Building a Text Classifier : NLP Project

Presented By
Rebecca Bayssari
rebeccca.bayssari@student-cs.fr

Presented to
Pierre Colombo
colombo.pierre@gmail.com

Abstract

The goal of this project is to employ conventional Natural Language Processing (NLP) methods to create a text classification model without depending on external APIs. The method uses logistic regression for classification and Term Frequency-Inverse Document Frequency (TF-IDF) for text vectorization.A preprocessing pipeline was implemented to clean and normalize textual data, followed by a hyperparameter optimization process using RandomizedSearchCV.After being evaluated on a validation set, the finished model's accuracy was 83.9%. For the test dataset , predictions were generated for a test dataset and saved in a CSV file.This project highlights that text classification can be effectively achieved machine learning techniques while ensuring complete control over the process.

1 Introduction

A basic task in Natural Language Processing (NLP), text classification finds use in document categorization, sentiment analysis, and spam detection. Despite the great accuracy provided by deep learning models and external APIs, their application may not always be practical because of computing limitations or data privacy issues.This project explores an alternative approach by employing traditional NLP methods and machine learning techniques to develop a text classifier without the need for external APIs.

There are multiple ways to approach text classification. Before settling on the final approach, I experimented with some methods that include fine-tuning pre-trained transformers such as BERT, training neural networks for text classification, or using efficient word embeddings like FastText. Each method has its advantages: neural networks can identify complex patterns in text , transformer-based models typically produce state-of-the-art results, and FastText offers a balance between speed and accuracy by considering word morphology. However, considering that I am working on a CPU, execution time played a significant role in the decision-making process. On the other hand, logistic regression and TF-IDF vectorization, offered a solution that was both interpretable and computationally economical without significantly compromising accuracy.

This work shows that traditional NLP methods can effectively classify text, offering a clear and resource-friendly alternative to complex deep learning models. By testing different approaches and choosing the best one for the situation, this study highlights the trade-offs between accuracy, simplicity, and efficiency in text classification.

2 Solution

2.1 Data Preprocessing

The first step in building the classifier was preprocessing the text data to ensure consistency and reduce noise. This involved converting all our text to lowercase, removing punctuation, and eliminating special characters. These transformations help to make the text more standard and improve the effectiveness of feature extraction techniques.After cleaning the text data, the dataset was divided into training and validation sets, with a portion reserved for validation to evaluate the model's generalization ability before predicting on unseen data.

2.2 Feature Extraction

For text representation, a TF-IDF (Term Frequency-Inverse Document Frequency) approach was applied.TF-IDF is a highly used technique in NLP that converts raw text into numerical features while giving higher importance to words that are frequent in a document but rare in the overall corpus. By using this technique, the classifier can concentrate on the most informative terms instead of frequent stopwords.

Character-level n-gram vectorization was used in

place of word-level vectorization, in order to capture subword patterns—which are especially helpful when working with various word variations.The model can identify similarities between words that have similar prefixes, suffixes, or letter sequences thanks to character n-grams. Various n-gram ranges were explored to determine the best balance between granularity and model performance.

2.3 Model Selection

Several ML models were considered for classification, including logistic regression, support vector machines (SVM), and random forests. Ultimately, logistic regression was chosen due to its simplicity, efficiency, and ability to handle high-dimensional text data effectively.
A pipeline was constructed to integrate both the feature extraction step (TF-IDF vectorization) and the classification step (logistic regression).This pipeline makes sure that model training and text preprocessing go smoothly and consistently.

2.4 Hyperparameter Optimization

To further enhance model performance, hyperparameter tuning was performed using RandomizedSearchCV. This approach allows efficient exploration of a large search space while avoiding exhaustive grid search computations. The key hyperparameters optimized included:

- N-gram range: To determine the most effective character-level patterns.
- Maximum number of features: To control the dimensionality of the TF-IDF matrix and prevent overfitting.
- Regularization strength (C) for logistic regression: To balance model complexity and generalization.

By evaluating different hyperparameter combinations through cross-validation(with 3 folds), the optimal configuration was selected based on accuracy performance.

2.5 Model Evaluation and Testing

The final model was evaluated on the validation set using accuracy metric. Once validated, the model was applied to the test dataset to generate predictions. Each test instance was assigned a label based on the trained classifier, and the results were saved in a csv file.

3 Results and Analysis

3.1 Model Performance and Comparison

First, an alternative model was implemented using word-level TF-IDF instead of character-level n-grams where the validation accuracy reached only 68.6% . Then i tried the model which combined TF-IDF character n-grams with logistic regression; itachieved an accuracy of 83.9% on the validation set and 83.953 % on the test set. . This suggests that character-level features captured more useful information in the given dataset, possibly due to spelling variations, morphological patterns, or short text samples. As an additional classifier, support vector machines (SVM) were tested; nevertheless, their accuracy was even lower than that of logistic regression. Given that SVMs frequently need extensive hyperparameter adjustment and might not function as well with high-dimensional sparse representations like TF-IDF, this result is consistent with expectations.

3.2 Hyperparameter Optimization and Final Model

To optimize model performance, RandomizedSearchCV was employed for hyperparameter tuning. The best-performing model was obtained with the following parameters:

- TF-IDF n-gram range: (1,4)
- Maximum number of features: 14,998
- Regularization strength (C) for logistic regression: 9.837

Despite achieving good results, the training process encountered convergence warnings, indicating that the optimization algorithm did not fully converge within the default number of iterations. This suggests that increasing the `max_iter` parameter or applying feature scaling could further stabilize the model and improve performance. However, since this section of the code required 7 hours to complete 10 iterations, I decided not to increase the number of iterations further. Given that I am working on an older PC, I was concerned that pushing the system further could lead to instability or a potential crash.
Overall, the results demonstrate that a well-optimized TF-IDF + logistic regression model provides a strong, interpretable, and efficient solution for text classification, offering a practical alternative to deep learning approaches.

References

First Reference
Second Reference