NLP Coursework

Tianyu Dai, Xuqi Liu and **Zhongwei Shi**
Department of Computing
Imperial College London
United Kingdom
{td20, xl3020, zs520}@ic.ac.uk

Abstract

This report describes our regression model for Task 1: Predict the score for how funny the edited headline is. Here we present two methods (with/without pre-trained models) to address this task. In approach 1, we utilize a multitude of the state-of-the-art text preprocessing frameworks to exploit the dataset. Our term proposes two pre-trained models: one adapts the Bidirectional LSTM (BiLSTM) in a recurrent neural network, and the other employs the Bidirectional Encoder Representations from Transformers (BERT). In approach 2, we first utilize the word2vec and feature extraction algorithm to learn word associations from a large corpus of text, then experiment various regression models with different hyperparameters to optimizer the test results. In summary, the BERT outperforms the BiLSTM in approach 1, whilst word2vec performs better than the feature extraction algorithm in approach 2.

1 Introduction

This report is organized in the following way: Section 2 and 3 will walk you through the implementation details of two approaches, including the architecture of models and all text preprocessing techniques we used. Section 4 describes our performance obtained for the blind test set. Section 5 concludes with some discussion on the test results.

Our source code is available online on Google Colab via this link.

2 Approach 1

In this section, we describe the details of our pre-trained neural network model. The implementation is built upon the baseline with quite a few modifications. Our term proposes two pre-trained models: one adapts the bidirectional LSTM (BiLSTM) in a recurrent neural network, and the other employs the Bidirectional Encoder Representations from Transformers (BERT).

2.1 BiLSTM

2.1.1 Data Preprocessing

We utilize a multitude of text preprocessing techniques to exploit the dataset. First, as our task is to predict how funny the **edited** headlines are, we change the original sentences by edit words to make the training sensible. Next, we adopt a spaCy tokenizer to break the raw text into words. We choose spaCy because it provides strong flexibility to specify special tokens that need (or not) be segmented by using special rules. To keep the consistency of our vocabulary, we use a WordNetLemmatizer to group together the inflected forms of a word so that they can be analysed as a single token. After that, we remove all stop words (as defined in the nltk package) to extract only useful tokens. Furthermore, as we use the GloVe model to learn token representations, we remove all words containing non-word characters or URLs and lowercase the rest to maximize the coverage of GloVe on the vocabulary.

Our text preprocessing turns out to be quite successful that a coverage rate of 98.2% (9,676 out of 9,857) is obtained, over 45.2% (8,735 out of 12,924) better than the baseline.

Our proposed model is printed in the Appendix A. Compared to the baseline, we employ two more layers in the network: a batch normalization layer and an activation layer. Next, we describe each layer elaborately.

2.1.2 Layers

Embedding Layer The embedding layer is inherited from the baseline. We use the pre-trained GloVe model in this layer to learn an embedding for most of the words in the training dataset (see the coverage rate above). We define it as the

first hidden layer because we can obtain the high-quality distributed vector representations of headlines, getting ready for deep training.

BiLSTM Layer The BiLSTM layer is inherited from the baseline. It is "bidirectional" version of the LSTM, i.e. it combines the forward hidden layer with the backward hidden layer, preserving both history and future context. With broader knowledge of word representation, the BiLSTM has a proven ability to capture the semantics of the sentence.

Batch Normalization Layer After the BiLSTM layer, we push each mini-batch into a batch normalization layer for data standardisation. With the change, the non-linear activation of the BiLSTM is eliminated so that we can stabilize the learning process and dramatically reduce the number of training epochs required to train deep networks.

Linear Layer with ReLU activation Finally, as this is a regression model, we use a fully-connected layer to map the normalized data from hidden state space to tag space. Moreover, as the funnest is defined by a number in the range $[0, 3]$, we apply an additional ReLU activation to produce non-negative real outputs.

2.1.3 Model Configuration

After a comprehensive hyperparameter search, we find that the model trained with an Adam optimizer with the learning rate of 5e-4, the hidden dimension of 50, and the batch size of 128 is more likely to outperform the rest. For this regression task, we choose the mean square error as the loss function and train our model for 20 epochs.

Dealing with over-fitting The baseline reveals a severely overfitted model. Even after epoch 1, the validation loss starts increasing, whilst the training loss keeps decreasing. We try various ways to alleviate this issue:

1. Since the training dataset is quite small, i.e. only 80% of 10,000 headlines is used for training, we merge the funLines dataset to allow the model to train with more data.

2. We add a dropout layer with the probability of 0.2 in the BiLSTM model to introduce the regularization term during the training.

