NLP Coursework

Rajat Rasal
rrr2417@ic.ac.uk

Rohit Prasad
rp3317@ic.ac.uk

Bargavi C
bc1417@ic.ac.uk

1 Note

Please find the code for the project at: **https://github.com/ML3ngiRNErT/funniness-regression/**. The README in the root dir will explain the purpose of the various files in the repo.

2 Introduction

In this assignment, we aim to explore and evaluate methods to determine how funny a given headline is. The dataset for this task was the basis of the SemEval-2020 Task 7. We divide our experiments into two approaches, the first using pre-trained word embeddings, and the second using feature engineering.

3 Using Pretrained Embedding

3.1 Preprocessing

The main purpose of preprocessing when using deep neural network is to assist the network with learning features its architecture cannot as well as removing noise to make the data closer to its true distribution. Since the dataset is quite small, we focused our efforts on the later as heavily augmenting the dataset would bias the architecture away from the expected test distribution.

Traditionally, for sentiment analysis tasks, all stopwords are removed to only leave words which have some sentimental value (citation). However, due to data sparsity we had to carefully considered which stopwords to remove. We analysed the sentiment of popular stopwords using dictionary-based methods (citation for NLTK) and cosine similarity in embedding layers, and discovered the stopwords have little to no sentimental value. We noticed that predictions were clustered around the mean and had very little separation when we kept stopwords in. Removing all stopwords helped the network with make predictions for mean grades with many data points.

Many pretrained word embeddings are case sensitive; dictionaries include both capitalised and lowercase version of the same word. In the Gensim library's archive of pretrained embeddings, the `word2vec-google-news` and `fasttext-wiki-news` were case-sensitive, whereas `glove-wiki-gigaword` was not. When analysing `fasttext-wiki-news`, we noticed that the both cases of some words were encoded in very different regions of the latent space, for example "France" is similiar to where "france" is closer to . In Fasttext, "Trump" refers to the person but "trump" refers to the verb. We overcame this by using Spacy's named-entity recognition to capitalise named of people (PERSON), government organisations (ORG), geopolitical entities (GPE) and nationalities (NORP). The data distribution of embeddings and the dataset are very different. HuggingFace transformers and Gensim embeddings use data from 2014. HuggingFace NER is trained is on onotext 2013 but challenge data is from 2017.

3.2 Models

3.2.1 LSTM

We focused on developing LSTMs based architectures as these can factor in context into the prediction. BiLSTMs did not improve our regression accuracy as sentences after preprocessing had a length of 10 on average. We attained our best results with a stacked LSTM with 2 layers. Deeper stacking did not improve our results.

Dropout and a shallow layers at the output was also essential to prevent the network from overfitting. When overfitting, the data distribution was being memorised perfectly, causing the validation set RMSE to diverge. Using low-dimensional hidden vectors in the LSTM also

regularised the networks capacity. A dimension of size 10 was sufficient to provide good interpolations on validation data. A Huber loss function, with $\beta = \sigma$, provided better extrapolation to the data distribution's tails.

Fasttest embeddings resulted in the best performance. This might be because they were trained on more recently data, possibly providing more meaningful context for the SemEval dataset. Fasttext is also trained on both Wikipedia and News data, unlike Glove or Word2Vec. The later embeddings resulted in the LSTM becoming an estimator of the mean even though they did not result in overfitting when analysing the RMSE (Fig 7). Fasttext embeddings allowed us to capture the tails in the distribution (Fig 2) resulting in a slightly higher RMSE. Fig 1 also shows us that the this model does struggle to capture the linearity in the true data distribution, but does cpature the upward trend if we consider interquartile ranges. This is supported by the scatterplot.

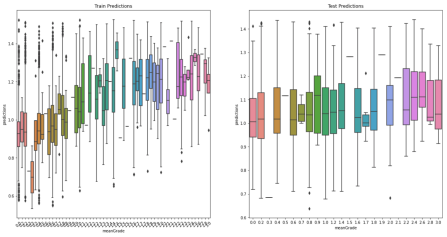

Figure 1: Boxplot for predictions at every point in the meanGrade

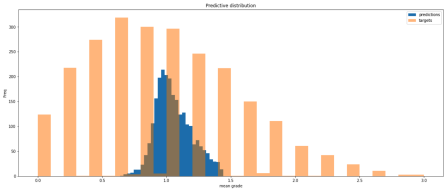

Figure 2: Distribution of predictions

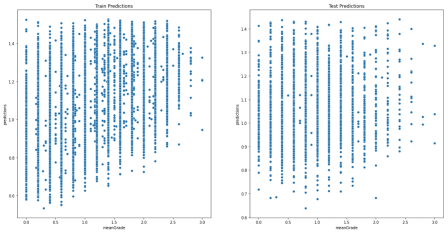

Figure 3: Scatterplot for predictions at every point in the mean grade for training and validation sets

