Package 'quickSentiment'

November 9, 2026

Title A Fast and Flexible Pipeline for Text Classification

Version 0.3.7

Description A high-level pipeline that simplifies text classification into three streamlined steps: preprocessing, model training, and standardized prediction.
It unifies the interface for multiple algorithms (including 'glmnet', 'ranger', 'xgboost', and 'naivebayes') and memory-efficient sparse matrix vectorization methods (Bag-of-Words, Term Frequency, TF-IDF, and Binary). Users can go from raw text to a fully evaluated sentiment model, complete with ROC-optimized thresholds, in just a few function calls. The resulting model artifact automatically aligns the vocabulary of new datasets during the prediction phase, safely appending predicted classes and probability matrices directly to the user's original dataframe to preserve metadata.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports doParallel, foreach, glmnet, magrittr, Matrix, methods, naivebayes, quanteda, ranger, stopwords, stringr, textstem, xgboost

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 2026-11-08 23:00:02 UTC

Contents

BOW_test 2
BOW_train 3

evaluate_performance 4
logit_model 5
nb_model 6
pipeline 7
plot.quickSentiment_prc 9
plot.quickSentiment_roc 9
predict_sentiment 10
pre_process 10
print.quickSentiment_eval 12
qs_negations 12
rf_model 13
xgb_model 14

Index **16**

`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. Defaults to `"bag_of_words"`. • `"bag_of_words"` (Alias: `"bow"`) - Standard count of words. • `"term_frequency"` (Alias: `"tf"`) - Normalized counts (frequency relative to document length). • `"tfidf"` (Alias: `"tf-idf"`) - Term Frequency-Inverse Document Frequency. • `"binary"` - Presence/Absence (1/0).
`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
```

`evaluate_performance` *Evaluate Model Performance (ROC and Precision-Recall)*

Description

Evaluate Model Performance (ROC and Precision-Recall)

Usage

```
evaluate_performance(predicted_probs, actual_classes, positive_label)
```

Arguments

`predicted_probs`	Numeric vector of predicted probabilities for the positive class.
`actual_classes`	Factor or character vector of the actual true labels.
`positive_label`	Character string. The target class you want to evaluate.

Value

An object of class 'quickSentiment_eval', which is a list containing the following metrics:

`target_class`	Character. The specific positive label used for the evaluation.
`auc_roc`	Numeric. The Area Under the Receiver Operating Characteristic curve.
`best_threshold_roc`	Numeric. The optimal probability threshold that maximizes Youden's J statistic.
`auc_pr`	Numeric. The Area Under the Precision-Recall curve.
`best_threshold_pr`	Numeric. The probability threshold that maximizes the F1-Score.
`accuracy_at_best`	Numeric. The overall accuracy of the model if 'best_threshold_pr' is applied.
`roc`	S3 object containing the ROC curve data and metrics.
`prc`	S3 object containing the Precision-Recall curve data and metrics.
`threshold_summary`	A data frame summarizing Accuracy, Precision, Recall, and F1 at 0.1 threshold increments.

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,
  tune = FALSE,
  weights = NULL
)
```

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
tune	Logical
weights	A numeric vector of observation weights. Default is NULL

Value

A list containing two elements:

pred	A vector of class predictions for the test set.
probs	A matrix of predicted probabilities.
model	The final, trained ‘cv.glmnet‘ model object.
best_lambda	The optimal lambda value found during cross-validation.

Examples

```
## Not run:
# Create dummy vectorized training and test data
train_matrix <- matrix(runif(100), nrow = 10, ncol = 10)
test_matrix <- matrix(runif(50), nrow = 5, ncol = 10)

# Provide column names (vocabulary) required by glmnet
colnames(train_matrix) <- paste0("word", 1:10)
colnames(test_matrix) <- paste0("word", 1:10)

y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run logistic regression model (glmnet)
model_results <- logit_model(train_matrix, y_train, test_matrix)

## End(Not run)
```

---

nb_model *Multinomial Naive Bayes for Text Classification*

Description

Multinomial Naive Bayes for Text Classification

Usage

```
nb_model(
  train_vectorized,
  Y,
  test_vectorized,
  parallel = FALSE,
  tune = FALSE,
  weights = NULL
)
```

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
tune	Logical. If TRUE, tests different Laplace smoothing values.
weights	A numeric vector of observation weights. Default is NULL

Value

A list containing four elements:

pred	A vector of class predictions for the test set.
probs	A matrix of predicted probabilities.
model	The final, trained 'naivebayes' model object.
best_lambda	Placeholder (NULL) for pipeline consistency.