3. We incorporate the idea of early stopping to stop the training when the validation loss starts increasing.

   With the change, we successfully postpone the occurrence of overfitting, enabling the model to converge after training for sufficent number of epochs.

2.2 BERT

2.2.1 Data Preprocessing

For BERT, similarly, we first change the original headlines with the edit words. However, we retain the entire feature space without pre-processing any punctuation, stemming, lemmatisation, stop words, and etc., because we find that sometimes, these text pre-processing techniques may change the meaning of the original headlines, and in practice, word choice is really important to the funniness of the sentence. Instead, as BERT has tremendous vocabulary, we can learn from those word representations.

2.2.2 Bert Tokeniser

We have tried two methods to get token. One is to concatenate original headlines and new headlines (see Figure 1), and the other is to concatenate new headlines and new words (Figure 2).

Figure 1: concatenating original headlines and new headlines

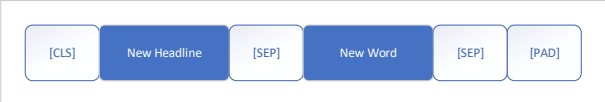

Figure 2: concatenating new headlines and new words

In the first approach, we assume that the degree of funniness depends on the relevance of the new and old sentences, i.e. a standalone headline may not be interesting, but it will be funny when it combines with the other one. However, the second approach relies on the assumption that the edited headline is funny because the edit word plays an important role in the new sentence.

We benchmark both tokenisers and find the former works better. Speaking of the model, we believe the capitalization has minimal effect on the

funniness of the headlines. Therefore, we lowercase all words and utilize the bert-base-uncased pre-trained model.

2.2.3 Optimizer & Learning Rate Schedule

In order to obtain a steady learning process, we train our model with an AdamW optimizer with a linear warmup learning rate schedule. Here we adopts the mean square error as the loss function as well.

This design choice is made because the learning rate in the AdamW optimizer is really sensitive to the final outcome. After a careful hyperparameter search, we finally set it to 3e-5. The use of linear warmup scheduler reduces volatility in the early stages of training. The model can start with small learning rate to stabilize the training. Then, the linearly increasing learning rate can help the model converge faster.

For the choice of epoch, we have tried many times. Although the original paper suggests 2-4 epochs, we find that the model easily starts overfitting after epoch 1. According to the RMSE of the validation set, we select 1 to make the model better.

3 Approach 2

In this section, we introduce an approach without pre-trained representation. We try to address this task from two aspects: first, find the word-embedding; second, use appropriate methods to train these word vectors to solve this regression task. The implementation is developed upon the baseline and can be divided into three steps: data pre-processing, constructing feature vectors, and regression.

3.1 Data Preprocessing

Similar to what we did in the first task, we replace the original words by the edit words. In this approach, to pre-process the data, we remove all the punctuation in the headlines, replace abbreviations, remove all the nonalphabetic symbols using regular expression and also use lowercase except for the initials and proprietary nouns. In the experiments, we also tried stemming and lemmatisation, but they are not bring positive improvement to the results. So, we do not apply them in the final implementation.

Then, to make the processed sentences can be used as the input of the training, we do tokenisation. We use SpaCy tokeniser to remove all the stop words and convert text into word lists. We also combines entities into a single token (e.g. weeks ago) to make them more meaningful.

3.2 Building Word Vectors

To convert the words to the vectors, we do experiments with two types of techniques, the first one is the feature extraction algorithms and the second one is the word2vec algorithm.

3.2.1 Feature Extraction algorithms

To do the feature extraction to the corpus, we first convert all the edit words to a matrix of token frequencies. There are two choices for the input data, the edit words and the edited sentences. There are several ways to extract features. For example, use the token counts to obtain the features representations by using TF-IDF algorithm. Also, decomposition means such as truncated SVD (aka LSA) and sparse PCA can be used to further find the latent features.

Since we do not know which algorithm can find the most useful features, we try each of them and put different features to the regression model to find the suitable one. Also, we combine the different features together use `FeatureUnion` as the input for regression and see if multiple features can bring better results.

After doing experiments, it is shown that when the edit words are the input, use the combined features extracted by TF-IDF and truncated SVD as input to the ridge regression outputs the best fit, which given a training RMSE of 0.41 and a validation RSME loss of 0.57. In the meantime, it is not surprised that this method performs not well on the testing set because it only uses the edit words. So, more attempts are done and described in the following part.

