NLP Coursework: Task 1 - Regression

Bastien Lecoeur bcl20@ic.ac.uk

Sylvie Shi ss9920@ic.ac.uk

Nasma Dasser nd2518@ic.ac.uk

Abstract

In this paper, we compare the performance of the pre-trained representation model BERT with machine learning methods to solve Task 1 (Regression) of the SemEval-2020 Task-7 competition. Using short news headlines, the aim is to predict the mean funniness of an edited version, given the original one. Additionally, we perform extensive analysis on the influence of pre-processing, including stop-word removal, on the final score. We developed a custom stop-word list that can in some cases outperform simpler stop word lists and show that pre-trained BERT with CNN outperforms machine learning methods, even with custom pre-processing. The colab Notebook is available under this LINK.

1 Introduction

In Natural Language Processing, humour detection is a challenging task. The SemEval-2020 Task 7 [4] aims to detect humor in English news headlines from micro-edits. Early humour detection systems are based on traditional machine learning methods like support vector machines, decision trees and Naive Bayes classifiers [1]. Most recently, pre-trained language models based on Transformer [6] have been used to detect humour in tweets and jokes [7]. In this paper, we tackle the task first (Approach 1) using pretrained BERT [3] combined with a Feed Forward Neural Network (FFNN) and with a Convolutional Neural Network (CNN). For Approach 2, we assess the importance of stop-word removal and train the model using XGBoost, Random Forest and Support Vector Regression (SVR).

2 Task Dataset

The Humicroedit dataset [4] contains 9653 training, 2420 development and 3025 testing examples. In Task 1, the goal is to predict the funniness score of an edited headline in the ranges of 0 to 3, where 0 means not funny and 3 means very funny.

The average funniness grade for a headline is 0.94 (Figure 1) and there are on average 12.17 words per sentence (Figure 2). After analysing the sentence structures, we identified the most common stop words (Figure 3) as well as bigrams (Figure 4). Interestingly, Donald Trump and words associated with him ranked the highest in the bigram frequency analysis. These findings, revealed crucial for the pre-processing steps.

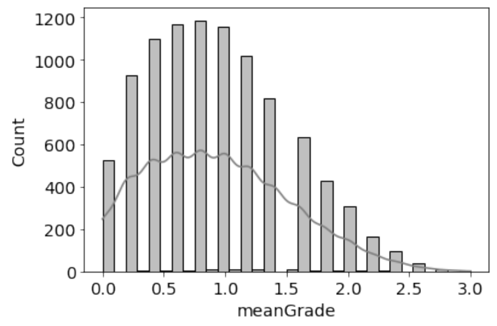

Figure 1: Distribution of the Grades in the Humicroedit dataset [4].

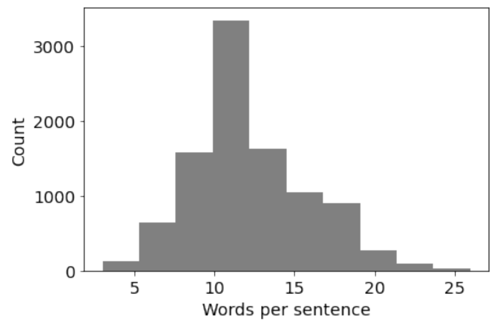

Figure 2: Words per sentence in the in the Humicroedit dataset [4].

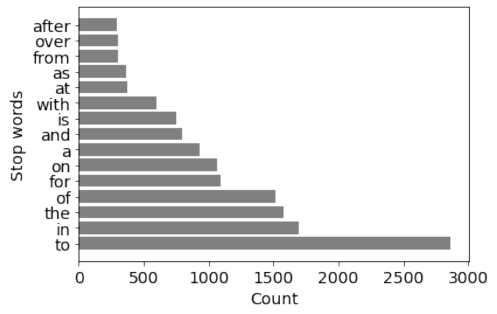

Figure 3: Stop words frequency in the Humicroedit dataset [4].

