A Robustly Optimized BERT Pretraining Approach (RoBERTa) for Textual Data Classification problems

Qu Tianming
Plitecnico di Torino

***Abstract*—In this report we propose a possible method for predict the sentiment of a given content on Twitter by means of classification techniques. The proposed approach shows, based on RoBERTa, the performs better than Linear Support Vector Classification.**

I. Problem overview

The objective of the project was a classification problems of predict the sentiment of the tweet between positive(1) or negative(0) using provided information. The dataset was split into two portions:

- a ***development set***, containing 224994 records, which is characterized by several attributes.
- a ***evaluation set***, containing 74999 records without the target attributes.

We will use the development set to training our classification model and predicting the sentiment of the evaluation set.

The development set was characterized by 5 attributes: **ids, date , flag, user, text**, which represent one tweet's publication date, original poster, and the most important attributes: the text of tweets. Ids is identifier of the tweet, which can not represent the sentiment, also the flags. Even though date and users might have some relationship with sentiment(as example in particular period and particular person may have special sentiment) but for our classification still too much noise.

To address this problem, we may try to consider only features with texts, in the other words, we will remove all other features(e.g., ids, date, etc...), but which can reduce the amount of training data and these features may includes some useful information.

Considering the attributes of the texts, Fig.1 shows the length distribution of texts, in the text dataset, the minimum length is 6, and the maximum is 359, average length is 78, we can also find there are not only the word expressing the sentiment, but also include social interaction like @name, emoticons(e.g., start with &), or Web page reference(e.g., start with http//), or very a few of them are non-interpreted text, characterized by £, $, etc..., Table 1 shows the frequency of feature text.

To solve this problems, we can consider to construct a word filter, which can help us to reduce these noise as much as possible. However, as mentioned above, we have already removed a huge amount of features, filtering noises may cause a model with very poor generalization capabilities.

Intuitively, we can observe that there are nearly 70% of the text characterized by the form of @+name and up to 7% tweeters use emoticons characterized by the form of &, 4.3% of tweets publish a web link, In the end, there a tiny fraction of the text contains non-interpreted words(less than 0.1 %). These attributes might **irrelevant** with sentiment, but they are also full in tweets. At first, we can notice that twitter user's names are appear frequently in the text(more than half), these are the features we have to deal with. On the other hand, considering that the non-interpreted text accounts for a very small proportion of the text, we may ignore them, we will elaborate on this in section II.

TABLE I: The number of text included feature

Feature	Text #	percentage
@	153806	68.4 %
&	14747	6.55 %
http	9851	4.38 %
£	177	0.07 %

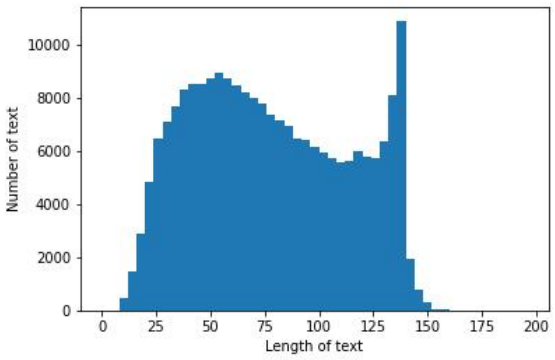

Figure 1. The distribution of text length

II. Proposed approach

A. Data preprocessing

As mentioned above, after we remove the extra features, we should focus on the word filtering. In order to achieve this function, first of all, we design a regular expression to match the character which we refer in previous, then we can eliminate them from our texts and transform them into lowercase.

After filtering, we finally obtain relatively clean texts. For subsequent processing, we decide to choose two methods to match ours different training models:

- We first compute the frequency of occurrence of the remaining word in the text, Figure 2 shows the most fre-

quent words in different sentiment texts (after delete the stop word). Intuitively, we have an interesting discovery: In the positive sentiment, the people are more likely to express their passion directly, this makes the word *"love" "thank"* and *"lol"* become frequently word, while in the other hands, the people who expressing the negative sentiment, they are not usually use negative words, which makes neutral word becomes the most frequently word, but we still can find some words represent negative sentiment like *"sad" "hate"*, but the frequency is far less than the positive words. In the analysis, in order to reduce our size of training data, we also need to remove infrequent words (In this report, we decide to remove the word whose frequency smaller than four.)
After the above operation, we organize them into a vocabulary, and use *TfidfVectorizer* technique to transform ours training text data into a tf-idf(term frequency-inverse document frequency) feature matrix, finishing this step, it is enough for our follow-up

