Best Answer Prediction on Stack Overflow

Xiang Cao
University of Toronto
UTORid: caoxian2

Ming Hou
University of Toronto
UTORid: houdong

Karthik Mahadevan
University of Toronto
UTORid: mahade20

Abstract

In best answer prediction tasks, the goal is to pick the best answer to a question from a set of candidate answers. In this report, we document our attempts to build an automated best answer predictor on Stack Overflow, a popular community-based question answering website focused on programming. We utilized a dataset with 10 percent of all Stack Overflow questions and answers, representing approximately 1.2 million questions and 2 million answers, to develop a model to select the best answer to a question from a set of possible answers. Through data preprocessing, a subset of the data was used to train several machine learning models using numerical and text features. The best performing models were the Multi Layer Perceptron (MLP) and Long Short-Term Memory (LSTM), achieving a per-question accuracy of around 40%, compared to a baseline performance of 27%.

1 Introduction

In community-based question answering (cQA) users post questions on a website to be answered by other users in a crowdsourced fashion [4]. cQA websites such as Stack Overflow have recently gained popularity, becoming a source for developers to quickly troubleshoot issues they face while programming. Despite their popularity and utility, they pose several challenges. For instance, when a user asks a question on Stack Overflow and receives a satisfactory answer, they can "accept" it, helping future users quickly identify the best answer to a question. However, in many cases, users posting the question forget to accept a best answer. Stack Overflow also allows users to upvote or downvote answers to a question, yet the website is filled with questions to which there are several answers without a score. In this work, we attempted to automatically identify the best answer from a set of answers to a question, by training a plethora of machine learning models on numerical and text-based features of question-answer pairs, finding that they can identify the best answer to a question with an accuracy of around 40%. The dataset [1] and code [2] can be found below.

2 Related Work

Community-based question answering (cQA) has received much attention from the research community, particularly in the context of automation. Problems such as tag prediction [16], duplicate question detection [18] and answer quality estimation [8] have recently been explored. Here, we focus on best answer prediction on Stack Overflow, which has also received treatment in the literature. Lin et al. [9] exploited shallow features (e.g. answer length) and simple semantic features (e.g. question-answer pairs) to train logistic regression and random forest models for classification, focusing primarily on datasets of questions that share a primary tag (e.g. Python). Calefato et al. [2] used linguistic, vocabulary, meta, and thread features, some of which are numerical and others which are textual, to train simple models (such as random forests) in a classification setting on a content dump from legacy developer community forums, achieving high accuracy. Recently, deep learning

[1] `https://www.kaggle.com/stackoverflow/stacksample`

[2] `https://github.com/karthikm-0/csc2515-project-part2`

methods that leverage word embeddings from pre-trained language models [12, 11] or sentence embeddings [3] have also been successfully used for best answer prediction on datasets other than Stack Overflow [13, 15].

Our work is novel in several ways. First, unlike previous literature where a subset of features were utilized as inputs to machine learning models, we explored novel combinations of a larger subset of features. Second, we treat the task of predicting the best answer as a regression problem, not a classification problem like prior work. Third, we experimented with the use of more complex models on simple features such as CNNs [7] and LSTMs [5]. We also explored complex text representations and models on the Stack Overflow dataset.

3 Data

The initial dataset was a set of 10 percent of the questions and answers available on the Stack Overflow website with a total of 1,264,204 questions and 2,014,375 answers. Question data points contained the title, body, score, creation date, close date, and user ID of the poster. Answer data points contained the body, creation date, score, user ID of the poster, and the ID of the associated question. The text bodies were in HTML format. To avoid ties, we removed all questions and answers for questions where there was more than one answer with the same highest score. We also removed all question-answer pairs for questions with fewer than seven answers. After this filtering process, the dataset contained 11,919 questions and 111,647 answers. Figure 1 shows the answer score distribution and Figure 2 shows the per-question best answer score distribution. Excluding punctuation, links, HTML tags, and code from the text, the mean length of questions was 87.7 words and the mean length of answers was 64.3 words. In addition, there were 31,337 unique words in the set of questions and 124,137 unique words in the set of answers with a total of 133,968 unique words across the questions and answers combined. There were 9,892 unique users who asked questions and 66,035 unique users who provided answers with a total of 72,403 unique users overall. Using this final dataset, we chose a 80/20 training/testing set split.

