# Package ‘quickSentiment’

February 6, 2026

**Title** A Fast and Flexible Pipeline for Text Classification

**Version** 0.1.0

**Description**
A high-level wrapper that simplifies text classification into three streamlined steps: preprocessing, model training, and prediction.
It unifies the interface for multiple algorithms (including 'glmnet',
'ranger', and 'xgboost') and vectorization methods (Bag-of-Words, Term Frequency-
Inverse Document Frequency (TF-IDF)),
allowing users to go from raw text to a trained sentiment model in two function
calls. The resulting model artifact automatically handles preprocessing for
new datasets in the third step, ensuring consistent prediction pipelines.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** quanteda, stopwords, foreach, stringr, textstem, glmnet,
ranger, xgboost, caret, Matrix, magrittr, doParallel

**VignetteBuilder** knitr

**Suggests** knitr, rmarkdown, spelling

**Language** en-US

**NeedsCompilation** no

**Author** Alabhya Dahal [aut, cre]

**Maintainer** Alabhya Dahal <alabhya.dahal@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-02-06 13:30:02 UTC

## Contents

## BOW_test *Transform New Text into a Document-Feature Matrix*

### Description

This function takes a character vector of new documents and transforms it into a DFM that has the exact same features as a pre-fitted training DFM, ensuring consistency for prediction.

### Usage

```
BOW_test(doc, fit)
```

### Arguments

| | |
|---|---|
| doc | A character vector of new documents to be processed. |
| fit | A fitted BoW object returned by `BOW_train()`. |

### Value

A quanteda `dfm` aligned to the training features.

### Examples

```
train_txt <- c("apple orange banana", "apple apple")
fit <- BOW_train(train_txt, weighting_scheme = "bow")
new_txt <- c("banana pear", "orange apple")
test_dfm <- BOW_test(new_txt, fit)
test_dfm
```

## BOW_train *Train a Bag-of-Words Model*

### Description

Train a Bag-of-Words Model

### Usage

```
BOW_train(doc, weighting_scheme = "bow", ngram_size = 1)
```

### Arguments

`doc` A character vector of documents to be processed.

`weighting_scheme` A string specifying the weighting to apply. Must be one of `"bow"`, `"binary"`, `"tf"`, or `"tfidf"`. Defaults to `"bow"`.

`ngram_size` An integer specifying the maximum n-gram size. For example, ‘ngram_size = 1‘ will create unigrams only; ‘ngram_size = 2‘ will create unigrams and bigrams. Defaults to 1.

### Value

An object of class `"qs_bow_fit"` containing:

- `dfm_template`: a quanteda `dfm` template
- `weighting_scheme`: the weighting used
- `ngram_size`: the n-gram size used

#’

### Examples

```
txt <- c("text one", "text two text")
fit <- BOW_train(txt, weighting_scheme = "bow")
fit$dfm_template
```

## `logit_model` *Train a Regularized Logistic Regression Model using glmnet*

### Description

This function trains a logistic regression model using Lasso regularization via the glmnet package. It uses cross-validation to automatically find the optimal regularization strength (lambda).

### Usage

```
logit_model(train_vectorized, Y, test_vectorized, parallel = FALSE)
```

### Arguments

`train_vectorized` The training feature matrix (e.g., a ‘dfm‘ from quanteda). This should be a sparse matrix.

`Y` The response variable for the training set. Should be a factor for classification.

`test_vectorized` The test feature matrix, which must have the same features as ‘train_vectorized‘.

`parallel` Logical

### Value

A list containing two elements:

| | |
|---|---|
| `pred` | A vector of class predictions for the test set. |
| `model` | The final, trained ‘cv.glmnet‘ model object. |

### Examples

```
# Create dummy vectorized data
train_matrix <- matrix(runif(100), nrow = 10)
test_matrix <- matrix(runif(50), nrow = 5)
y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run model
model_results <- logit_model(train_matrix, y_train, test_matrix)
print(model_results$pred)
```

## `pipeline` *Run a Full Text Classification Pipeline on Preprocessed Text*

### Description

This function takes a data frame with pre-cleaned text and handles the data splitting, vectorization, model training, and evaluation.

### Usage

```
pipeline(
  vect_method,
  model_name,
  df,
  text_column_name,
  sentiment_column_name,
  n_gram = 1,
  parallel = FALSE,
  stratify = TRUE
)
```

### Arguments

`vect_method` A string specifying the vectorization method. Must be one of `"bow"`, `"binary"`, `"tf"`, or `"tfidf"`.

`model_name` A string specifying the model to train. Must be one of `"logit"`, `"rf"`, or `"xgb"`.

`df` The input data frame.

`text_column_name`
The name of the column containing the **preprocessed** text.

`sentiment_column_name`
The name of the column containing the original target labels (e.g., ratings).

`n_gram` The n-gram size to use for BoW/TF-IDF. Defaults to 1.

`parallel` If TRUE, runs model training in parallel. Default FALSE.

`stratify` If TRUE, use stratified split by sentiment. Default TRUE.

### Value

A list containing the trained model object, the DFM template, class levels, and a comprehensive evaluation report.

