Language_based_assessment_data_8.
# The training is done combination wise, i.e., correlations are tested pair wise,
# column: 1-5,1-6,2-5,2-6, resulting in a dataframe with four rows.

## Not run:
word_embeddings <- word_embeddings_4$texts[1:2]
ratings_data <- Language_based_assessment_data_8[5:6]

trained_model <- textTrainLists(
  x = word_embeddings,
  y = ratings_data
)

# Examine results (t-value, degree of freedom (df), p-value,
# alternative-hypothesis, confidence interval, correlation coefficient).
```

```
trained_model$results

## End(Not run)
```

`textTrainN` *(experimental) Compute cross-validated correlations for different sample-sizes of a data set. The cross-validation process can be repeated several times to enhance the reliability of the evaluation.*

Description

(experimental) Compute cross-validated correlations for different sample-sizes of a data set. The cross-validation process can be repeated several times to enhance the reliability of the evaluation.

Usage

```
textTrainN(
  x,
  y,
  sample_percents = c(25, 50, 75, 100),
  handle_word_embeddings = "individually",
  n_cross_val = 1,
  x_append = NULL,
  append_first = FALSE,
  cv_method = "validation_split",
  outside_folds = 10,
  inside_folds = 3/4,
  strata = "y",
  outside_strata = TRUE,
  outside_breaks = 4,
  inside_strata = TRUE,
  inside_breaks = 4,
  model = "regression",
  eval_measure = "default",
  preprocess_step_center = TRUE,
  preprocess_step_scale = TRUE,
  preprocess_PCA = NA,
  penalty = 10^seq(-16, 16),
  mixture = c(0),
  first_n_predictors = NA,
  impute_missing = FALSE,
  method_cor = "pearson",
  model_description = "Consider writing a description of your model here",
  multi_cores = "multi_cores_sys_default",
  save_output = "all",
  simulate.p.value = FALSE,
```

