HW2 report

r12528053 陳映璇

1. Exploratory data analysis (EDA)

Before beginning the classification task, I first analyzed the distribution of the training dataset. Figure 1 illustrates the percentage representation of each emotion category. It is evident that the “joy” category has the highest proportion among all emotion classes.

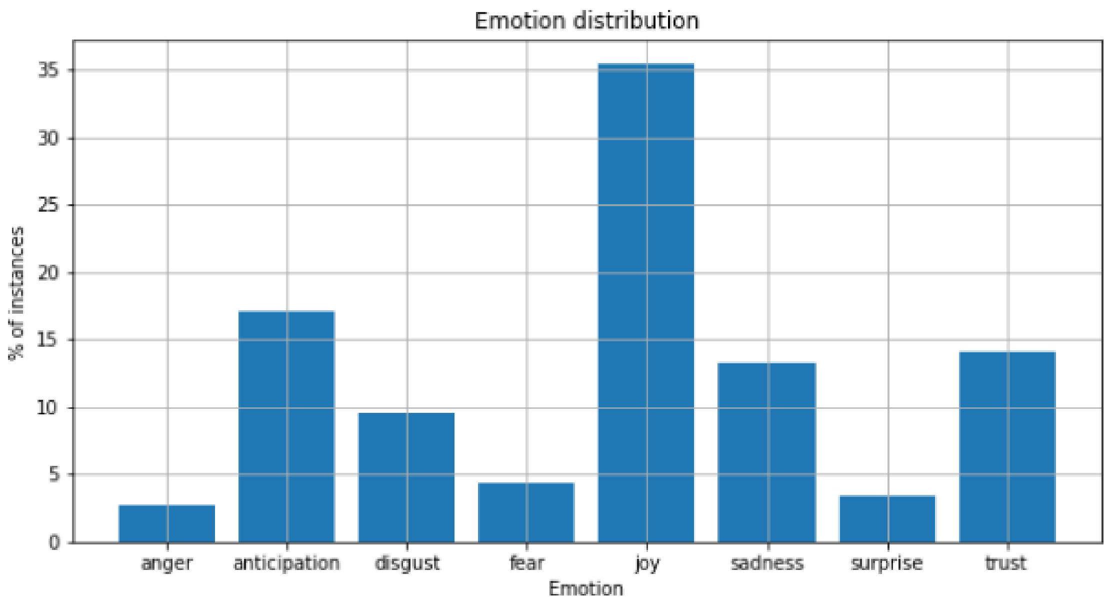

Figure 1 The emotion class distribution of the training dataset

2. Data preprocessing

a. Change the dataset format

In the data preprocessing step, I first converted the three original dataset --emotion.csv, data_identification.csv and tweets_DM.json-- into two Excel files: final_train.csv and test_1.xlsx. The final_train.csv file contains the tweets with their emotion labels. On the other hand, test_1.xlsx contains tweets with their emotion labels and IDs, making it easier to transform into the submission form. The entire process is implemented in the load.py.

b. Data cleaning

Next, I performed a data cleaning process to remove emojis and the string of “<LH>” using re library[1]. Then, I checked whether there were missing values in my dataset. If there were missing values in the training dataset, I dropped the corresponding case. However, for missing values in the testing dataset, I attempted to fill them with their true text. The whole process can be found in the preprocessing.py.

3. Feature engineering

a. Using Bag of Words

I utilized the scikit-learn CountVectorizer[2] with the NLTK tokenizer[3], setting the maximum number of features as 500, to calculate word frequencies. These word frequencies were then used as features to train decision tree, naïve bayes classifier and a simple ANN (Artificial Neural Network) model.

However, the embedding using this bag of word features may not perform well because of the curse of dimensionality and some of the important feature may be ignored using this sparse representation. As a result, I tried the following BERT tokenizer to solve the problem.

b. Using BERT tokenizer

Due to the applied attention mechanism, which operates bidirectionally, BERT[4] effectively addresses the problem of the sparse representations. BERT captures the relationships between words in context by not through the attention mechanism but also through the well-designed pre-train tasks. These tasks include the Masked Language Model (Masked LM) and next sentence prediction (NSP).

In the implementation part, I used the BERTTokenizerFast, specifaclly the fast tokenizer implemented in the Rust library provided by Hugging Face[5]. I retained the default parameters for special tokens, including the separate token ([SEP]), padding token ([PAD]), classification token ([CLS]) and the mask token ([MASK]). The only parameters I customized were the input text data, which consisted of the tweet text we used, and the maximum sequence length, which was set based on the length of our training data. This ensured that the tokenizer efficiently handled the text while maintaining compatibility with the BERT model's input requirements. By using the fast tokenizer, the preprocessing process was significantly accelerated without compromising accuracy or functionality.

4. Classification method

a. Bag Of Words

I applied three classification methods using Bag of Words features for training. The three methods are as follows: decision tree, naïve bayes classifier and ANN.

For the decision tree[6] and naïve bayes classifier[7], I used the built-in functions provided by scikit-learn. As for the ANN model, I constructed the network with three dense layers. The number of neurons in each layer is 64, 64, 8, respectively and the activation functions for the layers are “relu”, “relu” and “sigmoid”. The hyperparameter for the number of epochs was set to 100.

b. Using BERT model

In the classification method, I applied the previously mentioned BERT tokenizer along with the BERT model that is also provided by Hugging Face[5]. A single dense layer with the number of nodes equal to the number of emotion classes, which is eight. Cross entropy was applied as the loss function and the Adam optimizer was employed as the optimizer algorithm. The model was implemented using the PyTorch framework.

5. Result

a. Bag Of Words

Table 1 shows the mean F1 score results for the testing dataset using three different classification methods described in the second and third section. It can be observed that the naïve bayes performs the best among these three methods.

Classifier	Public leaderboard’s mean F1 score	Private leaderboard’s mean F1 score
Decision Tree	0.32182	0.31001
Naïve Bayes	0.40181	0.39054
ANN	0.39828	0.38466

Table 1 Mean F1 score results for the testing dataset

From Figures 2, we can observe that the loss value decreases consistently as the number of epochs increases, indicating that the model is progressively learning and minimizing the error during training. Similarly, the accuracy improves steadily with more epochs, reflecting that the model becomes better at correctly predicting the outputs over time. This trend highlights that the training process is effectively optimizing the model parameters, leading to better performance.

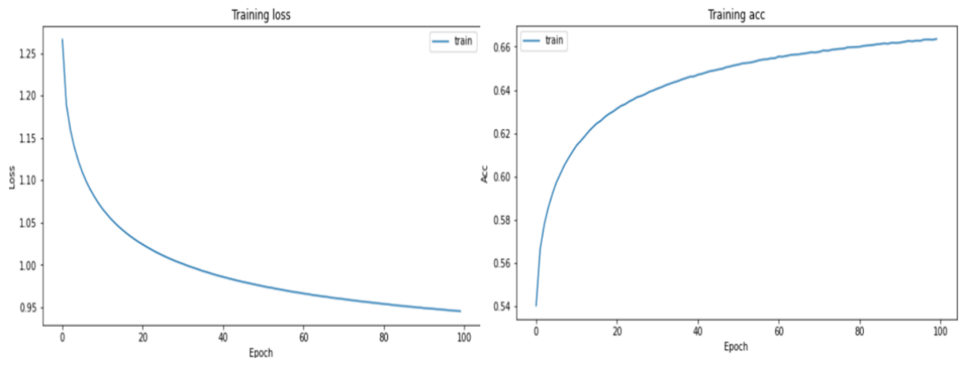

Figure 2 Training loss and accuracy curve

In this bag of word features, I also experimented with increasing the maximum number of features in the CountVectorizer. The best performance was achieved using 100000 features and the naïve bayes classifier, which resulted in a score of 0.42669 on the public leaderboard and 0.439 on the private leaderboard.

b. Using BERT model

The performance of using the BERTTokenizerFast and the BERT model achieved a score of 0.51654 on the public leaderboard and 0.50051 on the private leaderboard. This performance placed the model at rank 27 out of a total of 147 participants.

6. Reference:

[1] "Re library." https://docs.python.org/3/library/re.html (accessed.

[2] "Sklearn CountVectorizer." https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (accessed.

[3] "NLTK Documentation." https://www.nltk.org/api/nltk.tokenize.html (accessed.

[4] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[5] "Hugging Face BERT " https://huggingface.co/docs/transformers/en/model_doc/bert#transformers.BertModel (accessed.

[6] "Decision tree-sklearn " https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html (accessed.

[7] "Naive bayes classifier -Sklearn." https://scikit-

learn.org/1.5/modules/naive_bayes.html (accessed.