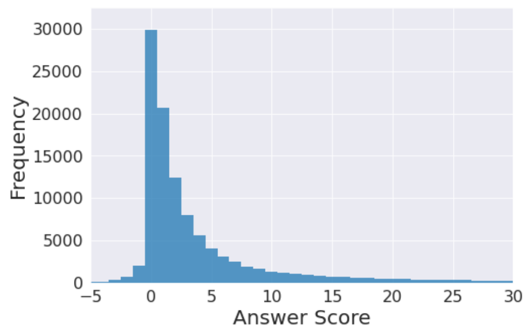


Figure 1: Answer Score Distribution

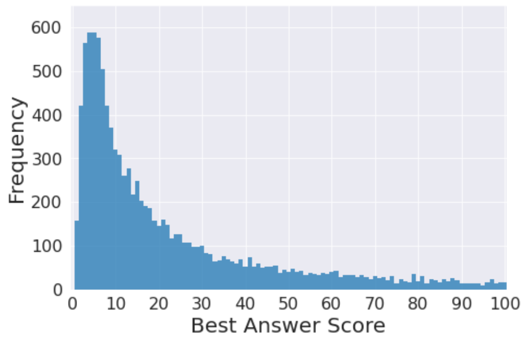


Figure 2: Per-Question Best Score Distribution

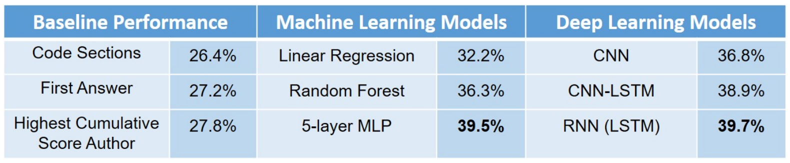
4 Models

We chose to treat the prediction task as a regression problem. Because there was no field indicating which answer was accepted for each question and many questions on Stack Overflow do not have accepted answers, the best answer for each question-answers set was considered to be the one with the highest score. We applied per-question normalization for answer scores in the dataset by dividing each answer score by the highest score among answers to the corresponding question. Our model predicts a normalized score for each answer in the test set. For each question in the test set, the answer with the highest predicted score is then predicted to be the best answer. We extracted a novel set of 28 features for each question-answer pair including shallow linguistic features, vocabulary features, user-related features, and features based on the chronological ordering of answers by creation date (see appendix section 8.1). Vocabulary and linguistic features were extracted after preprocessing which removed punctuation, links, HTML tags, and code from text. The 28 input features were then pre-processed with a quantile transformer, which mapped each feature independently to follow a uniform distribution using their quantile information, making it robust towards outliers.

Our initial experiments used classical machine learning models with the scikit-learn package including the Linear Regressor, Random Forest Regressor and Multi Layer Peceptron Regressor. We then explored several deep learning models with the Tensorflow framework. Since some of the features have sequence correlations between each other, such as answers' and questions' statistics, we implemented a recurrent neural network (RNN) model with 3 bi-directional Long Short-Term Memory layers (LSTM), followed by dropout layers, a flattened layer and a dense layer for regression output. We then experimented with a convolutional neural network(CNN) with 2 one-dimensional convolutional (CONV1D) layers and two max pooling layers, as well as a model with 2 one-dimension convolutional layers followed by a unidirectional LSTM layer. The structure of the CNN and LSTM models are shown in Appendix 8.2.

5 Results

The per-question accuracy of our models were calculated by using the trained models to make predictions on the test dataset, and comparing whether the predicted best answers matched the ground truth of best answers. We chose 3 naive methods, such as picking the answer with the most code segments, the earliest-posted answer, or the answer from the user with the highest cumulative score from their other answers, to obtain per-question accuracy baselines. The performance for each of our models is summarized in Figure 3.



