Package ‘text2emotion’

May 12, 2025

Type Package

Title Emotion Analysis and Emoji Mapping for Text

Version 0.1.0

Description Allows users to analyze text and classify emotions
such as happiness, sadness, anger, fear, and neutrality.
It combines text preprocessing, TF-IDF (Term Frequency-Inverse Document Frequency)
feature extraction, and Random Forest classification to predict emotions
and map them to corresponding emojis for enhanced sentiment visualization.

License GPL-2

Encoding UTF-8

Depends R (>= 4.4.0)

Imports stringr, textclean, magrittr, text2vec, ranger, caret,
parallel, stats, Matrix

RoxygenNote 7.3.2

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

NeedsCompilation no

Config/testthat/edition 3

VignetteBuilder knitr

Author Yusong Zhao [aut],
Fangyi Wang [aut, cre],
Zisheng Qu [aut]

Maintainer Fangyi Wang <123090550@link.cuhk.edu.cn>

Repository CRAN

Date/Publication 2025-05-12 08:20:05 UTC

Contents

`evaluate_rf_model` *Evaluate a Random Forest Model on Test Data*

Description

Evaluate a Random Forest Model on Test Data

Usage

```
evaluate_rf_model(
  rf_model,
  test_texts,
  test_labels,
  tfidf_model,
  vectorizer,
  stopwords,
  verbose = TRUE
)
```

Arguments

`rf_model`	A trained ‘ranger‘ model object.
`test_texts`	A vector of raw test texts.
`test_labels`	A factor vector of true labels.
`tfidf_model`	The TF-IDF transformer used for training.
`vectorizer`	The vectorizer used to build DTM.
`stopwords`	A character vector of stopwords.
`verbose`	Whether to print progress. Default TRUE.

Value

A list with test accuracy, test predictions, and aligned test data.

`handle_negation` *Handle Negation in Token List*

Description

This function processes a character vector of tokens and handles negations by combining the word "not" with the immediately following word (e.g., "not happy" becomes "not_happy"). This technique helps to better preserve sentiment polarity during text analysis.

Usage

```
handle_negation(tokens)
```

Arguments

`tokens`	A character vector of tokens (individual words).

Details

The negation handling procedure follows these steps:

- Iterate through each token.
- If a token is "not" and followed by another token, merge them into a single token separated by an underscore (e.g., "not_happy").
- Skip the next token after merging to avoid duplication.
- Otherwise, keep the token unchanged.

This method is especially useful in sentiment analysis tasks where the presence of negations can invert the sentiment polarity of words.

Value

A character vector of tokens with negations handled by combining "not" with the next word.

Examples

```
handle_negation(c("i", "am", "not", "happy"))
# Returns: c("i", "am", "not_happy")

handle_negation(c("this", "is", "not", "good", "but", "not", "terrible"))
# Returns: c("this", "is", "not_good", "but", "not_terrible")

handle_negation(c("nothing", "to", "worry", "about"))
# Returns: c("nothing", "to", "worry", "about")
```

predict_emotion_with_emoji

Predict Emotion with Emoji Representation

Description

This function takes input text, preprocesses it, extracts TF-IDF features using a pre-trained model, predicts the emotion using a trained classifier, and returns the result with optional emoji representation.

Usage

```
predict_emotion_with_emoji(text, output_type = "textemoji")
```

Arguments

text	Character string containing the text to analyze.
output_type	Type of output to return. Must be one of: • "emotion" - returns only the predicted emotion label • "emoji" - returns only the corresponding emoji • "textemoji" - returns original text appended with predicted emoji (default)

Value

Depending on output_type:

- For "emotion": character string of predicted emotion
- For "emoji": character string of corresponding emoji
- For "textemoji": original text with appended emoji

The function also prints the result to console.

Examples

```
## Not run:
# This example is not run because it requires manually downloading
# a large external model (~30MB), which cannot be retrieved automatically
# and may fail in offline environments.
predict_emotion_with_emoji("I'm so happy today!")
predict_emotion_with_emoji("This makes me angry", "emoji")
predict_emotion_with_emoji("I feel scared", "emotion")

## End(Not run)
```

preprocess_text *Preprocess Text with Slang Handling*

Description

This function performs multi-stage text preprocessing, including lowercasing, HTML cleaning, punctuation normalization, contraction expansion, internet slang replacement, emoticon replacement, and final standardization.

Usage

```
preprocess_text(text, use_textclean = TRUE, custom_slang = NULL)
```

Arguments

`text`	A character vector of input texts.
`use_textclean`	Logical. Whether to use `textclean` for internet slang and emoticon replacement. Default is `TRUE`.
`custom_slang`	A named character vector providing user-defined slang mappings. Optional.

Details

The preprocessing pipeline includes:

- Lowercasing the text.
- Replacing HTML entities and non-ASCII characters.
- Expanding common English contractions (e.g., "I'm" -> "I am").
- Replacing internet slang and emoticons if `use_textclean` is `TRUE`.
- Handling additional slang defined by the user.
- Normalizing repeated punctuations and whitespace.

Value

A character vector of cleaned and normalized text.