```
  seed = 2024
)
```

Arguments

x	Word embeddings from textEmbed (or textEmbedLayerAggregation). If several word embedding are provided in a list they will be concatenated.
y	Numeric variable to predict.
sample_percents	(numeric) Numeric vector that specifies the percentages of the total number of data points to include in each sample (default = c(25,50,75,100), i.e., correlations are evaluated for 25 each new sample.
handle_word_embeddings	Determine whether to use a list of word embeddings or an individual word_embedding (default = "individually", also "concatenate"). If a list of word embeddings are provided, then they will be concatenated.
n_cross_val	(numeric) Value that determines the number of times to repeat the cross-validation. (default = 1, i.e., cross-validation is only performed once). Warning: The training process gets proportionately slower to the number of cross-validations, resulting in a time complexity that increases with a factor of n (n cross-validations).
x_append	(optional) Variables to be appended after the word embeddings (x); if wanting to preappend them before the word embeddings use the option first = TRUE. If not wanting to train with word embeddings, set x = NULL (default = NULL).
append_first	(boolean) Option to add variables before or after all word embeddings (default = False).
cv_method	(character) Cross-validation method to use within a pipeline of nested outer and inner loops of folds (see nested_cv in rsample). Default is using cv_folds in the outside folds and "validation_split" using rsample::validation_split in the inner loop to achieve a development and assessment set (note that for validation_split the inside_folds should be a proportion, e.g., inside_folds = 3/4); whereas "cv_folds" uses rsample::vfold_cv to achieve n-folds in both the outer and inner loops.
outside_folds	(numeric) Number of folds for the outer folds (default = 10).
inside_folds	(numeric) The proportion of data to be used for modeling/analysis; (default proportion = 3/4). For more information see validation_split in rsample.
strata	(string or tibble; default "y") Variable to stratify according; if a string the variable needs to be in the training set - if you want to stratify according to another variable you can include it as a tibble (please note you can only add 1 variable to stratify according). Can set it to NULL.
outside_strata	(boolean) Whether to stratify the outside folds.
outside_breaks	(numeric) The number of bins wanted to stratify a numeric stratification variable in the outer cross-validation loop (default = 4).
inside_strata	Whether to stratify the outside folds.
inside_breaks	The number of bins wanted to stratify a numeric stratification variable in the inner cross-validation loop (default = 4).

`model` Type of model. Default is "regression"; see also "logistic" and "multinomial" for classification.

`eval_measure` (character) Type of evaluative measure to select models from. Default = "rmse" for regression and "bal_accuracy" for logistic. For regression use "rsq" or "rmse"; and for classification use "accuracy", "bal_accuracy", "sens", "spec", "precision", "kappa", "f_measure", or "roc_auc",(for more details see the yardstick package).

`preprocess_step_center`
(boolean) Normalizes dimensions to have a mean of zero; default is set to TRUE. For more info see (step_center in recipes).

`preprocess_step_scale`
(boolean) Normalize dimensions to have a standard deviation of one; default is set to TRUE. For more info see (step_scale in recipes).

`preprocess_PCA` Pre-processing threshold for PCA (to skip this step set it to NA). Can select amount of variance to retain (e.g., .90 or as a grid c(0.80, 0.90)); or number of components to select (e.g., 10). Default is "min_halving", which is a function that selects the number of PCA components based on number of participants and feature (word embedding dimensions) in the data. The formula is: preprocess_PCA = round(max(min(number_features/2), number_participants/2), min(50, number_features))).

`penalty` (numeric) Hyper parameter that is tuned (default = 10^seq(-16,16)).

`mixture` A number between 0 and 1 (inclusive) that reflects the proportion of L1 regularization (i.e. lasso) in the model (for more information see the linear_reg-function in the parsnip-package). When mixture = 1, it is a pure lasso model while mixture = 0 indicates that ridge regression is being used (specific engines only).

`first_n_predictors`
By default this setting is turned off (i.e., NA). To use this method, set it to the highest number of predictors you want to test. Then the X first dimensions are used in training, using a sequence from Kjell et al., 2019 paper in Psychological Methods. Adding 1, then multiplying by 1.3 and finally rounding to the nearest integer (e.g., 1, 3, 5, 8). This option is currently only possible for one embedding at the time.

`impute_missing` Default FALSE (can be set to TRUE if something else than word_embeddings are trained).

`method_cor` Type of correlation used in evaluation (default "pearson"; can set to "spearman" or "kendall").

`model_description`
(character) Text to describe your model (optional; good when sharing the model with others).

`multi_cores` If TRUE it enables the use of multiple cores if the computer system allows for it (i.e., only on unix, not windows). Hence it makes the analyses considerably faster to run. Default is "multi_cores_sys_default", where it automatically uses TRUE for Mac and Linux and FALSE for Windows.

`save_output` (character) Option not to save all output; default = "all". see also "only_results" and "only_results_predictions".

simulate.p.value
: (Boolean) From fisher.test: a logical indicating whether to compute p-values by Monte Carlo simulation, in larger than 2 × 2 tables.

seed
: (numeric) Set different seed (default = 2024).

Value

A tibble containing correlations for each sample. If n_cross_val > 1, correlations for each new cross-validation, along with standard-deviation and mean correlation is included in the tibble. The information in the tibble is visualised via the textTrainNPlot function.

See Also

See textTrainNPlot.

Examples

```
# Compute correlations for 25%, 50%, 75% and 100% of the data in word_embeddings and perform
# cross-validation thrice.

## Not run:
tibble_to_plot <- textTrainN(
  x = word_embeddings_4$texts$harmonytext,
  y = Language_based_assessment_data_8$hilstotal,
  sample_percents = c(25, 50, 75, 100),
  n_cross_val = 3
)

# tibble_to_plot contains correlation-coefficients for each cross_validation and
# standard deviation and mean value for each sample. The tibble can be plotted
# using the testTrainNPlot function.