3 Approach 1

3.1 Method

The main idea in approach 1 is that we want to use a context-sensitive embedding as the first layer in

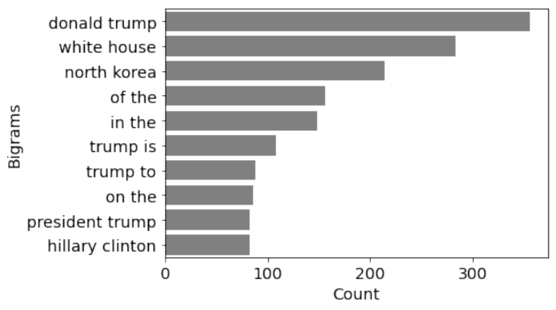

Figure 4: Most words frequent bigrams in the Humicroedit dataset [4].

building a deep learning regression model. Whereas static word embeddings like `Word2Vec` or `GLoVE` give the same meaning to a word in different contexts, a dynamic (contextualized) word embedding trained on a general language model task with a large corpus is able to represent polysemy, capture long-term dependencies in language and help the model learn sentiments. We care about contextualized embeddings because humor is often conveyed through puns, quirky expressions and parodies, which involve using word combinations in unusual ways.

We chose BERT because it is both task-agnostic and deeply bidirectional. As opposed to a bidirectional RNN model, which is not truly bidirectional as states from the two directions do not interact with each other, BERT representations are jointly conditioned on both left and right context in all layers.

For the experiment, we use the PyTorch implementation of the smallest base variant (`'bert-base-uncased'`) of the pre-trained BERT models by HuggingFace [2]. The model was trained on BookCorpus [8] in a self-supervised fashion, meaning that it does not require data labelled by human, with two objectives: masked language modelling (MLM) and next sentence prediction (NSP).

3.2 Preprocessing

We experimented with two procedures for preprocessing for Approach 1. As we are interested in a regression task that gives an absolute score for humor instead of comparing two headlines, we directly replaced the word in </> in the original headline with the word given for micro-edits. In the first preprocessing approach (function `question_sentence_preprocessing` in the notebook), we took the edited headlines, converted them to all lower cases, converted `'t` was converted to `not`, and removed a subset of punctuation (kept question mark) and trailing white spaces. The second preprocessing approach (`full_sentence_preprocessing` in the notebook) is more straightforward, in which we converted all words into lower cases, removed all special characters and trailing white spaces. We used these two preprocessing approaches because the BERT model was trained with special characters, and we wanted to examine if different levels of text preprocessing would make a difference in the performance.

3.3 Model Structures

Two model structures were implemented for Approach 1. The first model is a simple feed-forward neural network that comprises a fixed BERT layer that takes IDs (`bert_id`) and attention masks (`bert_attn`) created with `encoder_plus` for each minibatch, and a 3-layer fully connected network with hidden dimensions=[128, 64, 32] with `LeakyReLU` as activation functions after each layer. The output layer consists of a linear layer that maps 32 neurons to one scalar output, followed by a `ReLU` activation function. As BERT produces a vector embedding of size 768 for each token, after passing minibatches to BERT, we max pool the resulting tensor in order to have one vector representation for each sentence (headline). Details of the FFNN architecture can be found in the notebook in class `FFNN`.

The second model is a convolutional neural network. The model contains the same fixed BERT layer as in the FFNN, followed by two convolutional layers with kernel sizes 5*5 and 3*3 with 128 and 64 feature maps respectively. Results from convolutional layers are then passed through a max pooling layer and a drop-out layer with `drop_prob=0.3`.

3.4 Results

Four Models were trained (an FFNN with preprocessing1 ("question preprocessing"), an FFNN with preprocessing2 ("full preprocessing"), a CNN with preprocessing1 ("question preprocessing") and a CNN with preprocessing2 ("full preprocessing")) with `AdamOptimizer` and a learning rate of $1e-5$. We implemented early stopping during training, in which we halt the training process if the validation loss does not improve after 5 epochs.

As shown in Figures 5,6, 7 and 8, there is no material difference in using the two preprocessing approaches. After experimenting with 4 different models, we chose to use BERT+CNN with the first preprocessing method since it resulted in the best validation RMSE, as shown in Table 1. (Note that we kept more decimal places for validation errors because they were decreasing much slower than the training errors, especially for convolutional neural networks,

Model \ Metric	RMSE (Train)	RMSE (Validation)
BERT+FFNN (preprocessing1)	0.5500	0.5582
BERT+FFNN (preprocessing2)	0.5500	0.5600
BERT+CNN (preprocessing1)	0.5200	0.5507
BERT+CNN (preprocessing2)	0.4900	0.5508

Table 1: Performance Comparison - Approach 1

as shown in Figures 7 and 8). The trained CNN model achieved an RMSE of **0.545** on the final test dataset.