Examples

```
preprocess_text("I'm feeling lit rn!!!")
preprocess_text("I can't believe it... lol :)", use_textclean = TRUE)
```

train_full_model *Train a full model pipeline including text preprocessing, TF-IDF vectorization, random forest tuning, and training.*

Description

Train a full model pipeline including text preprocessing, TF-IDF vectorization, random forest tuning, and training.

Arguments

`custom_slang` A named list for custom slang replacements (optional).

`max_features` Maximum number of features for TF-IDF vectorizer (default 10000).

`min_df` Minimum document frequency for TF-IDF (default 2).

`max_df` Maximum document frequency for TF-IDF (default 0.8).

`mtry_grid` Grid of values for ‘mtry‘ parameter to tune in random forest (default: c(5, 10, 20)).

`ntree_grid` Grid of values for ‘ntree‘ parameter to tune in random forest (default: c(100, 200, 300)).

`stopwords_file` Path to the stopwords RDS file (default: "final_stopwords.rds").

`vectorizer_file` Path to save the trained vectorizer (default: "trained_vectorizer.rds").

`tfidf_model_file` Path to save the trained TF-IDF model (default: "trained_tfidf_model.rds").

`rf_model_file` Path to save the trained random forest model (default: "trained_rf_ranger_model.rds").

`train_df_cache_path` Path to cache the training data frame (default: "train_df_cached.rds").

Value

A list containing the trained TF-IDF model, vectorizer, random forest model, and test accuracy.

train_rf_model *Train a Random Forest Model with TF-IDF Features*

Description

Train a Random Forest Model with TF-IDF Features

Usage

```
train_rf_model(
  train_matrix,
  train_labels,
  ntree = 300,
  mtry = NULL,
  seed = 123,
  verbose = TRUE,
  train_df_cache_path = "train_df_cached.rds"
)
```

Arguments

`train_matrix`	A sparse matrix (‘dgCMatrix‘) of training features.
`train_labels`	A factor vector of training labels.
`ntree`	Number of trees. Default 300.
`mtry`	Variables to consider at each split. If NULL, auto-selected.
`seed`	Random seed. Default 123.
`verbose`	Whether to print progress. Default TRUE.
`train_df_cache_path`	Path to cache the train data frame. Default "train_df_cached.rds".

Value

A trained ‘ranger‘ model object.

`train_tfidf_model` *Train a TF-IDF Model (for Training Phase)*

Description

Train a TF-IDF model with customizable tokenization and vocabulary pruning.

Usage

```
train_tfidf_model(
  preprocessed_text,
  max_features = 10000,
  min_df = 2,
  max_df = 0.8
)
```

Arguments

`preprocessed_text`
A character vector containing the preprocessed text.

`max_features` The maximum number of features (terms) to include in the vocabulary. Default is 10000.

`min_df` Minimum document frequency for terms. Default is 2 (terms must appear in at least 2 documents).

`max_df` Maximum document frequency as a proportion of documents. Default is 0.8 (terms must appear in less than 80% of documents).

Details

This function performs the following steps:

1. Tokenizes the preprocessed text into words and removes stopwords. 2. Defines custom stopwords and retains important emotional function words. 3. Creates a vocabulary based on unigrams and trigrams, pruning terms based on document frequency and term count. 4. Builds the TF-IDF sparse matrix for the input text.

Value

A list with the following components:

tfidf_model The trained TF-IDF model object.

vectorizer The vocabulary vectorizer used in training.

tfidf_matrix The TF-IDF sparse matrix representing the text data.

Examples

```
preprocessed_text <- c("I'm feeling so happy today!", "I feel really excited and hopeful!")
result <- train_tfidf_model(preprocessed_text)
result$tfidf_model  # Access the trained TF-IDF model
```

tune_rf_model *Tune Random Forest Model Hyperparameters*

Description

This function performs hyperparameter tuning for a Random Forest model using grid search. It searches over the grid of ‘mtry‘ (number of variables to consider at each split) and ‘ntree‘ (number of trees in the forest) to find the best model based on training accuracy.

Usage

```
tune_rf_model(
  train_matrix,
  train_labels,
  mtry_grid = c(5, 10, 20),
  ntree_grid = c(100, 200, 300),
  seed = 123,
  verbose = TRUE
)
```

Arguments

`train_matrix` A sparse matrix (class ‘dgCMatrix‘) representing the training feature data.

`train_labels` A factor vector representing the training labels.

`mtry_grid` A vector of values to search for the ‘mtry‘ parameter (number of variables to consider at each split). Default is ‘c(5, 10, 20)‘.

`ntree_grid` A vector of values to search for the ‘ntree‘ parameter (number of trees in the forest). Default is ‘c(100, 200, 300)‘.

`seed` A seed value for reproducibility. Default is ‘123‘.

`verbose` A logical indicating whether to print progress information during the grid search. Default is ‘TRUE‘.

Details

The function trains multiple Random Forest models using different combinations of ‘mtry‘ and ‘ntree‘ values, and evaluates their performance based on training accuracy. The hyperparameters that give the highest accuracy are returned as the best parameters. The process uses the ‘ranger‘ package for training the Random Forest model.

Value

A list containing the best hyperparameters (‘mtry‘, ‘ntree‘, and ‘accuracy‘):

- ‘mtry‘: The best number of variables to consider at each split.
- ‘ntree‘: The best number of trees in the forest.
- ‘accuracy‘: The accuracy achieved by the model with the best hyperparameters.

Index