- In the second way, we will use ***RobertaTokenizer*** offered by *transformers* library, in more detail, this technique is based on ***BERT tokenizer***, which most important two parameters are: ***Input IDs*** and ***Attention mask***. The *Input IDs* are often the only required parameters to be passed to the model as input. They are token indices, numerical representations of tokens building the sequences that will be used as input by the model. First of all, the tokenizer will split our text into separate words, which based on **WordPiece** tokenizer, which working principle is roughly as follows:*"we first break words into wordpieces given a trained wordpiece model. Special word boundary symbols are added before training of the model such that the original word sequence can be recovered from the wordpiece sequence without ambiguity. At decoding time, the model first produces a wordpiece sequence, which is then converted into the corresponding word sequence."* [1]. After dividing the texts, These tokens can then be converted into IDs which are understandable by the model. This can be done by directly feeding the sentence to the tokenizer, which leverages the Rust implementation for peak performance, but for our training model, *Input IDs* must be organized into same length, we conducted preliminary research on this: after transformed, the **longest token** is 415, the **shortest token** is 2, and the **average length** is 19, Figure 3 shows the distribution of the token length. Intuitively, we can find that most of the tokens are distribute around 0 to 50, so we can choose the 50 as our maximum length, which can guarantee us to remain the most information, for the sequence which not sufficient, either padding or truncation was used to ensure that all sentences were represented using the same sequence length.
During the processing of *Input IDs*, The *Attention mask* also followed: The attention mask is an optional argument used when batching sequences together. This argument indicates to the model which tokens should be attended to, and which should not, The attention mask is a binary tensor indicating the position of the padded indices so that the model does not attend to them. In the *attention mask*, 1 indicates a value that should be attended to, while 0 indicates a padded value. This attention mask is in the dictionary returned by the tokenizer under the key **"attention_mask"**, for our *Input Ids*, the short sequences were padding with 0, mapping to the *Attention mask*, this position can be seen as "not be attended" that is, "0" in the *Attention mask*.

After completing the above processing, we can make the selection of our training model

Figure 2. The most frequent words in different sentiment

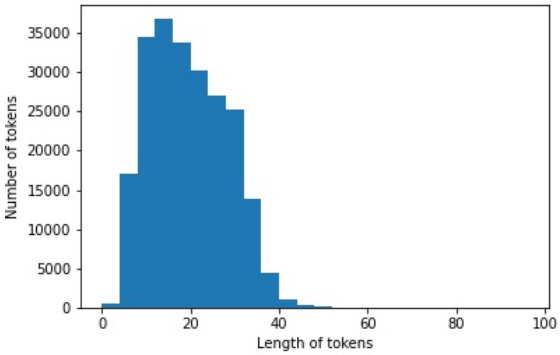

Figure 3. The token number distribution

B. Model selection

We will now discuss more in depth the two models proposed for the classification task:

- **Support Vector machines**(SVM): SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik (1982, 1995) and Chervonenkis (1974). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In our project, we will choose ***LinearSVC*** as training model, it is similar to SVC with parameter kernel='linear', but implemented in terms of

liblinear rather than *libsvm*, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

- **RoBERTa**: which stand for Robustly Optimized BERT Pretraining Approach, it is based on **BERT(Bidirectional Encoder Representations from Transformers)**, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. [2] **RoBERTa** modifies key hyperparameters of **BERT**, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates [3] Figure 4 shows the approximately framework of this model, as the figure following, when we put our texts as tokens and with the attention mask into the training model, as the output, BERT classification model will generate a possibility for which label the text belongs to.

After the selection of training model, in order to maximum the model's performance and matching our computing device, we also need to adjust the model hyperparameters, more details we will discuss in following section.

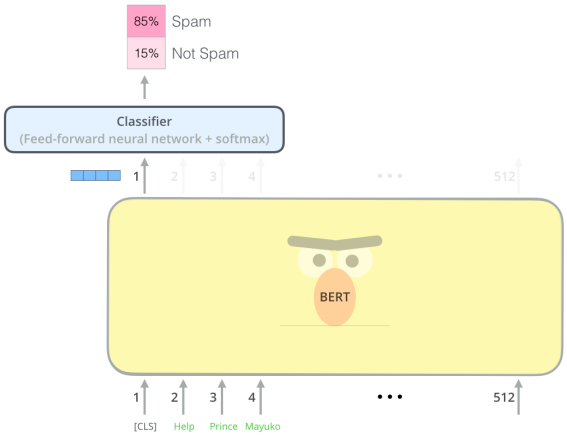

Figure 4. BERT Framework

C. Hyperparameters tuning

First of all, to reduce the affect of overfitting, we will split our development data set. We decide to keep 80% of the development data set as training set, the the remaining 20% as the test set.