3.2.2 Word2Vec

Apart from finding the latent feature representations of the corpus, obtain the word vectors by neural network training based on the training dataset is also implementable and more common and useful in practice. Word2Vec is a common way of converting the corpus to vectors by learning the word associations using neural network. It has two model architectures which are continuous bag-of-words (CBOW) and continuous skip-gram, and we implement both to find the best suitable one. The word2Vec model we used is provided by the Gensim package, i.e.

`gensim.models.word2vec.Word2Vec()`. It takes the processed sentence tokens as input and output a model with all words and its vectors.

The input dataset can be the sentences with edit words, the edit words and the original words. In order to obtain the word vectors that can make the task result very well, a lot of time is taken on fine-tuning the hyper-parameters, including the vector size, the window size, min_count size, model type (CBOW or skip-gram) and so on. After doing experiments, the best combination of model hyper-parameters is using CBOW model with 20 for vector size, 10 for window size, and 4 for min_count size.

Overall, the word vectors that given the best prediction results are obtained by first train the word2Vec model with all edited and pre-processed headlines, and then concatenate the average vectors of the headlines, the vector of the edited word, and the original word as the feature vectors.

3.3 Regression

After getting the word vectors, we need to do the regression to make the prediction of the score. We try several regression models provided by the scikit-learn package. The `LinearRegression` is just the naive approach of doing linear regression and the `Ridge` model uses L2 regularisation which can better help do multi-variate regression. The parameters of these models are tuned. The parameters that they output the best performance in this task are: for the `LinearRegression`, we set the normalize to be True, and for the `Ridge` regression, we set the alpha value equals to 0.1, the tol to 1 and the normalize as True.

For the final results, using the obtained word vectors described in the previous part as input to the `LinearRegression` model and the `Ridge` model, the previous model has a RMSE error of 0.572 for validation and 0.567 for testing, and the latter model output has a RMSE error of 0.570 for validation and 0.565 for testing. Therefore the Ridge regression performs slightly better than the Linear regression.

4 Performance

In this section, we utilize the blind test set to evaluate the performance of the BERT (approach 1) and word2vec+Ridge Regression (approach 2) model. They are elected because they have better performance than the others. To compare our results with the leaderboard, we also calculate the root mean square error (RMSE) of our model. The test results of both approaches are summarized in Table 1, where RMSE@N means RMSE by taking the $N\%$ most funny headlines and $N\%$ least funny headlines in the test set, for $N \in \{10, 20, 30, 40\}$.

Table 1: performance in two approaches

	RMSE	RMSE@10	RMSE@20	
1	0.530	0.846	0.723	
2	0.565	0.957	0.809	
	RMSE@30	RMSE@40		
1	0.640	0.578		
2	0.705	0.627		

5 Discussion

From Table 1, we can observe that the RMSE of both models decrease with respect to greater N, and the overall RMSE (of which N is maximized) is the lowest. This implies our models have the proven ability to accurately predict the funniness score in almost all general cases. However, when it comes to extreme cases where the edited headline are among those most or least funny set, our model can still classify whether it is funny or not, but fail to give a accurate score. The test results are satisfying - our BERT model ranks 13th on codalab for task 1 at the time of writing this report, and it outperforms the models of other participants in both general and extreme cases.

Nevertheless, we evaluate the BiLSTM model in approach 1 by the test set. The lowest RMSE obtained is 0.58. Though it cannot compete with BERT, it dramatically alleviates the occurrence of overfitting. Recall that our BERT has a severe tendency to overfit, a possible improvement is to combine BERT and BiLSTM together, allowing the model to train for sufficient number of epochs. We can also incorporate the Conditional Random Field (CRF) model for structured prediction.

For approach 2, our experiments prove that using the neural network approach (word2Vec) for obtaining word vectors can perform more robustly than using feature extraction algorithms. It may be because the vectors obtained from training are more case-dependent. Due to the limited time, numbers of regression models have not been fully explored. Actually, we also try to use BiLSTM for regression in approach but no outstanding performance so far in hyper-parameter searching.

6 Appendix

6.1 Colab link

https://colab.research.google.com/drive/1txIcMowqN2jPtxkgarsq-wlRGF8CqysF?usp=sharing

6.2 BiLSTM Structure

```
BiLSTM(
  (embedding): Embedding(9554, 100,
     padding_idx=0)
  (lstm): LSTM(100, 50, dropout=0.2,
     bidirectional=True)
  (bn): BatchNorm1d(100, eps=1e-05,
     momentum=0.1, affine=True,
     track_running_stats=True)
  (hidden2label):
     Linear(in_features=100,
     out_features=1, bias=True)
  (activation): ReLU(inplace=True)
)
```