Examples

```
# 1. Create dummy numeric matrices with BOTH row and column names
train_matrix <- matrix(
  as.numeric(sample(0:5, 100, replace = TRUE)),
  nrow = 10, ncol = 10,
  dimnames = list(paste0("doc", 1:10), paste0("word", 1:10))
)

test_matrix <- matrix(
  as.numeric(sample(0:5, 50, replace = TRUE)),
  nrow = 5, ncol = 10,
  dimnames = list(paste0("doc", 1:5), paste0("word", 1:10))
)

# 2. Create dummy target variable
y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# 3. Run model
model_results <- nb_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,
  text_vector,
  sentiment_vector,
  n_gram = 1,
  balance = FALSE,
  tune = FALSE,
```

```
  parallel = FALSE
)
```

Arguments

vect_method	A string specifying the vectorization method. Defaults to "bag_of_words". • "bag_of_words" (Alias: "bow") - Standard count of words. • "term_frequency" (Alias: "tf") - Normalized counts. • "tfidf" (Alias: "tf-idf") - Term Frequency-Inverse Document Frequency. • "binary" - Presence/Absence (1/0).
model_name	A string specifying the model to train. Defaults to "logistic_regression". • "random_forest" (Alias: "rf") • "xgboost" (Alias: "xgb") • "logistic_regression" (Alias: "logit", "glm")
text_vector	A character vector containing the **preprocessed** text.
sentiment_vector	A vector or factor containing the target labels (e.g., ratings).
n_gram	The n-gram size to use for BoW/TF-IDF. Defaults to 1.
balance	Logical. If TRUE, calculates inverse class weights to correct for imbalanced datasets. Defaults to FALSE.
tune	Logical. If TRUE, the pipeline will perform hyperparameter tuning for the selected model. Defaults to FALSE. [NEW]
parallel	If TRUE, runs model training in parallel. Default FALSE.

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")
)


out <- pipeline("bow", "naive_bayes",  text_vector = df$text, sentiment_vector = df$y)
```

`plot.quickSentiment_prc` *Plot Precision-Recall Curve*

Description

Plot Precision-Recall Curve

Usage

```
## S3 method for class 'quickSentiment_prc'
plot(x, ...)
```

Arguments

x	An object of class ‘quickSentiment_prc‘.
...	Additional graphical parameters.

`plot.quickSentiment_roc` *Plot ROC Curve*

Description

Plot ROC Curve

Usage

```
## S3 method for class 'quickSentiment_roc'
plot(x, ...)
```

Arguments

x	An object of class ‘quickSentiment_roc‘.
...	Additional graphical parameters.

predict_sentiment *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

```
predict_sentiment(pipeline_object, text_column, threshold = 0.5)
```

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.
text_column	A character vector of preprocessed documents. Apply pre_process() before prediction, using the same settings as during training.
threshold	Numeric. Optional custom threshold for binary classification. If NULL, uses the optimized threshold from training (if available).

Value

A data frame containing the 'predicted_class' and probability columns.

Examples

```
if (exists("my_artifacts")) {
  dummy_df <- data.frame(text = c("loved it", "hated it"), stringsAsFactors = FALSE)
  preds <- predict_sentiment(my_artifacts, df = dummy_df, text_column = "text")
 }
```

pre_process *Preprocess a Vector of Text Documents (Chunked, With Row-Level Fault Isolation)*

Description

This function cleans raw text data using the same fast, vectorized string and tokenization operations as before, but runs them in chunks. If a chunk processes cleanly (the common case), it stays fully vectorized and fast. If a chunk throws an error (e.g. from a malformed row), only THAT chunk falls back to row-by-row processing so the exact offending row(s) can be isolated, quarantined, and set to NA – without slowing down or risking the rest of the dataset.

Usage