# Examine tibble
tibble_to_plot

## End(Not run)
```

---

textTrainNPlot

(experimental) Plot cross-validated correlation coefficients across different sample-sizes from the object returned by the textTrainN function. If the number of cross-validations exceed one, then error-bars will be included in the plot.

Description

(experimental) Plot cross-validated correlation coefficients across different sample-sizes from the object returned by the textTrainN function. If the number of cross-validations exceed one, then error-bars will be included in the plot.

Usage

```
textTrainNPlot(
  train_data,
  sample_percents = c(25, 50, 75, 100),
  n_cross_val = rep(1, length(tibble_list)),
  x_unit = "percent",
  y_range = NULL,
 title = "Cross-validated correlation coefficients across different sample sizes",
  x_axes_label = "Sample Size (percent)",
  y_axes_label = "Correlation Coefficient (r)",
  point_color = rep("#5dc688", length(tibble_list)),
  bar_color = rep("#60A1F7", length(tibble_list)),
  line_color = rep("grey", length(tibble_list)),
  bar_width = rep(1, length(tibble_list)),
  bar_size = rep(0.8, length(tibble_list)),
  line_size = rep(0.6, length(tibble_list)),
  line_type = rep("straight", length(tibble_list)),
  point_size = rep(3, length(tibble_list)),
  log_transform_x = FALSE
)
```

Arguments

train_data	(list) One or several objects returned by the function textTrainN as a list (e.g, list(object1, object2)). Also, If several models are provided, then one can add a vector c() with settings (i.e the parameters below) for each model (make sure to add the settings in the order as the models are ordered, if you look to keep the original settings then write "").
sample_percents	(numeric) Vector containing the percents of the total number of datapoints that is included in each sample (default = c(25,50,75,100)).
n_cross_val	(numeric) Value of the number of times cross-validation has been repeated (default = 1, i.e., cross-validation has only been applied once). If several models are provided, then one can add a vector c() with settings for each model (make sure to add the settings in the order as the models are ordered).
x_unit	(character, "percent" or "quantity") Determines whether the x-axis-values should represent the number of elements in each sample, or the number of percent of the total data they represent (default = "percent").
y_range	(numeric) Optional. Determines the y_range. E.g, y_range = c(1,2) sets the y_range from 1 to 2 (default = NULL).
title	(character) Determine plot title (default = "Cross-validated correlation coefficients across different sample sizes").
x_axes_label	(character) Determine x-axis-label (default = "Sample Size (percent)").
y_axes_label	(character) Determine y-axis-label (default = "Correlation Coefficient (r)").
point_color	(character, (Hex color codes)) Determine point color (default = "#5dc688").
bar_color	(character, (Hex color codes)) Determine error-bar color (default = "#60A1F7").

line_color	(character, (Hex color codes)) Determine line color (default = "grey").
bar_width	(numeric) Determine bar-width (default = 1).
bar_size	(numeric) Determine bar-size (default = 1).
line_size	(numeric) Determine line-size (default = 1).
line_type	(character, either "straight" or "smooth") Determine line-type (default = "straight").
point_size	(numeric) Determine points size (default = 1).
log_transform_x	(boolean) Determine wether to log-transform x in case of displaying number of samples (default = FALSE).

Value

A plot with correlation coefficient on y-axis and sample size in quantity or percent on x axis. If number och cross-validations exceed 1, then error bars measuring standard deviations will be plotted.

Plot Example

Example of a plot created by textTrainNPlot.

See Also

See textTrainN.

Examples

```
# Plot cross-validated correlation coefficients across different sample-sizes from the object
# returned by the textTrainN function.