3.2.2 Output constraints

In data sparse cases, heavy data augmentation can result in a model which interpolates well but gives unusual results when extrapolating to heavy-tails of the distribution. We tried to force our model to predict in the outer regions of the distribution with constraints, by constraining the outputs with a sigmoid + learned affine transformation [1] which could push predictions one or two standard deviations from the mean. This is an approach which we should explore more. With the naive implementation of a learned affine transformation, we are manipulating the output in a coarse grained way. A better alternative would have considered more expressive activation functions, such a RELU or Softplus which can better model heavier tails, and used a higher dimensional transform, say a linear layer. ConvNets can also be used for sentiment analysis. However, we noted that punchlines, exclamation or questions in headlines often link the beginning of a sentence to the end. This could have been accounted for by a deeper multiscale convolutional architecture using 1D convolutions of varying kernel size with overlapping pooling.

3.2.3 Transformers

Another approach that we experimented with was using pre-trained transformers. We fine-tuned two pre-trained transformers - BERT and RoBERTa - to our regression problem and evaluated them on our test data. We chose models that were trained on sequence classification tasks and modified the number of classes to be equal to 1 to get a regressor. The models were taken from the Transformers library provided by Hugging Face.

A common issue that we found with these models was that they tend to become tuned to new data very quickly. We chose a learning rate of 3e-5 with no warmup steps for both the transformers to mitigate this issue. The transformers were trained for 2 epochs and evaluated roughly every 50 steps. This ensured that the models learnt slowly and did not overfit to the training data.

Another issue that we encountered was that the training loss and validation RMSE were very noisy. Instead of constantly increasing or decreasing (like with the LSTM models), they fluctuated, making it tricky to analyse whether the model was actually learning anything new. We

[1] Code taken from https://github.com/RajatRasal/deepscm

Model Name	Training Mean RMSE	Dev Mean RMSE	Test Mean RMSE
Stacked LSTM w/ Glove	0.58	0.60	-
Stacked LSTM w/ Fasttext	0.55	0.57	0.58
BERT (`bert-base-uncased`)	0.56	0.59	0.56
RoBERTa (`roberta-base`)	0.50	0.64	0.55
WACCER w/ BERT NER	0.58	0.58	0.58
WACCER w/ Spacy NER	0.58	0.59	0.58

Table 1: Summary of RMSE over 5 runs of training for various models. The LSTMs were trained for 50 epochs with a learning rate of $1e-5$. The final test dataset was downloaded from https://cs.rochester.edu/u/nhossain/humicroedit.html without needing to submit to the competition.

hypothesise that this is because the model did not see all of the data enough times to be able to

There is a difference in the tokenization step for the LSTM and the Transformer models. The LSTM tokenizes the input data by splitting the sentence by the spaces. The BERT tokenizer uses the WordPiece algorithm whereas the RoBERTa tokenizer uses the Byte-Pair Encoding (BPE) algorithm. We did not convert these tokens to word vectors, as the pre-trained models takes care of it for us.

From the table of results above (Table 1), it would seem that BERT performs better than RoBERTa on the `dev` dataset. However, the RMSE alone cannot be a measure of the models' performance. When the distribution of the predictions for both models are compared (Fig A.2 and Fig 13), we can see that RoBERTa actually models the target distribution more closely, whereas BERT only predicts points around the mean of the distribution.

4 No-pretrained embeddings

4.1 WACCER

Next, we took a more unorthodox approach using some of our linguistic intuition with respect to what makes a headline funny. From inspection of some training examples, we identified some overarching patterns. The context of the headlines being 'news' lends itself to many jokes relating to public figures and other *named entities*, often used repeatedly in other training samples. For instance, many samples refer to Trump and his hair, Hillary and her disingenuity or Obama and questionable healthcare policies. We took it upon as a challenge to model these *running jokes*, which we believed had correlation with the grade of the headline.