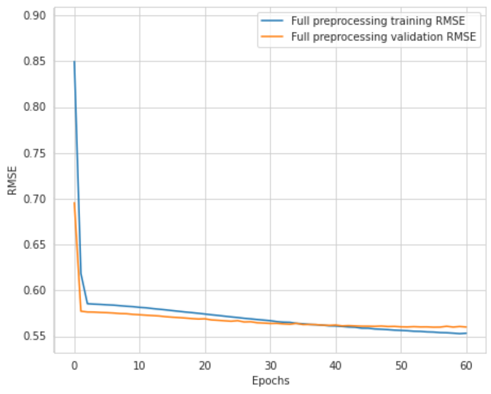

Figure 5: BERT-FFNN with full processing.

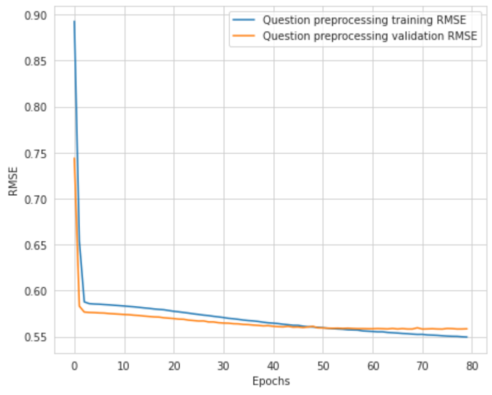

Figure 6: BERT-FFNN with question processing.

4 Approach 2

4.1 Method

For the second approach, we focus first on the preprocessing and second on improving the baseline by implementing three machine learning models: XGboost, Random Forest and SVR and tuning their hyperparameters to obtain the best performance. XGboost is trained with the `xgb` librar. For Random Forest, we use the `scikit-learn` Python library. The

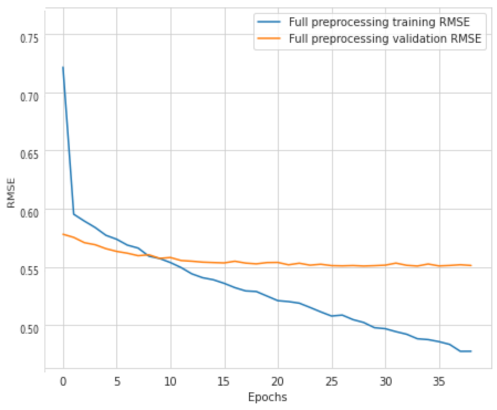

Figure 7: BERT-CNN with full processing.

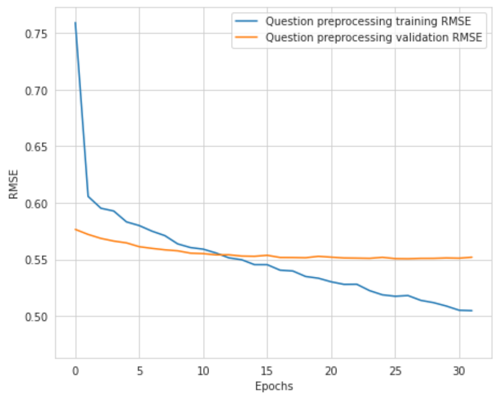

Figure 8: BERT-CNN with question processing.

model is first trained with N=50 estimators and then hyperparameter tuning is done by iterating through a list of estimators between 40 and 100. And finally, we implement SVR with four different kernels (linear, polynomial, radial basis function-rbf and sigmoid) and vary the parameter c between 2 and 40. For each model, we obtain the final RMSE scores, using the different stop word modified datasets and compare the results.

4.2 Preprocessing

After lower casing and removing special characters, we remove stop words by using different models with a multiple stop words lists. The stop word lists were:

- **Empty list**: No stop words.
- **simple_stopwords**: 25 stop words containing determiners and location prepositions.
- **custom_stopwords**: 174 stop words based on the default google list[5].
- **nltk_stopwords**: 179 stop words being the default list of the nltk package.
- **all_stopwords**: 211 stop words containing all the stop words from the previous lists.

We plotted the length of the sentences with each of these methods (Figure 9) and found that `all_stopwords` produced the exact same sentences length as `nltk_stopwords` and therefore decided to remove it from further tests. To represent the words in the models, we then used TF-IDF vectors provided by the `TfidfVectorizer` from the `scitkit-learn` library.