## Not run:
# Plot the performance of a single model across different sample sizes
plot_object1 <- textTrainNPlot(
  train_data = tibble_to_plot,
  n_cross_val = 3,
  x_unit = "quantity"
)

# Visualize plot
plot_object1

# Plot the performance of several models across different sample sizes.
plot_object2 <- textTrainNPlot(train_data = list(object1, object2, object3),
                               n_cross_val = c(2,1,1),
                          line_color = c("","","#0000FF")) # "" gives the default settings.
# Visualize plot
plot_object2

## End(Not run)
```

`textTrainRandomForest` *Train word embeddings to a categorical variable using random forest.*

Description

Train word embeddings to a categorical variable using random forest.

Usage

```
textTrainRandomForest(
  x,
  y,
  x_append = NULL,
  append_first = FALSE,
  cv_method = "validation_split",
  outside_folds = 10,
  inside_folds = 3/4,
  strata = "y",
  outside_strata = TRUE,
  outside_breaks = 4,
  inside_strata = TRUE,
  inside_breaks = 4,
  mode_rf = "classification",
  preprocess_step_center = FALSE,
  preprocess_scale_center = FALSE,
  preprocess_PCA = NA,
  extremely_randomised_splitrule = "extratrees",
  mtry = c(1, 10, 20, 40),
  min_n = c(1, 10, 20, 40),
  trees = c(1000),
  eval_measure = "bal_accuracy",
  model_description = "Consider writing a description of your model here",
  multi_cores = "multi_cores_sys_default",
  save_output = "all",
  simulate.p.value = FALSE,
  seed = 2020,
  ...
)
```

Arguments

x	Word embeddings from textEmbed.
y	Categorical variable to predict.
x_append	(optional) Variables to be appended after the word embeddings (x); if wanting to preappend them before the word embeddings use the option first = TRUE. If not wanting to train with word embeddings, set x_append = NULL (default = null).

append_first (boolean) Option to add variables before or after all word embeddings (default = FALSE).

cv_method (character) Cross-validation method to use within a pipeline of nested outer and inner loops of folds (see nested_cv in rsample). Default is using cv_folds in the outside folds and "validation_split" using rsample::validation_split in the inner loop to achieve a development and assessment set (note that for validation_split the inside_folds should be a proportion, e.g., inside_folds = 3/4); whereas "cv_folds" uses rsample::vfold_cv to achieve n-folds in both the outer and inner loops.

outside_folds (numeric) Number of folds for the outer folds (default = 10).

inside_folds (numeric) Number of folds for the inner folds (default = 3/4).

strata (string or tibble; default "y") Variable to stratify according; if a string the variable needs to be in the training set - if you want to stratify according to another variable you can include it as a tibble (please note you can only add 1 variable to stratify according). Can set it to NULL.

outside_strata (boolean) Whether to stratify the outside folds.

outside_breaks (numeric) The number of bins wanted to stratify a numeric stratification variable in the outer cross-validation loop (default = 4).

inside_strata (boolean) Whether to stratify the outside folds.

inside_breaks The number of bins wanted to stratify a numeric stratification variable in the inner cross-validation loop (default = 4).

mode_rf Default is "classification" ("regression" is not supported yet).

preprocess_step_center
(boolean) Normalizes dimensions to have a mean of zero; default is set to FALSE For more info see (step_center in recipes).

preprocess_scale_center
(boolean) Normalizes dimensions to have a standard deviation of one; default is set to FALSE. For more info see (step_scale in recipes).

preprocess_PCA Pre-processing threshold for PCA. Can select amount of variance to retain (e.g., .90 or as a grid c(0.80, 0.90)); or number of components to select (e.g., 10). (To skip this step, set preprocess_PCA to NA) Default is "min_halving", which is a function that selects the number of PCA components based on number of participants and feature (word embedding dimensions) in the data. The formula is: preprocess_PCA = round(max(min(number_features/2), number_participants/2), min(50, number_features))).

extremely_randomised_splitrule
Default is "extratrees", which thus implement a random forest; can also select: NULL, "gini" or "hellinger"; if these are selected your mtry settings will be overridden (see Geurts et al. (2006) Extremely randomized trees for details; and see the ranger r-package for details on implementations).

mtry Hyper parameter that may be tuned; default: c(1, 20, 40),

min_n Hyper parameter that may be tuned; default: c(1, 20, 40)

trees Number of trees to use (default 1000).

`eval_measure` (character) Measure to evaluate the models in order to select the best hyperparameters default "roc_auc"; see also "accuracy", "bal_accuracy", "sens", "spec", "precision", "kappa", "f_measure".

`model_description` (character) Text to describe your model (optional; good when sharing the model with others).

`multi_cores` If TRUE it enables the use of multiple cores if the computer system allows for it (i.e., only on unix, not windows). Hence it makes the analyses considerably faster to run. Default is "multi_cores_sys_default", where it automatically uses TRUE for Mac and Linux and FALSE for Windows. Note that having it to TRUE does not enable reproducable results at the moment (i.e., cannot set seed).

`save_output` (character) Option not to save all output; default "all". See also "only_results" and "only_results_predictions".

`simulate.p.value` (Boolean) From fisher.test: a logical indicating whether to compute p-values by Monte Carlo simulation, in larger than 2 × 2 tables.

`seed` (numeric) Set different seed (default = 2020).

`...` For example settings in yardstick::accuracy to set event_level (e.g., event_level = "second").

Value

A list with roc_curve_data, roc_curve_plot, truth and predictions, preprocessing_recipe, final_model, model_description chisq and fishers test as well as evaluation measures, e.g., including accuracy, f_meas and roc_auc (for details on these measures see the yardstick r-package documentation).

See Also

See `textEmbedLayerAggregation`, `textTrainLists` and `textTrainRegression`.

Examples

