**NLP Misinformation Detection Project Report**

Jiacheng Ye

904973

**Abstract**

In this report, I will outline my system, the reason behind the choices I have made, the correctness of my approaches, and the performance of the system results for my techniques.

**1 Background**

All codes are run in Colab with GPU runtime environment enabled.

Most of the data used are provided by this subject. There are about 10MB extra data used as negative samples. All extra data are crawled from Kaggle, and they are stored in the same file called “negative\_data.json”. The extra data contains 3 parts. First part is all about non-climate-change fake news, there are about 400 of them. Second part is all about non-climate-change real news, there are about 500 of them. Third part is all about climate-change real news, and there are about 2200 of them. All real news is crawled from “The New York Times”, “Guardians”, “People”, and “Reuters”. I have only chosen these publishers are because, according to some research, they are the most reliable publishers.

In labels, 1 indicates this article is a misinformation article about the climate change. 0 indicates this article is not a misinformation article about the climate change, but it can be a fake news as well.

All models are trained with “train.json” & “negative\_data.json”, and validated with “dev.json”, then tested with “test-unlabelled.json” on Codalab.

All references are listed in my code.

**2 The System**

I changed and improved my system 4 times. I will break it down into 4 sections to demonstrate how my techniques changed. Also, for the errors that all models make (e.g. imbalanced datasets), I will discuss them in detail in Section 3.

**2.0 Results Table**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Logistic Regression with Count/Tf-idf | Keras Model with Bert | MLP Classifier with Tf-idf |
| F1 score (dev) | 0.8 | 0.88 | 0.86 |
| F1 score  (test) | NaN\* | 0.56 | 0.63 |

\*Reasons for no result explained in Section 3

**2.1 Preprocessing**

I standardized all the texts. All texts are lower-cased. Any strings in email format, hyperlink format are replaced with whitespaces. “rt ” and “#” are special tokens in Twitter, and they are replaced with whitespaces. All numbers are removed. All Unicode are removed (e.g., \u2501).

Then I remove all words that contains only 1 character (e.g., “a”, “aa”, “aaa”)

Then I remove all the stopwords provided by NLTK.

**2.2 Logistic Regression, Count/TF-IDF Vectorizer**

**2.2.1 Count Vectorizer**

I first construct a very simple baseline model by using Logistic Regression Classifier with Count Vectorizer. This system uses Bag-Of-Words based on counting words as features only, and then tries to find a linear relationship between the BOWs.

**2.2.2 Analysis for Count Vectorizer**

This base model achieves **0.8** F1 score but performs very poor with the test dataset on Codalab, due to imbalanced dataset. Although it looks fine with the F1 score, by looking at the distributions of features, it is clearly not too good. The features are not separated well enough. This makes sense as, firstly, the dev dataset is very small, it only contains 100 samples; and secondly, neither Logistic Regression nor Count Vectorizer can capture semantic features very well.

手机屏幕截图

描述已自动生成

Feature distribution of Count Vectorizer

**2.2.2** **Analysis for TF-IDF Vectorizer**

It is more reasonable to choose TF-IDF Vectorizer or the Count Vectorizer. As it captures more document-related words. For example, “climate change” will appear both in misinformation articles and the real climate change news articles. Count vectorizer cannot distinguish them, while TF-IDF vectorizer can. This is proved by examining the distribution of TF-IDF Vectorizer. It is certainly more separated.

图片包含 游戏机, 桌子, 飞行, 白色

描述已自动生成

Feature distribution of Tf-idf vectorizer

But the F1 score does not change. This is due to the limitations of Logistic Regression. It cannot capture the relations between the features very well.

**2.3 Logistic Regression, Word2Vec**

As it is still in the process of choosing the best feature selection method, I did not change the classifier.

By introducing Word2Vec, this model is able to have a pretrained word embedding from a much larger corpus. But this is turned out to be a bad decision. The feature distribution is as separated as Tf-idf ’s horizontally, but vertically they are all grouped together. The features for 0 and 1 classes are all mixed together.

**2.3.1 Analysis for Word2Vec**

Word2Vec is context-free. It always gives the same word embedding for each word no matter what the documents are. This is not acceptable for this project. As I mentioned in 2.2.3, the model needs the features which can distinguish between real climate change articles and misinformation climate change articles.

**2.4 Keras Model, Bert**

By using a Bert layer within a Neural Network, it actually solves the issue with using Word2Vec. As the model now has good pre-trained word embeddings, and also will learn and adjust the embeddings based on new documents. As Bert is used with deep neural network models, I choose to use the Keras library.

One special thing about using this model is that, I actually do not need preprocessing. And due to hardware limitations, the maximum length of my sample text has to be set to contain 512 words only. So I kept the stopwords, but all other preprocessing steps are kept. I also re-crawled the same data. But this time I limited each text to only contain 600 words, so I actually was able to contain more data this time.