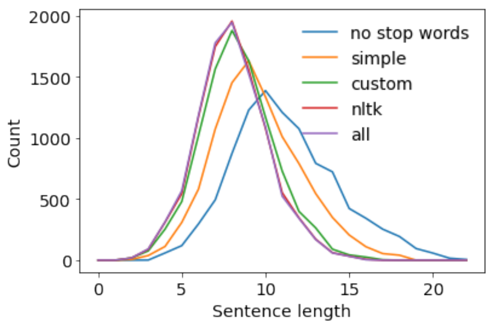

Figure 9: Sentence lengths for each stop words list.

4.3 Results

Results (Table 2), showed that SVR with a polynomial kernel yielded the lowest RMSE score. Interestingly, using the custom stop-word list yielded different results based on the classifier, with Random Forest performing the worst. For Random Forest, we also observed that after N=80 estimators (Figure 10), the RMSE score does not decrease significantly. Using a sigmoid kernel with SVR also dramatically increased the RMSE (Figure 11). Results on the final held-out test set for SVR with poly kernel and no stop word removal resulted in an RMSE of **0.5682**.

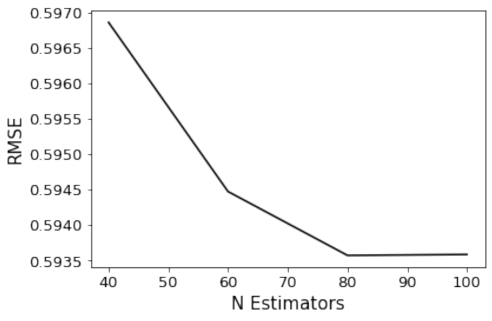

Figure 10: Random Forest Estimators versus RMSE.

	RMSE by Stop words			
	None	Simple	Custom	NLTK
XGBoost	0.5732	0.5737	**0.5726**	0.5730
Random Forest	**0.5960**	0.6042	0.6125	0.6096
SVR (poly)	**0.5712**	0.5730	0.5742	0.5739

Table 2: RMSE scores on the dev dataset. In bold the lowest scores by stop word pre-processing.

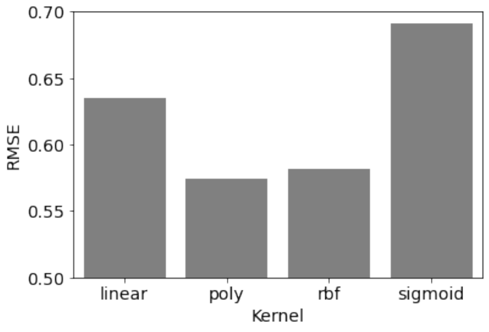

Figure 11: SVR Kernels versus RMSE.

5 Analysis and Discussion

In this project, we explored various machine learning and deep learning algorithms in tackling one of the common challenges in natural language processing: humor (and sarcasm) detection. In approach 1, we saw that a contextualised word embedding pretrained on a general language modelling task was powerful and able to be extracted and applied directly to downstream tasks. We experimented with different learning rates for the optimizer, and came to realize that one needs a relatively small learning rate for this task, as the gradient descent algorithm overshoots easily with a learning rate of above $1e-5$. From training the models multiple times, we also observed that they often struggle to generalize on the validation set. This issue could potentially be mitigated by 1) fine-tune the BERT model with our dataset for this specific task, or 2) combine outputs from different layers of the pretrained BERT model instead of only using the final output as embeddings to improve generalization, 3) investigate the dataset more and introduce appropriate inductive bias to help the model generalize.

In the second approach, we saw that classical methods, can yield satisfying results for small datasets, but require more parameter tuning and careful preprocessing dependent on the task. We also saw the importance of pre-processing when using machine learning methods, as for example stop-word removal which can impact the RMSE score. The best model, SVR with polynomial kernel performed best when using no stop word removal. This can be interpreted as humour requiring "details" and careful sentence structuring to make it more relevant.

References

[1] Santiago Castro et al. “Is this a joke? detecting humor in spanish tweets”. In: *Ibero-American Conference on Artificial Intelligence*. Springer. 2016, pp. 139–150.

[2] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

[3] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[4] Nabil Hossain et al. "Semeval-2020 Task 7: Assessing humor in edited news headlines". In: *arXiv preprint arXiv:2008.00304* (2020).

[5] *Stopwords*. URL: `https://www.ranks.nl/stopwords` (visited on 03/02/2021).

[6] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[7] Orion Weller and Kevin Seppi. "Humor detection: A transformer gets the last laugh". In: *arXiv preprint arXiv:1909.00252* (2019).

[8] Yukun Zhu et al. "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books". In: *arXiv preprint arXiv:1506.06724*. 2015.