```
# Examines how well the embeddings from column "harmonywords" in
# Language_based_assessment_data_8 can binarily classify gender.

## Not run:
trained_model <- textTrainRandomForest(
  x = word_embeddings_4$texts$harmonywords,
  y = as.factor(Language_based_assessment_data_8$gender),
  trees = c(1000, 1500),
  mtry = c(1), # this is short because of testing
  min_n = c(1), # this is short because of testing
  multi_cores = FALSE # This is FALSE due to CRAN testing and Windows machines.
)


# Examine results (t-value, degree of freedom (df), p-value,
# alternative-hypothesis, confidence interval, correlation coefficient).
```

```
trained_model$results

## End(Not run)
```

`textTrainRegression` *Train word embeddings to a numeric variable.*

Description

Train word embeddings to a numeric variable.

Usage

```
textTrainRegression(
  x,
  y,
  x_append = NULL,
  append_first = FALSE,
  cv_method = "validation_split",
  outside_folds = 10,
  inside_folds = 3/4,
  strata = "y",
  outside_strata = TRUE,
  outside_breaks = 4,
  inside_strata = TRUE,
  inside_breaks = 4,
  model = "regression",
  eval_measure = "default",
  preprocess_step_center = TRUE,
  preprocess_step_scale = TRUE,
  preprocess_PCA = NA,
  penalty = 10^seq(-16, 16),
  mixture = c(0),
  first_n_predictors = NA,
  impute_missing = FALSE,
  method_cor = "pearson",
  model_description = "Consider writing a description of your model here",
  multi_cores = "multi_cores_sys_default",
  save_output = "all",
  simulate.p.value = FALSE,
  seed = 2020,
  ...
)
```

Arguments

`x`	Word embeddings from textEmbed (or textEmbedLayerAggregation). If several word embedding are provided in a list they will be concatenated.

y	Numeric variable to predict.
x_append	(optional) Variables to be appended after the word embeddings (x); if wanting to preappend them before the word embeddings use the option first = TRUE. If not wanting to train with word embeddings, set x = NULL (default = NULL).
append_first	(boolean) Option to add variables before or after all word embeddings (default = False).
cv_method	(character) Cross-validation method to use within a pipeline of nested outer and inner loops of folds (see nested_cv in rsample). Default is using cv_folds in the outside folds and "validation_split" using rsample::validation_split in the inner loop to achieve a development and assessment set (note that for validation_split the inside_folds should be a proportion, e.g., inside_folds = 3/4); whereas "cv_folds" uses rsample::vfold_cv to achieve n-folds in both the outer and inner loops.
outside_folds	(numeric) Number of folds for the outer folds (default = 10).
inside_folds	(numeric) The proportion of data to be used for modeling/analysis; (default proportion = 3/4). For more information see validation_split in rsample.
strata	(string or tibble; default "y") Variable to stratify according; if a string the variable needs to be in the training set - if you want to stratify according to another variable you can include it as a tibble (please note you can only add 1 variable to stratify according). Can set it to NULL.
outside_strata	(boolean) Whether to stratify the outside folds.
outside_breaks	(numeric) The number of bins wanted to stratify a numeric stratification variable in the outer cross-validation loop (default = 4).
inside_strata	Whether to stratify the outside folds.
inside_breaks	The number of bins wanted to stratify a numeric stratification variable in the inner cross-validation loop (default = 4).
model	Type of model. Default is "regression"; see also "logistic" and "multinomial" for classification.