```
pre_process(
  doc_vector,
  remove_brackets = TRUE,
  remove_urls = TRUE,
  remove_html = TRUE,
  remove_nums = FALSE,
  remove_emojis_flag = TRUE,
  to_lowercase = TRUE,
  remove_punct = TRUE,
  remove_stop_words = TRUE,
  custom_stop_words = NULL,
  keep_words = NULL,
  lemmatize = TRUE,
  retain_negations = TRUE,
  chunk_size = 5000
)
```

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.
custom_stop_words	A character vector of additional custom words to remove (e.g., c("rt", "via")). Default is NULL.
keep_words	A character vector of words to protect from deletion (e.g., c("no", "not", "nor")). Default is NULL.
lemmatize	A logical value indicating whether to lemmatize words to their dictionary form.
retain_negations	Logical. If TRUE (the default), automatically protects common negation words.
chunk_size	Integer. Number of documents processed per vectorized batch. Defaults to 5000. Larger chunks are faster on clean data; smaller chunks isolate failures faster when a bad chunk needs to fall back to row-by-row.

Value

A character vector of cleaned text, same length as `doc_vector`. Rows that could not be processed are set to `NA_character_`. Details on any failures are attached as `attr(result, "quarantine")`, a data.frame with `row_index`, `original_text` (truncated to 200 chars), and `error_message`.

Examples

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

# Check for any quarantined rows after a run
attr(clean_text, "quarantine")
```

---

print.quickSentiment_eval *Print quickSentiment Evaluation Results*

Description

Print quickSentiment Evaluation Results

Usage

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

Arguments

`x`	An object of class ‘quickSentiment_eval‘.
`...`	Further arguments passed to or from other methods.

---

qs_negations *Standard Negation Words for Sentiment Analysis*

Description

A character vector of 25 common negation words. These words are automatically protected by the `pre_process` function when `retain_negations = TRUE` to prevent standard stopword lists from destroying sentiment polarity.

Usage

```
qs_negations
```

Format

An object of class `character` of length 25.

`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 natively utilizes sparse matrices (dgCMatrix) to avoid memory exhaustion and utilizes Out-Of-Bag (OOB) error for rapid hyperparameter tuning.

Usage

```
rf_model(
  train_vectorized,
  Y,
  test_vectorized,
  parallel = FALSE,
  tune = FALSE,
  weights = NULL
)
```

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
`tune`	Logical. If TRUE, tunes ‘mtry‘ using native OOB error
`weights`	A numeric vector of observation weights. Default is NULL

Value

A list containing four elements:

`pred`	A vector of class predictions for the test set.
`probs`	A matrix of predicted probabilities.
`model`	The final, trained ‘ranger‘ model object.
`best_lambda`	Placeholder (NULL) for pipeline consistency.

Examples

```
## Not run:
# Create dummy vectorized training and test data
train_matrix <- matrix(runif(100), nrow = 10, ncol = 10)
test_matrix <- matrix(runif(50), nrow = 5, ncol = 10)

# Provide column names (vocabulary) required by ranger
colnames(train_matrix) <- paste0("word", 1:10)
colnames(test_matrix) <- paste0("word", 1:10)

y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run random forest model
model_results <- rf_model(train_matrix, y_train, test_matrix)

## End(Not run)
```

---

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,
  tune = FALSE,
  weights = NULL
)
```

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
tune	Logical
weights	A numeric vector of observation weights. Default is NULL

Value

A list containing four elements:

pred	A vector of class predictions for the test set.
probs	A matrix of predicted probabilities.
model	The final, trained 'xgb.Booster' model object.
best_lambda	Placeholder (NULL) for pipeline consistency.

Examples

```
## Not run:
# Create dummy vectorized training and test data
train_matrix <- matrix(runif(100), nrow = 10, ncol = 10)
test_matrix <- matrix(runif(50), nrow = 5, ncol = 10)

# Provide column names (vocabulary) required by xgboost
colnames(train_matrix) <- paste0("word", 1:10)
colnames(test_matrix) <- paste0("word", 1:10)

y_train <- factor(sample(c("P", "N"), 10, replace = TRUE))

# Run xgboost model
model_results <- xgb_model(train_matrix, y_train, test_matrix)

## End(Not run)
```

Index

∗ **datasets**
qs_negations, 12

BOW_test, 2
BOW_train, 3

evaluate_performance, 4

logit_model, 5

nb_model, 6

pipeline, 7
plot.quickSentiment_prc, 9
plot.quickSentiment_roc, 9
pre_process, 10, *12*
predict_sentiment, 10
print.quickSentiment_eval, 12

qs_negations, 12

rf_model, 13

xgb_model, 14