We broke the implementation down into a few steps:

1. Identify all named entities in headlines
2. Find words that co-occur with each entity and weight by mean grade
3. Cluster embeddings for top k co-occurences for each entity into n clusters
4. Calculate an *entity funniness score* for each headline based on cosine similarity of words with cluster centers
5. Build a regression model from the scores

4.1.1 Named Entity Recognition

Since the model heavily relied on entities, we needed to ensure these were detected reliably. Spacy's NER pipeline relies somewhat heavily on the casing of the input and several thousands of our training samples were capitalised. From this point, we decided to experiment along two separate avenues - using a pre-trained BERT uncased NER model or pre-processing the text further to correct casing. To implement the latter, we used Moses to train a true-casing model on the text that was correctly cased (60%), and ran inference on the incorrectly cased text.

4.1.2 Co-occurrence and Clustering

Next, intuitively we wanted to find the *funniest words that occurred near our entities*. Assuming 'near' meaning 'in the same sentence', we created a map of $(e, w) \rightarrow \texttt{score}$ where

$$\texttt{score} = \sum_{\{y:w,e\in x\}} y^3$$

is the sum of the *cubes* of the grades of the headlines in which both the entity e and co-occurring word w were found. Cubing suppresses less funny entity-word pairs (< 1) and boosts funnier ones. Next, for each entity we take

the top k words sorted by their score and get their GloVe embeddings. Clustering into n groups and projecting into a 2D space using PCA illustrates the result (see figures 4 and 5). The intuition behind clustering (via k-means) is to isolate separate jokes assuming their words' cosine similarities are high; each cluster represents a different joke.

4.1.3 Entity Funniness Score (EFS)

Finally to score a headline, we iterated over the sample's word vectors w and entities e, summing the *maximum* cosine similarities from $\{w \cdot e_1, w \cdot e_2, w \cdot e_3), \ldots\}$ (where e_i refers to the i^{th} cluster center of e), over all e and w. If the headline had no entities, it was given 0. The Pearson correlation coefficient of this score with funniness was 0.142, indicating it had some predictive capabilities. Scoring by negative cosine similarity of the edit and original words (which we would expect to be proportional to funniness assuming more distant replacements are funnier), results in a PCC of 0.155. This indicates EFS captures a non-trivial amount of variance in the mean grade.

4.1.4 Building a regression model

We fed the EFS along with other statistical features into a simple ANN with a scaled sigmoidal output. These included edit position (edits later in the headline tend to follow a setup-punchline pattern), headline length and the negative cosine edit distance as discussed earlier. However, we found the best results using EFS on its own; arriving at our final **Weighted Automatic Clustered Comedy Entity Recognition** model.

5 Conclusions

An overview of RMSE scores can be found in 1.

From the diagrams in the appendix and those above, we can see that all models struggle to extrapolate to the tails of the distribution. Although RoBERTa (Fig 13) does capture the whole distribution, it fails regress over the full support of the training distribution (Fig 14). Further analysis showed us that the vast majority of the predictions of mean grade greater than 2.2 were erroneous, once again captured by the discrepancy between training and validation regression in (Fig 14). This is primarily down to data sparsity at the tails, we in regions of high data we are fairly good predictions across all models.

Figure 4: PCA projection of clustered entity co-occurrences for `donald`

Figure 5: PCA projection of clustered entity co-occurrences for `hillary`

We also noticed that there were many sentences which were only classified correctly by one of the three models. This suggests that a stacking ensemble could be built using the individual weak learners. This would produce a final classification by learning the outputs of the individual weak learners. Such techniques benefit from learners which can optimised on different regions of the data distribution, which is the case here.

Additionally, by building WACCER we showed that a relatively acceptable RMSE, similar to that of the transformer model, can be achieved by just

modelling one facet of the comedy in running jokes. This goes to show that predicting a high variance, subjective value such as funniness to a high accuracy is incredibly difficult from just a few thousand single sentence samples.

Thinking outside of the competition, we believed that language models solely based on a dataset are not the best approach to solving this problem. There is severe lack of contextual information in pretrained representations. By using embeddings trained on more recent data or conditioning the outputs on other relevant latent representations (say knowledge from the body of the news article), we would not need to perform ”hacks” like ensembling or constraining the outputs (sec 3.2.2), and could learn concepts from the natural language directly.

A Appendices

A.1 LSTM w/ Glove

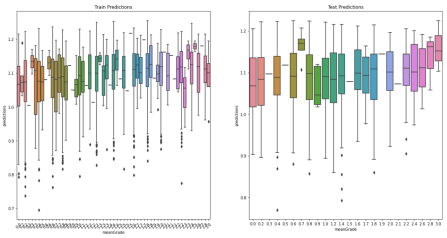

Figure 6: Scatterplot for predictions at every point in the mean grade for training and validation sets

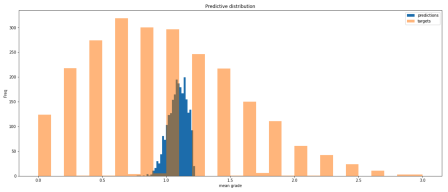

Figure 7: Distribution of predictions

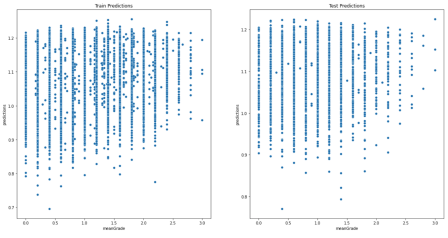

Figure 8: Scatterplot for predictions at every point in the meanGrade

A.2 BERT

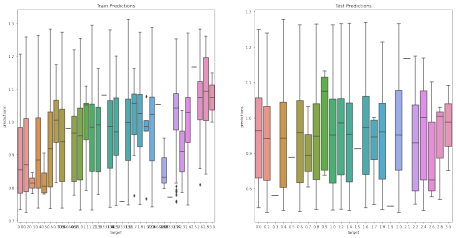

Figure 9: Boxplot for predictions at every point in the meanGrade

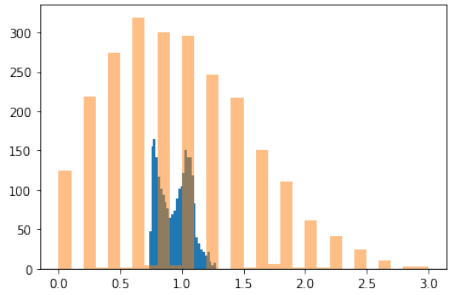

Figure 10: Distribution of predictions

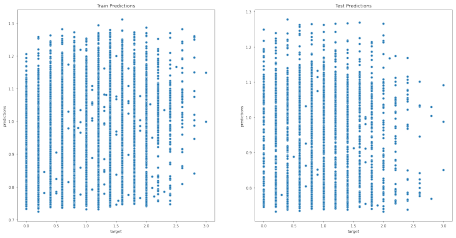

Figure 11: Scatterplot for predictions at every point in the mean grade for training and validation sets

A.3 RoBERTa

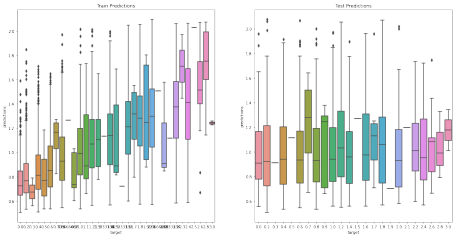

Figure 12: Boxplot for predictions at every point in the meanGrade

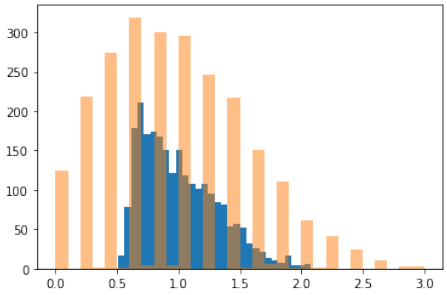

Figure 13: Distribution of predictions

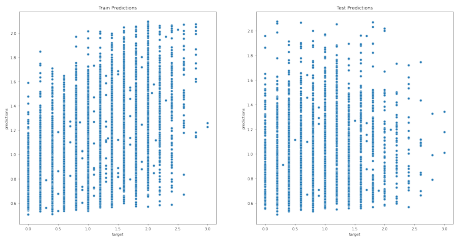

Figure 14: Scatterplot for predictions at every point in the mean grade for training and validation sets

A.4 Word Clouds

Figure 15: LSTM + Fasttext unique word cloud

Figure 16: BERT unique word cloud

Figure 17: RoBERTa unique word cloud

Figure 18: Each learner has a unique subset of sentences which only it can classify. Top to bottom, LSTM w/ Fasttext, BERT, RoBERTa