### Examples

```
df <- data.frame(
  text = c("good product", "excellent", "loved it", "great quality",
           "bad service", "terrible", "hated it", "awful experience",
           "not good", "very bad", "fantastic", "wonderful"),
  y = c("P", "P", "P", "P", "N", "N", "N", "N", "N", "N", "P", "P")
)
# Note: We use a small dataset here for demonstration.
# In real use cases, ensure you have more observations per class.
out <- pipeline("bow", "logit", df, "text", "y")
```

## prediction *Predict Sentiment on New Data Using a Saved Pipeline Artifact*

### Description

This is a generic prediction function that handles different model types and ensures consistent preprocessing and vectorization for new, unseen text.

### Usage

```
prediction(pipeline_object, df, text_column)
```

### Arguments

`pipeline_object`
A list object returned by the main ‘pipeline()‘ function. It must contain the trained model, DFM template, preprocessing function, and n-gram settings.

`df` A data frame containing the new data.

`text_column` A string specifying the column name of the text to predict.

### Value

A vector of class predictions for the new data.

### Examples

```
preds <- prediction(my_artifacts, c("cleaned text one", "cleaned text two"))
```

## pre_process *Preprocess a Vector of Text Documents*

### Description

This function provides a comprehensive and configurable pipeline for cleaning raw text data. It handles a variety of common preprocessing steps including removing URLs and HTML, lowercasing, stopword removal, and lemmatization.

### Usage

```
pre_process(
  doc_vector,
  remove_brackets = TRUE,
  remove_urls = TRUE,
  remove_html = TRUE,
  remove_nums = TRUE,
  remove_emojis_flag = TRUE,
  to_lowercase = TRUE,
  remove_punct = TRUE,
  remove_stop_words = TRUE,
  lemmatize = TRUE
)
```

### Arguments

`doc_vector` A character vector where each element is a document.

`remove_brackets` A logical value indicating whether to remove text in square brackets.

`remove_urls` A logical value indicating whether to remove URLs and email addresses.

`remove_html` A logical value indicating whether to remove HTML tags.

`remove_nums` A logical value indicating whether to remove numbers.

`remove_emojis_flag` A logical value indicating whether to remove common emojis.

`to_lowercase` A logical value indicating whether to convert text to lowercase.

`remove_punct` A logical value indicating whether to remove punctuation.

`remove_stop_words` A logical value indicating whether to remove English stopwords.

`lemmatize` A logical value indicating whether to lemmatize words to their dictionary form.

## Value

A character vector of the cleaned and preprocessed text.

## Examples

```
raw_text <- c(
  "This is a <b>test</b>! Visit https://example.com",
  "Email me at test.user@example.org [important]"
)

# Basic preprocessing with defaults
clean_text <- pre_process(raw_text)
print(clean_text)

# Keep punctuation and stopwords
clean_text_no_stop <- pre_process(
  raw_text,
  remove_stop_words = FALSE,
  remove_punct = FALSE
)
print(clean_text_no_stop)
```

---

# `rf_model` *functions/random_forest_fast.R Train a Random Forest Model using Ranger*

---

## Description

This function trains a Random Forest model using the high-performance ranger package. It handles the necessary conversion from a sparse DFM to a dense matrix and corrects for column name inconsistencies.

## Usage

```
rf_model(train_vectorized, Y, test_vectorized, parallel = FALSE)
```

## Arguments

`train_vectorized`
: The training feature matrix (e.g., a ‘dfm‘ from quanteda).

`Y`
: The response variable for the training set. Should be a factor.

`test_vectorized`
: The test feature matrix, which must have the same features as ‘train_vectorized‘.

`parallel`
: Logical

## Value

A list containing two elements:

| | |
|---|---|
| `pred` | A vector of class predictions for the test set. |
| `model` | The final, trained ‘ranger‘ model object. |

## Examples

```
# Create dummy vectorized data
train_matrix <- matrix(runif(100), nrow = 10)
test_matrix <- matrix(runif(50), nrow = 5)
y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run model
model_results <- rf_model(train_matrix, y_train, test_matrix)
print(model_results$pred)
```

# `xgb_model` *Train a Gradient Boosting Model using XGBoost*

## Description

This function trains a model using the xgboost package. It is highly efficient and natively supports sparse matrices, making it ideal for text data. It automatically handles both binary and multi-class classification problems.

## Usage

```
xgb_model(train_vectorized, Y, test_vectorized, parallel = FALSE)
```

## Arguments

`train_vectorized`
The training feature matrix (e.g., a ‘dfm‘ from quanteda).

`Y` The response variable for the training set. Should be a factor.

`test_vectorized`
The test feature matrix, which must have the same features as ‘train_vectorized‘.

`parallel` Logical

## Value

A list containing two elements:

| | |
|---|---|
| `pred` | A vector of class predictions for the test set. |
| `model` | The final, trained ‘xgb.Booster‘ model object. |

## Examples

```
# Create dummy vectorized data
train_matrix <- matrix(runif(100), nrow = 10)
test_matrix <- matrix(runif(50), nrow = 5)
y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run model
model_results <- xgb_model(train_matrix, y_train, test_matrix)
print(model_results$pred)
```

# Index