eval_measure	(character) Type of evaluative measure to select models from. Default = "rmse" for regression and "bal_accuracy" for logistic. For regression use "rsq" or "rmse"; and for classification use "accuracy", "bal_accuracy", "sens", "spec", "precision", "kappa", "f_measure", or "roc_auc",(for more details see the yardstick package).
preprocess_step_center	(boolean) Normalizes dimensions to have a mean of zero; default is set to TRUE. For more info see (step_center in recipes).
preprocess_step_scale	(boolean) Normalize dimensions to have a standard deviation of one; default is set to TRUE. For more info see (step_scale in recipes).
preprocess_PCA	Pre-processing threshold for PCA (to skip this step set it to NA). Can select amount of variance to retain (e.g., .90 or as a grid c(0.80, 0.90)); or number of components to select (e.g., 10). Default is "min_halving", which is a function that selects the number of PCA components based on number of participants and feature (word embedding dimensions) in the data. The formula is: preprocess_PCA = round(max(min(number_features/2), number_participants/2), min(50, number_features))).

`penalty` (numeric) Hyper parameter that is tuned (default = 10^seq(-16,16)).

`mixture` A number between 0 and 1 (inclusive) that reflects the proportion of L1 regularization (i.e. lasso) in the model (for more information see the linear_reg-function in the parsnip-package). When mixture = 1, it is a pure lasso model while mixture = 0 indicates that ridge regression is being used (specific engines only).

`first_n_predictors` By default this setting is turned off (i.e., NA). To use this method, set it to the highest number of predictors you want to test. Then the X first dimensions are used in training, using a sequence from Kjell et al., 2019 paper in Psychological Methods. Adding 1, then multiplying by 1.3 and finally rounding to the nearest integer (e.g., 1, 3, 5, 8). This option is currently only possible for one embedding at the time.

`impute_missing` Default FALSE (can be set to TRUE if something else than word_embeddings are trained).

`method_cor` Type of correlation used in evaluation (default "pearson"; can set to "spearman" or "kendall").

`model_description` (character) Text to describe your model (optional; good when sharing the model with others).

`multi_cores` If TRUE it enables the use of multiple cores if the computer system allows for it (i.e., only on unix, not windows). Hence it makes the analyses considerably faster to run. Default is "multi_cores_sys_default", where it automatically uses TRUE for Mac and Linux and FALSE for Windows.

`save_output` (character) Option not to save all output; default = "all". see also "only_results" and "only_results_predictions".

`simulate.p.value` (Boolean) From fisher.test: a logical indicating whether to compute p-values by Monte Carlo simulation, in larger than 2 × 2 tables.

`seed` (numeric) Set different seed (default = 2020).

`...` For example settings in yardstick::accuracy to set event_level (e.g., event_level = "second").

Details

By default, NAs are treated as follows: 1. rows with NAs in word embeddings are removed. 2. rows with NAs in y are removed 3. rows with NAs in x_append are removed; if impute_missing is set to TRUE, missing values will be imputed using k-nearest neighbours. When rows are omitted, the user will get a warning. The CV predictions will include NAs with the same length as the input.

Value

A (one-sided) correlation test between predicted and observed values; tibble of predicted values (t-value, degree of freedom (df), p-value, alternative-hypothesis, confidence interval, correlation coefficient), as well as information about the model (preprossing_recipe, final_model and model_description).

See Also

See textEmbedLayerAggregation, textTrainLists and textTrainRandomForest.

Examples

