Midterm Project: Disaster Tweets Prediction

Weitong Zhang*

1 Introduction

In this project, we use the Neural Language Processing technique to predict the disaster Tweets to help agencies better monitoring Twitter. We will introduce the dataset, methodology, and further work in this report.

2 Dataset

The training dataset and testing dataset are both provided in the CSV format. In detail, there are 5 columns for the training dataset, which are 'id', 'keyword', 'location', 'text' and 'target'. Since 'id' is a trivial number for each item and some of the data, 'keyword' and 'location' is left blank, our main goal is to use 'text' data to predict whether this tweet is a disaster tweet ('target' is 1) or not ('target' is 0). Column 'target' is not provided in the testing dataset and the output goal is to predict the 'target' for each item with an 'id' number in the testing dataset. There are 7613 items in the training dataset and 3263 items in the testing dataset.

3 Methodology

In this section, we will introduce the methodology for this problem. Our prediction pipeline could be coarsely divided into several parts and we will introduce the detail of each part in the latter subsection.

1. Data Cleaning: Clean the useless data, like repeated punctuation to make the text easier to be classified.
2. Text Encoding: Encode the text into a vector where we can use the vector to train the model.
3. Training: Given the encoded vector, train a model to classify the target.
4. Output: Generate the submission CSV file.

The Data Cleaning and Text Encoding are unsupervised and do not require the target data. Thus these two modules could be applied to both training and test dataset. We only use the training dataset to train the model and use the test dataset to generate the output file.

*Department of Computer Science, University of California, Los Angeles, CA 90095, USA; Email: wt.zhang@ucla.edu UID: 705302329

3.1 Data Cleaning

In this subsection, we discuss the detail of data cleaning. We first convert all text to lower case and remove the HTML or URL tag. This could help reduce the total number of words in the dataset and make it easy to train. Secondly, we provide two further cleaning methods for different encoding methods: For the frequency-based encoding method, like the trivial counter or Tf-idf weighted frequency encoder, we replace all numbers with 'NUMBERS' token and replace the abbreviations with their original words (e.g. replace 'thks' with 'thanks'). This would help the encoder merge the abbreviation with their original words and reduce the difficulty for the training module. As for the pretrained neural-network-based encoding method, like pre-trained BERT network, we do not replace the abbreviation since it is believed that the massive training corpora for the BERT network could automatically do this for us or even more find some minor differences when using the abbreviations instead of original words[1].

3.2 Text Encoding

As aforementioned subsection, the encoding methods used in this project could be generally divided into two parts, the frequency-based method and pre-trained neural-network-based method. We will introduce these methods separately here.

3.2.1 Frequency based encoder

First, we introduce the frequency-based encoder. Generally speaking, these encoders just count the frequency of each word in each sentence without considering the order of the word. `CountVectorizer` provided by sci-kit-learn directly counts the frequency of each word on each sentence. `TfidfVectorizer` is using tf-idf method to normalize the frequency. In detail, the encoder maintains a term-frequency (TF) and inverse-document-frequency (IDF) for each word in each sentence. TF represents the frequency of this word in the current sentence and IDF represents the inverse frequency of this word in the whole document. As a result, the output score of this word is the product of TF and IDF, which represent the relative frequency of that word. Tf-idf score could generate a low weight for the stop words like 'a', 'the'. Therefore it would generate a better encoding feature which is easy to be classified.

3.2.2 Pretrained neural networks

One of the major problems of the frequency-based encoder is that they do not consider the order of words. In recent years, a series of work has proposed several pre-trained network structures for the general language problem. BERT(Devlin et al., 2018) is one of the most powerful ones. The BERT uses a tokenizer to take the sentence as an input and generates the token and its corresponding attention mask. Using these tokens and masks, the BERT model is pre-trained on a massive unlabeled sentence pair. Then the BERT could generate the feature mapping for sentences and the individual words. As shown in Figure 1, taking the sentence pair (A, B), BERT generates the embedding for each word (token) as $T_1, \cdots, T_N$ and the embedding for the whole sentence C. While token embedding $T_1, \cdots, T_N$ could be further utilized in a sequence-to-sequence level application such as Neural Machine Translation (NMT), we can only use the sentence embedding C since we