For the **LinearSVC** model, there is a very important hyperparameter: Penalty coefficient, C, C Parameter is used for controlling the outliers — low C implies we are allowing more outliers, high C implies we are allowing fewer outliers, we need choose proper C, if C is too large or too small, the generalization ability becomes poor. In the other hand, parameter tol and random_state do not have such influence to our result like C, for these parameter, we build a simple grid search to find the optimize.

TABLE II: Hyperparameters of training model

PARAMETER	TYPE	VALUE
LinearSVC		
C	float	1
tol	float	1e-5
random_state	int	0
max_iter	int	1000
RoBERTa optimizer		
learning_rate	float	1e-5
weight_decay	float	1e-3
OPT		Adam
Device define		
DID	int	0
RoBERTa training model		
EPOCH	int	2
max_length	int	50
batch_size	int	60
seed	int	10

For the **RoBERTa** model, first of all, we define a optimizer offered by *torch* library, *Adam* served as the optimizer algorithm through **gradient descent** method. We also do a grid search for proper learning rate and weight decay. In order to make training model to match our computing device, we define the device and the batch size to ensure the maximum efficiency of our computing equipment(After testing, the efficiency of GPU is much higher than that of CPU).

Table II shows the hyperparameter after we tune.

III. Results

We will compare the result which we obtained of the two model now, with the hyperparameter we discussed in the previous section.

- With the tuning the combination of the parameters for the **LinearSVC** is $\{C = 1, tol = 1e\text{-}5, random\_state = 0, max\_iter = 1000\}$, with this combination we obtain *F1 score* = 0.82 in local test, and *F1 score* = 0.770 on the public evaluation. Which is approximate to the baseline on the leader board. We may consider for deeper search to further optimize this model. At least, since it only consider as a baseline, the performance can roughly meet our requirement.

- For the **RoBERTa**, after two process of training with the hyperparameter which shown in the table II, we will choose the model which has the better performance.After training model process all our data, we have a local test with a F1 score = 0.88 and for the public evaluation, we have a F1 score = 0.871, this score can achieve top 10 for the leader board.

IV. Discussion

The result we obtained represent that the performance of RoBERTa is much better than LinearSVC, even through the parameter tuning and data preprocessing of the former are much more complicated than those of the latter. In fact, the time cost of RoBERTa (around 4 hours for one EPOCH on GTX1070) is much more than LinearSVC (less than 1 minute on cpu), the final result is still satisfactory.

There is a quite interesting fact that when we have a 5 EPOCH of RoBERTa training model, we found that, as the increasing of EPOCH, the accuracy of test set in our model is decreasing(0.85 at the fifth EPOCH), which absolutely the effect of overfitting, that is what we should avoid.

Considering to improve the performance of our model, there are various that might be taken into account:

- First, we might consider more detail of the prepocessing, the feature which we removed could have some useful information like user or date, and we can research more about filtering configuration of the text words, can control the size of the data set.
- Second, for the RoBERTa training model, we can try more kinds optimizer algorithm, except Adam, SGD can also be an option for us, with different optimize algorithm, we can get different performance.
- Last but not least, for our tow models, we can run a more exhaustive grid search on the hyperparameter tune. Moreover, additional classification models might be considered.

In this project, the choice of preprocessing is critical, a suitable preprocessing could have been useful for better performance and control the size of the dataset.

In addition, BERT as the base model of RoBERTa, although it is very large, complicated, and have millions of parameters, we only need to fine-tune it in only 2-4 epochs. That result can be achieved because BERT was trained on the huge amount and already encode a lot of information about our language. An impresive performance achieved in a short amount of time has shown why BERT is one of the most powerful NLP models available at the moment.

References

[1] Wu Y, Schuster M, Chen Z, et al. Google's neural machine translation system: Bridging the gap between human and machine translation[J]. arXiv preprint arXiv:1609.08144, 2016.

[2] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[3] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.