BT5153 In-class Kaggle Competition

Nanhai Zhong A0231953E

1. Abstract

The purpose of this competition is to build a binary classification model to predict whether the given text could get large voting numbers or small ones. The metric of performance is accuracy.

In this project, several different combinations of data preprocessing, feature engineering, validation methods, and classification models have been tried. In this paper, different methods in each stage will be elaborated on in advance. Then, the whole process of exploratory will be introduced. In the end, the final model with the best performance will be shown.

2. Exploratory Data Analysis

The original training dataset is imbalanced with 28851 records in class 1 and 19311 records in class 0. (Figure 1)

Some high-frequency words in each class are the same, such as “function” and “use”. But there is still some difference. Also, we could see the same words in different forms, such as “use” and “using”. Therefore, lemmatization could be used in the word-level model. (Figure 2)

The average text length of class 0 is about 75. It is smaller than the average length in class 1, which is about 120.

3. Data Preprocessing

3.1 Data Cleaning

There are four versions of data cleaning used. **Basic cleaning** includes removing HTML, removing noise such as numbers and punctuations, removing URLs, and converting all the characters to lower case. Based on the basic cleaning, **improved cleaning** removes the stopwords and conducts lemmatization to keep all words in the dictionary form, which will only be used in the word-level model. **Easy cleaning** is basic cleaning without removing HTML. **No preprocessing** means using the original corpus directly.

3.2 Oversampling

Because of the imbalance, random oversampling is considered in this project.

4. Feature Engineering

4.1 Simple Vectorization

Use the Tokenizer function, convert the texts to sequences and conduct padding.

4.2 Word2Vector

Train the Word2Vector model on the textual data to get an embedding matrix. In different processes of the exploration, the textual data could be the training dataset after splitting, the original training dataset, and the whole corpus of both the training dataset and the test dataset. It turns out by including more text into this model, the performance of the following predicting improves.

4.3 Bert Transformer

Encode and decode the textual data. Use the tokenization-padding-masking process to

gain the final Bert vector. Same as Word2Vector, the encoding and decoding are based on different scopes of textual data.

5. Validation Methods

Split the original training dataset into a 30% validation dataset and a 70% training dataset to support the evaluation of the model performance. The splitting might be used before or after feature engineering, which depends on the feature engineering strategy.

6. Classification Models

Several models have been tried, including NB, LR, RF, SVM, XGBoost, NN, and CNN. The final performance depends both on the classification models and the feature engineering methods.

7. Whole Process of Exploratory for Best Model

First Try: Use improved cleaning for word-level and basic cleaning for W2V and Bert, split the training datasets before feature engineering, and conduct oversampling. The combination of word-level and CNN can only reach about 0.68 (in the validation set, same below), while the W2V with CNN could reach 0.72 and Bert with NN could reach 0.71. In the following steps, I stop exploring on the word-level model.

Second Try: Because I found sometimes in the first several epochs the loss of validation is lower than the loss of training, which might be the problem of data augmentation. Therefore, I decided to remove the oversampling.

Third Try: The limited accuracy might be because the information remaining is not enough. So, I start to test on easy cleaning and no preprocessing and split the training dataset after feature engineering. The performance of W2V with CNN improves a little.

Fourth and Fifth Try: In order to keep more information after feature engineering, I ignore the data leakage problem and use both training and test texts to do feature engineering. Use no preprocessing strategy on the fourth try and use easy preprocessing on the fifth try. In this case, the performance of LR (chosen as one of the final submissions) with Bert reaches 0.73.

Sixth Try: Try to explore other packages for Bert. Use Simpletransformers and Kashgari. Ironically, the Simpletransformers outperforms all the previous models as a black box, which is chosen as the final model with the best performance of 0.75.

8. Final Model with Simpletransformers

Use no preprocessing strategy without oversampling. Use the ClassificationModel function and choose to use the pre-trained model 'bert-base-uncased'. First, train on the training dataset, tune and evaluate the model with validation datasets. After finding the best parameter (num_train_epochs=2), retrain the model with the whole original training dataset and predict on the test dataset.

9. Some thoughts

Regardless of the final black-box algorithm, there is some step in my exploratory process that cannot be reproduced in the practical application. For example, I use the whole corpus to train the W2V, which is a kind of data leakage. But in reality, if we want to predict something in the future we won't get the information. So, a better way might be to use a pre-trained model and to conduct fine-tune.

Appendix

Figure 1. The number of records in each class

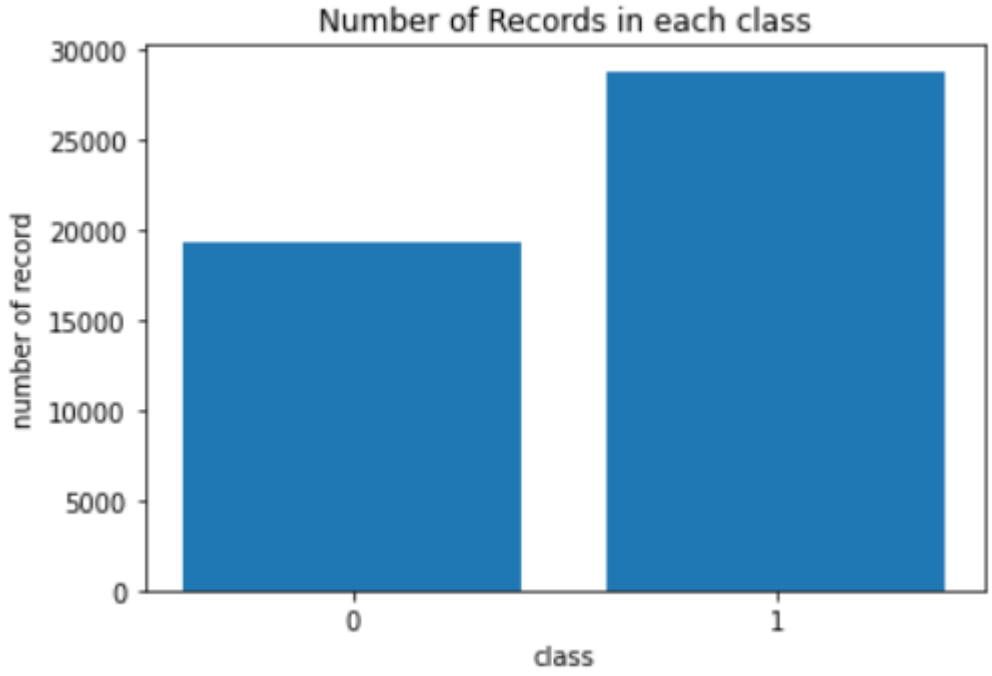

Figure 2. Most frequent words in each class