**2.4.1 Analysis for Bert**

As shown in the result table, the f1 score increases a lot as intended. But due to hardware limitations, max. number of words are set to 512, epochs are set to 5 only, and the batch\_size is set to 4 only. This is a huge downgrade for our model, as Bert essentially needs to be feed with large datasets and run with larger batches.

Another issue with this model is the problem of overfitting. To avoid this, I added 2 Dropout layers. This actually does gives a better performance.

This model tends to have high recall and low precision. By looking into the results, only articles with correct information about climate change will be labelled incorrectly as 1. This may be because of the different distribution between train and validation datasets. This may also because of some features that are always misunderstood and create those False Positives. Sadly, because I was running out of time (more details about this in Section 3), I was unable to go through the dataset and identify those features.

On the other hand, the hyper-parameters I fine-tuned are class weights and the threshold of probabilities.

|  |  |  |  |
| --- | --- | --- | --- |
| 0: 0.6  1: 1.2 | 0: 4  1: 0 | 0: 30  1: 10 | 0: 10  1: 30 |
| 0.83 | 0.8 | 0.88 | 0.83 |

Class weights and corresponding f1 score on dev

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.5 | 0.7 | 0.9 | 0.95 | 0.995 |
| 0.83 | 0.83 | 0.84 | 0.88 | 0.88 |

Thresholds and corresponding f1 score on dev

The class weights are used to change the return value of the loss function used for the model. The more weight a class has, there are more ‘punishment’ when predicting it wrong during training. The first pair of class weight are calculated automatically by a library called ‘sklearn.class\_weight’. The rest are fine-tuned by me. It is clear that I should assign a heavy amount of weights to label 0.

The threshold of probabilities are used to determine if this probability should be classified as 0 or 1. Initially, it is at 0.5. By looking at the results, I found out the model predicts 90% of 1s with probabilities > 0.95. Also, since my model was bias to 1, I increased the threshold to adjust its predictions. Only the predictions with a great certainty will be classified as label 1. As shown in the table, it does help.

On the test dataset however, it performs poorly. The recall is high, but the precision is low. This means that the model still predicts too much 1s. But if I increase the threshold a little higher, the recall will drop. So I stop fine-tuning it, I believe a better dataset matters more at this point.

**2.5 MLP Classifier, TF-IDF Vectorizer**

After some research about what is the alternative solution if Bert is too heavy to use. Then I find the MLP Classifier. Ideally the Bert Model is the best solution, but in this task, MLP with TF-IDF is better.

This is because our dataset is too small. And our hardware is not powerful enough. Besides, we have already known that TF-IDF vectorizer actually does a pretty good job as extracting features to distinguish those 2 classes. A MLP Classifier is also a multi-layer neural network, which means it must out performance the baseline model. And although it performances worse than the Bert model on validation dataset, it performs much better on test dataset.

**3 Error Analysis**

The first 37 of my submission on CodaLab all have F1 score less than 0.25. Some of them are even 1. I spend about 2 weeks to try to find out exactly what happened, and I failed. Until 05/11, I found out that it is because during Pandas reads json to dictionary and then from dictionary to dataframe, it somehow shuffles the input! The input order from “0….1410” to “0,1,10,100,1000,….,999”. This is probably the biggest error ever happened during this project, and it wasted me 2 weeks to try to find out where was wrong in my models. This is why I have no test result for my baseline model and the Word2Vec Model, as the results were unusable.

Another major error makes the models perform not so well is the imbalanced dataset. All of my models tend to predict 1 more than predict 0. Even after I increases the threshold to 0.995. The validation dataset has 100 sample, and the ratio of class 0 and 1 are 50:50. This can cause the model to overfit, as there are too little data to validate on. And the test dataset on CodaLab contains very imbalanced class ratio. Situations like the model happens to predict all the FPs and only a portion of TPs will happen a lot, which leads to a low F1 score.

Another major error can be the difference of data sources between train data and test data. As more than half of the training data are provide externally by me, the distribution of data can be very different with the test data. For example, I have 400 fake news, 500 real news, 2200 real climate news, and rest are misinformation provided by this project. Maybe in the test dataset, those distributions are different, or maybe some of them are not included at all.

Last major error is the texts in the data. A text can contain both misinformation and real information at the same time! There is no way the model can distinguish them without I carefully looking into the texts. Unfortunately, due to the first error, I do not have time to do so.

4 Takeaways

One thing I have learnt is that it is really important to dive into the dataset before start fine-tuning the model. It is important to learn how to extract features from just the raw data. Also, visualization is a great tool to help as well. Although I cannot find any tool to visualize the neural networks efficiently.