```
# Examines how well the embeddings from the column "harmonytext" can
# predict the numerical values in the column "hilstotal".

## Not run:
trained_model <- textTrainRegression(
  x = word_embeddings_4$texts$harmonytext,
  y = Language_based_assessment_data_8$hilstotal,
  multi_cores = FALSE # This is FALSE due to CRAN testing and Windows machines.
)

# Examine results (t-value, degree of freedom (df), p-value, alternative-hypothesis,
# confidence interval, correlation coefficient).

trained_model$results

## End(Not run)
```

---

textTranslate *Translation. (experimental)*

Description

Translation. (experimental)

Usage

```
textTranslate(
  x,
  source_lang = "",
  target_lang = "",
  model = "xlm-roberta-base",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "warning",
  return_incorrect_results = FALSE,
  return_tensors = FALSE,
  return_text = TRUE,
  clean_up_tokenization_spaces = FALSE,
  set_seed = 202208L,
  max_length = 400
)
```

Arguments

x	(string) The text to be translated.
source_lang	(string) The input language. Might be needed for multilingual models (it will not have any effect for single pair translation models). using ISO 639-1 Code, such as: "en", "zh", "es", "fr", "de", "it", "sv", "da", "nn".
target_lang	(string) The desired language output. Might be required for multilingual models (will not have any effect for single pair translation models).
model	(string) Specify a pre-trained language model that have been fine-tuned on a translation task.
device	(string) Name of device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number
tokenizer_parallelism	(boolean) If TRUE this will turn on tokenizer parallelism.
logging_level	(string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug
return_incorrect_results	(boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").
return_tensors	(boolean) Whether or not to include the predictions' tensors as token indices in the outputs.
return_text	(boolean) Whether or not to also output the decoded texts.
clean_up_tokenization_spaces	(boolean) Whether or not to clean the output from potential extra spaces.
set_seed	(Integer) Set seed.
max_length	Set max length of text to be translated

Value

A tibble with transalted text.

See Also

see textClassify, textGeneration, textNER, textSum, and textQA

Examples

```
# translation_example <- text::textTranslate(
#  Language_based_assessment_data_8[1,1:2],
#  source_lang = "en",
#  target_lang = "fr",
#  model = "t5-base")
```

`textWordPrediction` *Compute predictions based on single words for plotting words. The word embeddings of single words are trained to predict the mean value associated with that word. P-values does NOT work yet (experimental).*

Description

Compute predictions based on single words for plotting words. The word embeddings of single words are trained to predict the mean value associated with that word. P-values does NOT work yet (experimental).

Usage

```
textWordPrediction(
  words,
  word_types_embeddings = word_types_embeddings_df,
  x,
  y = NULL,
  seed = 1003,
  case_insensitive = TRUE,
  text_remove = "[()]",
  ...
)
```

Arguments

`words`	Word or text variable to be plotted.
`word_types_embeddings`	Word embeddings from textEmbed for individual words (i.e., decontextualized embeddings).
`x`	Numeric variable that the words should be plotted according to on the x-axes.
`y`	Numeric variable that the words should be plotted according to on the y-axes (y=NULL).
`seed`	Set different seed.
`case_insensitive`	When TRUE all words are made lower case.
`text_remove`	Remove special characters
`...`	Training options from textTrainRegression().

Value

A dataframe with variables (e.g., including trained (out of sample) predictions, frequencies, p-values) for the individual words that is used for the plotting in the textProjectionPlot function.

Examples

```
# Data
# Pre-processing data for plotting
## Not run:
df_for_plotting <- textWordPrediction(
  words = Language_based_assessment_data_8$harmonywords,
  word_types_embeddings = word_embeddings_4$word_types,
  x = Language_based_assessment_data_8$hilstotal
)
df_for_plotting

## End(Not run)
#' @seealso see \code{\link{textProjection}}
```

---

textZeroShot *Zero Shot Classification (Experimental)*

Description

Zero Shot Classification (Experimental)

Usage

```
textZeroShot(
  sequences,
  candidate_labels,
  hypothesis_template = "This example is {}.",
  multi_label = FALSE,
  model = "",
  device = "cpu",
  tokenizer_parallelism = FALSE,
  logging_level = "error",
  return_incorrect_results = FALSE,
  set_seed = 202208L
)
```

Arguments

sequences	(string) The sequence(s) to classify (not that they will be truncated if the model input is too large).
candidate_labels	(string) The set of class labels that is possible in the to classification of each sequence. It may be a single label, a string of comma-separated labels, or a list of labels.
hypothesis_template	(string; optional) The template that is used for turning each of the label into an NLI-style hypothesis. This template must include a "" or similar syntax so that

the candidate label can be inserted into the template. For example, the default template is "This example is ." With the candidate label "sports", this would be fed into the model like "<cls> sequence to classify <sep> This example is sports . <sep>". The default template works well in many cases, but it may be worthwhile to experiment with different templates depending on the task setting (see https://huggingface.co/docs/transformers/).

`multi_label` (boolean; optional) It indicates whether multiple candidate labels can be true. If FALSE, the scores are normalized such that the sum of the label likelihoods for each sequence is 1. If TRUE, the labels are considered independent and probabilities are normalized for each candidate by doing a softmax of the entailment score vs. the contradiction score.

`model` (string) Specify a pre-trained language model that have been fine-tuned on a translation task.

`device` (string) Name of device to use: 'cpu', 'gpu', or 'gpu:k' where k is a specific device number

`tokenizer_parallelism` (boolean) If TRUE this will turn on tokenizer parallelism.

`logging_level` (string) Set the logging level. Options (ordered from less logging to more logging): critical, error, warning, info, debug

`return_incorrect_results` (boolean) Stop returning some incorrectly formatted/structured results. This setting does CANOT evaluate the actual results (whether or not they make sense, exist, etc.). All it does is to ensure the returned results are formatted correctly (e.g., does the question-answering dictionary contain the key "answer", is sentiments from textClassify containing the labels "positive" and "negative").

`set_seed` (Integer) Set seed.

Value

A tibble with the result with the following keys: sequence (string) The imputed sequence. labels (string) The labels sorted in the order of likelihood. scores (numeric) The probabilities for each of the labels.

See Also

see `textClassify`, `textGeneration`, `textNER`, `textSum`, `textQA`, `textTranslate`

Examples

```
# ZeroShot_example <- text::textZeroShot(sequences = c("I play football",
# "The forest is wonderful"),
# candidate_labels = c("sport", "nature", "research"),
# model = "facebook/bart-large-mnli")
```

word_embeddings_4 *Word embeddings for 4 text variables for 40 participants*

Description

The dataset is a shortened version of the data sets of Study 3-5 from Kjell, Kjell, Garcia and Sikström 2018.

Usage

```
word_embeddings_4
```

Format

A list with word embeddings for harmony words, satisfaction words, harmony text, satisfaction text and decontextualized word embeddings. BERT-base embeddings based on mean aggregation of layer 11 and 12.

words words

n word frequency

Dim1:Dim768 Word embeddings dimensions

Source

https://osf.io/preprints/psyarxiv/er6t7/

Index