quickSentiment: A Fast and Flexible Pipeline for Text Classification in R

quickSentiment is an R package designed to streamline the process of text classification. It provides a complete, end-to-end workflow from text cleaning to model training, evaluation, and prediction. Built on a modular architecture, it allows users to easily experiment with different vectorization methods and high-performance machine learning models.

This package is the R counterpart to the Python quick_sentiments library, bringing the same philosophy of simplicity and power to the R ecosystem.

Key Features

- **Modular Pipeline:** A single, powerful pipeline() function to run the entire training and evaluation process.
- **Multiple Vectorization Methods:** Includes support for Bag-of-Words (bow), Term Frequency (tf), and TF-IDF (tfidf), with easy n-gram integration.
- **High-Performance Models:** Comes with built-in support for three powerful and popular classification models:
    - **Logistic Regression** (via glmnet for speed and regularization)
    - **Random Forest** (via the high-performance ranger package)
    - **XGBoost** (via the industry-standard xgboost package)
- **Reproducible Predictions:** The pipeline returns a self-contained artifact object, bundling the trained model and all necessary components to ensure that predictions on new data are consistent and reliable.

Installation

You can install the development version of quickSentiment from GitHub with:

```
# install.packages("devtools")

devtools::install_github("AlabhyaMe/quickSentiment")
```

The Core Workflow: A Three-Step Process

The quickSentiment workflow is designed to be logical and flexible, centered around three key functions that mirror the Python version.

1. pre_process_final(): Cleaning Your Text

Before any machine learning model can understand human language, text data needs to be cleaned and normalized. The pre_process_final() function is your package's robust tool for this crucial step. It takes raw text and applies a series of configurable cleaning transformations.

This step is kept separate from the main pipeline, allowing you to use the same cleaned text for multiple different models or analyses (e.g., classification, topic modeling) in the future.

Usage:

```
library(quickSentiment)

library(readr)

# Load your data

my_data <- read_csv("path/to/your/data.csv")

# Create a new column of cleaned text

my_data$cleaned_text <- pre_process_final(my_data$reviewText)
```

Behind the Hood: This function leverages the power of stringr for efficient regex cleaning, textstem for English lemmatization, and quanteda for high-performance tokenization and stopword removal. It is designed to be a comprehensive, one-stop shop for text cleaning.

2. pipeline(): The Main Engine for Training

This is the core orchestrator of your sentiment analysis. It takes your preprocessed data, intelligently handles the train/test split, applies the chosen text vectorization method, trains a specified machine learning model, and evaluates its performance.

You can easily experiment with different configurations just by changing a string argument.

Usage:

```
# Train a TF-IDF Logistic Regression model with bigrams

pipeline_artifacts <- pipeline(

  df = my_data,

  text_column_name = "cleaned_text",

  sentiment_column_name = "rating", # The original column with ratings or labels

  vect_method = "tfidf",

  model_name = "logit",
  n_gram = 2
)
```

Behind the Hood: The pipeline() function is designed for maximum flexibility. It uses a **"model contract"** system internally. Each model function (logit_model, rf_model, etc.) is required to return a standardized list. This allows the main pipeline to remain generic and easily extensible—to add a new model, you simply create a new function that adheres to the contract. The pipeline also returns a comprehensive list of "artifacts," including the trained model and the vectorization template, which are essential for the next step.

3. prediction(): Scoring New, Unseen Data

Once your pipeline has run, you can use the pipeline_artifacts object it returned to make predictions on new, unseen data. This function is a key part of the streamlined workflow.

Usage:

```
# Create a vector of new, raw text to predict on
new_reviews <- c(
  "This was the best product I have ever purchased!",
  "A complete waste of money, I would not recommend this."
)

# The prediction function uses the artifacts to ensure a consistent workflow
final_predictions <- prediction(
  pipeline_object = pipeline_artifacts,
  new_raw_text_vector = new_reviews
)

print(final_predictions)
```

Behind the Hood: This is where the power of the self-contained artifact comes in. The prediction() function is a generic "dispatcher." It inspects the class of the trained_model inside your pipeline_object (cv.glmnet, ranger, xgb.Booster) and automatically uses the correct syntax and data format required by that specific model. It also handles the crucial step of transforming the new text using the exact same DFM template (dfm_template) and preprocessing logic from the training run. This robust process guarantees that your predictions are consistent and reliable, eliminating the risk of data leakage or transformation mismatches.

Future Development

This package is under active development. Future plans include:

- Adding more vectorization methods, such as pre-trained embeddings (Word2Vec, GloVe).
- Integrating topic modeling capabilities.
- Adding more machine learning models.

License