[1] like abbreviations are more often used in a casual scenario

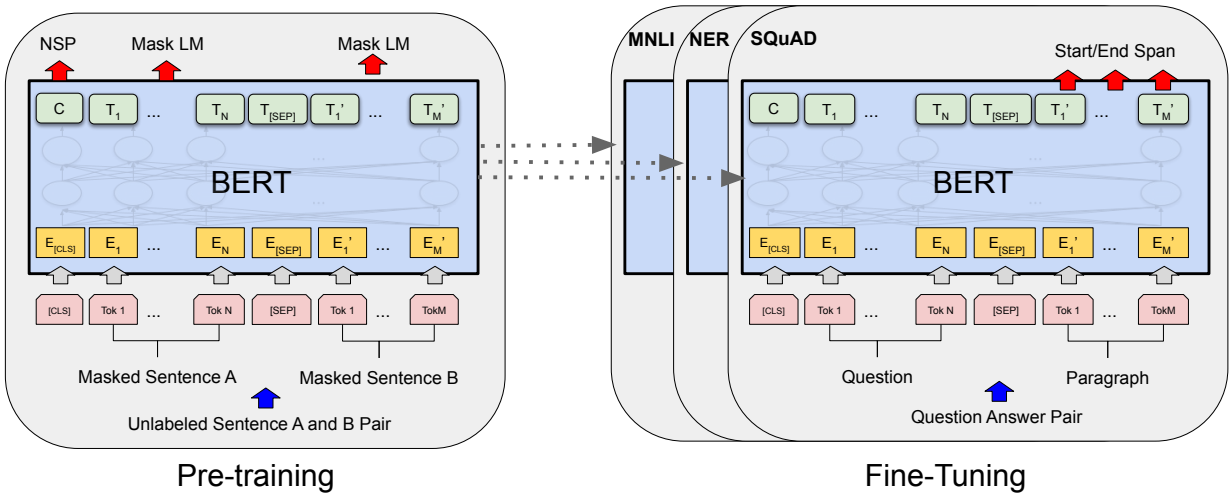

Figure 1: Overall structure of BERT, image credit Devlin et al. (2018)

are classifying the whole sentence. In detail, the base BERT model pre-trained on uncased English corpora provides a 768-dimension embedding vector and a larger BERT model could provide us with a 1024-dimension embedding vector.

3.3 Training

Next, we learn to predict the target given the generated embedding as the input. To simplify the experiments, we only use the learner provided by the sci-kit-learn package, including Logistic Regression Classifier, SVM Classifier, Decision Tree Classifier, and KNN Classifier, and so on. It turns out we can also use the given data to fine-tune the BERT model. However, since this method requires too many computation resources, we remain it as further work for this project.

3.4 Output

The final step is to generate the submission file contains the prediction of the target for each item in the test dataset. Here we retrain the learner using all training datasets instead of split the training dataset into the training and validation parts. This part is trivial and sees the notebook for details.

4 Experiments

We discuss the experiment details in this section: we divided the training dataset into training and validation dataset with a ratio of $7:3$, then we use the training dataset to train and report the performance on the validation set.

5 Conclusion

From the experiment results, we can show that the algorithm can get about 80% accuracy using Logistic Regression on both frequency-based encoding methods and BERT feature mapping. One

Learner \ Encoding	Counter	Tf-Idf	BERT-base	BERT-large
Logistic Regression	0.96 / 0.79	0.89 / 0.79	0.85 / 0.78	0.87 / 0.76
SVM Classifier	0.94 / 0.78	0.97 / 0.80	0.83 / 0.80	0.84 / 0.81
kNN Classifier	0.76 / 0.70	0.84 / 0.77	0.83 / 0.76	0.83 / 0.75
Decision Tree	0.99 / 0.72	0.99 / 0.70	0.98 / 0.67	0.98 / 0.69
Random Forest	0.99 / 0.78	0.99 / 0.78	0.99 / 0.78	0.98 / 0.78

Table 1: Average accuracy for different encoding methods and learners. The accuracy on the left of the slash is the training accuracy, while the accuracy on the right is the validation accuracy.

could find that compared with the frequency-based embedding methods, BERT feature mapping boosts the performance. This is because BERT can consider the order of the words.

Here we also discuss whether using transfer learning to fine-tune the BERT method would help. Since we have shown that using the BERT feature mapping can only increase the performance of the method a little bit, and there is only about 7k data in the training dataset, we can guess that using the BERT model with fine-tuning will help improve the performance.

Because fine-tune BERT model is both a time-consuming and resource-consuming task, we skip this part and leave it as a feature work. We have selected the result with the highest validation accuracy and submitted it to Kaggle. It turns out that our method could reach 80.9% accuracy on the public test dataset.

6 Acknowledgement

The data cleaning part is inspired from the preprocessing part in https://www.kaggle.com/zinebkhanjari/disastertweets#Preprocessing, which includes the translation table (only the JSON file, code excluded) between abbreviations and its corresponding original words, regex replacement for removing the emojis, repeated punctuation, and so on. I, as the author, wrote all other parts of the code, including all of the encoding, training, and output part and the translation logic from abbreviations to the original words. I have moved all the copy-paste code to `clean.py` and all code in the notebook is written by myself only.

7 Code Repository

Code, pre-trained feature vector and prediction results available at https://github.com/ZeroWeight/laughing-guacamole and you might need python 3.9 with `pip3 install -r requirement.txt` to install the dependency package.

References

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .