NLP Coursework: Sentence-level Quality Estimation

James Dorricott
jd3114@ic.ac.uk

Ilyas Saltykov
is916@ic.ac.uk

Sun Wha Whang
sww116@ic.ac.uk

Abstract

This paper presents different approaches undertaken for the sentence-level QE task 2020, where we participated on the English-Chinese language pair task. The proposed models explore various areas of natural language processing, such as contextual word embedding, LSTM and transformer models.

1 Data

We chose to only focus on the English-Chinese dataset as a result of the score distribution having a larger variance than English-German. We hypothesised that the larger variance would mean that the dataset would be more granular, which would in turn enable our models to "learn" more. This turned out to be true since the Pearson Correlation scores achieved for English-Chinese turned out to be higher than for English-German.

2 Initial Models

To begin with, we tried a couple of simple approaches to give us a feel for the data and how simple approaches would perform. This was done in the spirit of "always start simple". As we will see later, these simple approaches turned out to perform surprisingly well.

2.1 Pre-processing

For the 3 initial models we used the same approach to pre-processing. The approach was an extension of the pre-processing code provided in the baseline model. Each sentence was tokenized after which stopwords and words containing non-alphanumeric literals were removed. From the resulting tokenized sentences, we converted them to a list of word embeddings per sentence. Finally, for each sentence, we averaged the word embeddings of that sentence to produce the **mean** word embedding. A notable change from the baseline was how we handled the case where a Chinese character was not found in the embedding dictionary. Instead of replacing it with `0` as in the baseline scalar model, we replaced it with a zeroed-out vector of the same dimension and the embeddings.

Finally, pre-processing resulted in a list of mean word embeddings that correspond to their respective original sentences. We did that for word embeddings of dimension 100 and 300. Later on, we will see the performance of both dimensions. Figure 3 summarizes the libraries and models used.

2.2 Support Vector Regression (SVR)

We extended the baseline model's approach to support multi-dimensional inputs. This allowed us to use the mean word embeddings from the pre-processing as input into SVR. Similarly to the baseline model, the `kernel='rbf'` performed the best (see Figures 1 and 2).

2.3 Random Forest

As in SVR, we extended the baseline's Random Forest approach to support vector inputs. A notable difference is that, in practice, we found that we had to reduce the number of estimators compared to the baseline's scalar approach. This was due to the high-dimensionality of the input, resulting in a **really** long time for the model to train.

2.4 Feed Forward Neural Network

The two previous models being non-neural, we felt that it would be important to try a basic neural network to see how it would perform with the data. The main concern was how well a neural network would learn a mapping for the type of data we were working with.

After experimenting with `PyTorch`, we decided that, for the purpose of an initial model, using `sklearn's MLPRegressor` would do the job.

As we feared, a simple FFNN very quickly started overfitting (the training loss would hit a minimum and start increasing again). As a result, we employed early-stopping which yielded surprisingly good results. Additionally, increasing the number of hidden layers tended to reduce the performance of the model, most likely by enabling the network to learn overly specific features of the training data, hence failing to generalise well. See Figures 1 and 2 for performance.

3 RNN Models

In order to extract more contextual information from each sentence, we experimented with placing an RNN layer between the embedding and the FFNN from Section 2.4. Since an RNN takes each word embedding as its input, the embedding for each sentence was not averaged over words. The models were built with `PyTorch`, and we experimented with both bidirectional LSTM and GRUs.

3.1 Pre-procesing

All pre-processing from Section 2.1 was used, but with some additional steps that follow here.

After inspecting the machine translated Chinese data, we found that a decent proportion of the sentences contained English words. Since we use separate embedding models for the two languages, these English words were identified as unknown from the Chinese embedding, which led to a large information loss. Therefore, we chose to replace these with the English embedding instead. The FastText Chinese embedding, however, did contain existing embeddings for English words as well. Therefore, both replacing these English words with the English FastText embedding and leaving them as they are were experimented with.

FastText does not contain any embedding for numbers, which can have a big impact on prediction. Thus we used the `inflect` library to convert any numbers to its word equivalent e.g. '1' to 'one'.

Two different pairs of embedding models were tested to convert the word tokens into their embedding: Glove 6B for English with Kyubyong for Chinese, and FastText wiki for both English and Chinese. The exact details of these embedding models and the libraries used with are shown in Appendix A. These models were loaded using Gensim.

3.2 Model Architecture

We tested two different architectures, where the first model took the concatenated source and machine translated sentences as a single input. The output features of the last layer of the RNN was then max-pooled over the sequence length. This was then fed into the fully-connected layer to perform regression, which outputs the predicted scores.

The other architecture took the source and machine translated sentences as two separate inputs. These inputs were then passed to two identical RNN layers in parallel. Each output was max-pooled and then concatenated to be fed as a single input to the fully-connected layer.

3.3 Sequence Padding

Since we used batch-training, the sequences in a single batch had to be the same length. Thus we had to pad the sequences. We chose the fixed sequence length to be longer than the longest sentence in the train data. This minimised the loss of information should the model encounter longer sentences at test time, in which case the sentence would be cut to the fixed sequence length. 100 and 50 were chosen as the sequence length for the first and second model, respectively. We also used `pad_packed_sequence()` to inform the RNN which inputs to pad.

4 Transformer Based Models

In this section, we present two different transformer-based QE models, BERT[1]+LSTM and XLM[2]+LSTM. These were built by replacing the pre-trained word embedding in the LSTM model from Section 3 with the pre-trained transformer models. The models were implemented in `PyTorch`, and `HuggingFace` was used to integrate the pre-trained transformers with our model.

We used cased multilingual BERT and XLM pre-trained with Masked Language Modeling (MLM) on 17 languages. Unfreezing different number of transformer layers were experimented with to observe their performance differences.

4.1 Pre-processing

Both BERT and XLM required similar pre-processing. Punctuation and uppercases were kept and no lemmitisation was needed. The sentences were then tokenized into sub-word level using byte pair encoding (BPE). BERT and XLM need different special tokens added to identify the sub-words. BERT needs '##' in front of non-leading sub-words whereas XLM requires '<\w>' added at the end of the final sub-word.

Special tokens were added at the beginning, between and the end of the pair of sentences to concatenate them:

[CLS] src [SEP] mt [SEP]	for BERT
</s>src </s>mt </s>	for XLM

where src and mt are the source and machine translated sentence tokens. Padding tokens [PAD] and <pad> were then added to keep all sequence lengths the same.

`BertTokenizer` and `XLMTokenizer` from `HuggingFace` were used to convert these tokens into their input IDs, which are recognised by the transformers. We also created attention mask and token type IDs to input to the transformers. These tell the transformer location of the separation in the pair of sentences and padding.

4.2 Model Architecture

The input IDs, attention masks and token type IDs generated from Section 4.1 were used as the inputs to BERT and XLM. The last hidden-states of the transformer were then fed into the bidirectional LSTM. The LSTM output was max-pooled along the sequence length and passed to the fully-connected layers to perform regression. ReLU was used as our activation function and dropouts were added between the fully-connected layers to prevent over-fitting. AdamW optimizer was used to train the network with a learning rate scheduler.

5 Pre-extracted Feature Model

While English source sentences did not contain any Chinese characters, we found that Chinese translated sentences contained English words (as described previously), mainly named entities. Therefore, we took the number of identical words in each pair of sentences as our feature.

We also counted the number of common digits in each sentence pair as a feature since different numbers in the pair could have a high correlation to a bad translation.

For RNN-based models, we removed punctuation initially. However, this punctuation could alter the meaning of the sentence and could be an indicator of the finer quality of the translation. Thus we used a list of punctuation from the `string` library and counted the number of occurrences of each in the source and machine translated sentences. The difference in the occurrences were then used as our pre-extracted features.

These extracted features were then fed into a simple FFNN to perform regression. In order to boost the performance of the BERT+LSTM model from Section 4, we tried concatenating these features with the LSTM output before feeding them into the fully-connected layer.

5.1 Ensemble of Models

In order to maximise our performance, we tried ensembling our models discussed in the above sections. We used the score outputs from each model to combine them in a way that would give a higher Pearson. We experimented with linear SVR and a simple FFNN on these scores.

6 Cost Functions

The main evaluation metric is Pearson correlation. However, Pearson correlation itself does not give any information on the accuracy of the individual predicted scores. Therefore, we experimented with combining Pearson correlation and Smooth L1 loss to compute our cost function. The two metrics were combined linearly using β:

$$\text{L} = \beta \times \text{Smooth L1} + (1-\beta) \times (1-\text{Pearson}) \quad (1)$$

Since Pearson's r is in the range $[-1, 1]$, we used $(1-\text{Pearson})$ as the actual value to minimise.

7 Hyper-parameter Tuning

For all neural network based models mentioned above, we performed some hyper-parameter tuning. Although there were many hyper-parameters, we focused on tuning the learning rate, number of epochs and the number of hidden states. Random search was then used on these hyper-parameters once a fairly well performing set of hyper-parameter was found from trial-and-error. We then chose the optimal hyper-parameter set based on the model's performance on the development set, which can be found in Appendix B.

Finally, our production models were trained on both the train and development set with the optimal hyper-parameter to maximise the data.

8 Results

The results of the different model performances on the development set are shown in Table 1. The ensemble model combined BERT+LSTM with Pre-extracted model using linear SVR. The models with * were submitted to CodaLab and their performance on the blind test set can be found in Table 2.

Model	Pearson	MAE	RMSE
Ensemble*	0.4521	0.6339	0.8381
BERT + LSTM*	0.4516	0.6122	0.8516
XLM + LSTM*	0.3593	0.6432	0.8846
SVR (rbf)*	0.3316	0.6495	0.8974
FFNN	0.3039	0.6705	0.8757
LSTM + fastText	0.2983	0.6647	0.8796
GRU + GloVe*	0.2873	0.6829	0.8724
Random Forest	0.2524	0.7093	0.8803
Pre-extracted*	0.1313	0.7051	0.9290

Table 1: Result of the different QE models on the English-Chinese development set. The models with * are our final submissions to CodaLab.

Model	Pearson	MAE	RMSE
Ensemble	0.4603	0.6567	0.8460
BERT + LSTM	0.4592	0.6344	0.8614
XLM + LSTM	0.3811	0.6650	0.8924
SVR	0.3320	0.6670	0.9162
GRU	0.3001	0.6939	0.8887
Pre-extracted	0.2096	0.7024	0.9295

Table 2: Result of the different QE models on the English-Chinese blind test set.

9 Discussion

We found that updating the parameters of the transformer from the beginning lead to catastrophic loss of pre-trained knowledge. This was due to randomly initialised weights of LSTM requiring a much larger update than the transformer, thus distorting the stable transformer weights too much. Instead, the transformer models performed the best when we initially freeze all transformer layers and only unfreeze last 1-2 layers for the final few epochs to boost its final learning. Indeed, BERT+LSTM model in Table 1 was trained with transformer frozen for the first 5 epochs and updated the last layer of the transformer for the last two epochs.

Major challenge with using two different pre-trained embeddings for English and Chinese was that there is no guarantee that these two embeddings have similar mapping. This is why the multilingual transformer models like BERT and XLM performed better as they use global embedding for both languages.

In general, the performance was worse for model ensembles. This may be due to each model being highly correlated as they all rely heavily on contextual embedding. On the other hand, when we combined any neural network based model with the pre-extracted feature based model, the Pearson score increased by a slight amount. This shows that although pre-extracted feature model by itself may perform poorly, it was able to capture meaningful relation that contextual based models could not. This is shown by our Ensemble model performing better than just BERT in Table 2.

Simple models such as SVR were found to outperform RNN-based models. This was because large network like RNN was more prone to overfitting give a small available dataset.

10 Conclusion

As we expected, the most sophisticated of our approaches, the ensemble model (BERT Ensemble in Figure 2), performed the best due to the fact that it combines and utilises tools produced by cutting edge NLP research, with the pre-extracted feature model adding additional performance. Rather surprisingly, a simple statistical approach (SVR) outperformed non-ensembled, sophisticated models such as LSTM or GRU, highlighting the value of such approaches; we believe that the inherent properties of the dataset are likely what caused SVR to perform so well.

11 Code

The code for our QE models can be found on our git repository on `https://gitlab.doc.ic.ac.uk/sww116/nlp-coursework`.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291, 2019.

[3] https://github.com/fxsjy/jieba.

[4] https://github.com/Tony607/Chinese_sentiment_analysis/blob/master/data/chinese_stop_words.txt.

[5] NLPL Word Embeddings Repository. http://vectors.nlpl.eu/repository/20/35.zip.

[6] https://github.com/Kyubyong/wordvectors.

Appendix A: Details of the Pre-processing and Embedding Libraries

Language	**English**		**Chinese**		
Tokenizer	spaCy		jieba [3]		
Stopwords	NLTK		Tony607 [4]		
Pre-trained Embeddings	Glove 6B	fastText	Oslo NLPL word2vec [5]	Kyubyong [6]	fastText
Embedding Dimension	300	300	100	300	300

Table 3: Different libraries used to pre-process the English and Chinese sentences, and the pre-trained embedding models used to convert them into their word embeddings.

Appendix B: Details of Model Hyper-parameters

Model	BERT + LSTM	XLM + LSTM	FFNN	GRU
Optimiser	Adam	Adam	SGD	Adam
Learning Rate	0.0001	0.0001	0.01	0.0001
Scheduler (γ)	0.5	0.5	`sklearn adaptive`	0.7
Scheduler step size (Iteration)	500	500	`sklearn adaptive`	Kyubyong [6]
Batch Size	64	64	32	64
Epoch	7	7	`sklearn` default with early-stopping	6
Maximum Sequence Length	100	300	–	60
RNN Hidden Layer	640	640	–	720
Number of Stacked RNN Layers	1	1	–	2
Hidden Layer	320	64	50	640
Dropout	0.1	0.1	–	0.1
Loss Function	Pearson's r + Smooth L1 (β=0.5)	Pearson's r + Smooth L1 (β=0.5)	`sklearn` default	Smooth L1

Table 4: The final hyperparameters of the models.