Baseline Performance		Machine Learning Models		Deep Learning Models	
Code Sections	26.4%	Linear Regression	32.2%	CNN	36.8%
First Answer	27.2%	Random Forest	36.3%	CNN-LSTM	38.9%
Highest Cumulative Score Author	27.8%	5-layer MLP	**39.5%**	RNN (LSTM)	**39.7%**

Figure 3: Per-Question Accuracy Results of Various Machine Learning Models

All three baselines achieved accuracies of around 27% with the highest being 27.8%. All of the models we evaluated surpassed the baselines. The best performing model was the LSTM with accuracy of 39.7% obtained after tuning several hyperparameters such as layer count, unit size, learning rate, dropout rate, and number of epochs. The second best performing model, with accuracy of 39.5%, was the MLP regressor with 5 layers of 20 neurons each. Overall, the LSTM outperformed the strongest baseline by 11.9%. However, despite experimenting with several machine learning models, our numerical feature based models stayed below 40% per-question accuracy.

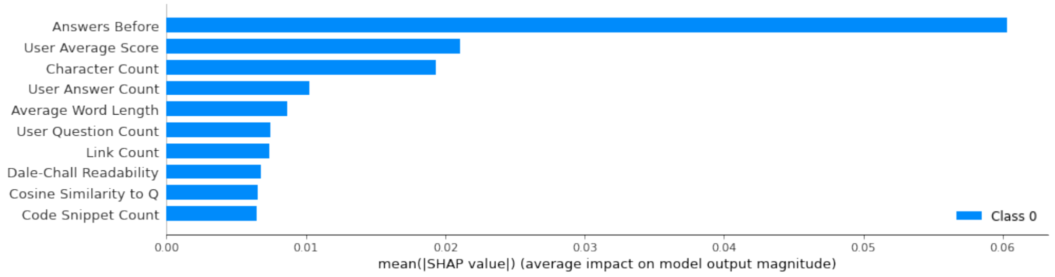


Figure 4: SHAP Summary Plot

To determine feature importance for our best performing models we used the SHAP (Shapley Additive Explanations) feature importance measure for deep learning models [10]. The top ten features by SHAP value are shown in Figure 4. We found the most important feature for each answer by a significant margin to be the number of answers that were posted before, which is likely because earlier answers are seen by more users. These results also indicate that user history is important, as answers from more reputable or prolific users may be more helpful or perceived as such. The remaining top features were primarily linguistic features, including character, link, and code snippet counts, which could have reflected the level of detail and additional insight in the answer. Another prominent linguistic feature category was readability, which included Dale-Chall readability score and average word length [14].

6 Discussion

We chose per-question accuracy as the performance metric for this project because of the uneven distribution of dataset labels. The overall percentage of answers that could be considered best answers was only 10.7 %. Consequently, if we chose to treat the task as a per-answer classification problem, the machine learning models would have learned to develop biased weights to predict that most of the answers were not best answers, as even if the models predicted no answers were best answers, per-answer accuracy would still have been 89.3%. Our machine learning models gradually improved in performance as complexity increased. With more trainable parameters, machine learning models can better fit the dataset and achieve increased performance. The 5-layer MLP model had 2701 trainable parameters, which was 45 times more than linear regression model's 60 parameters, and was capable of non-linear transformation to turn off some units' output, which helped the model achieve 39.5 % accuracy. Complex deep learning models such as CNNs require a large number of features to convolve through, and the limited amount of numerical features in the dataset may have limited the performance of these complex models. Among the deep learning models, the LSTM model outperformed the CNN model, likely because when it processed the dataset's features, it could correlate answer features that were several columns away, whereas the CNN could only capture correlation between neighbouring features. An interesting observation is that the MLP matched and even outperformed deep learning models. This could imply that the correlation and sequential information between features may not have been as strong, allowing the MLP with its feed-forward structure to approximate the function required for predictions better than the CNN and RNN architectures.

It is difficult to compare these results with prior work as we treated best answer prediction as a regression problem whereas prior work utilized classification. Lin et al. [9] only evaluated Python questions with between four and ten answers using classification and found their models could achieve up to 48% accuracy. In contrast, our regression approach achieved up to 40% accuracy where questions can be about any topic and contain at least seven answers, suggesting that it may work better than models from prior work for diverse questions, and for questions with many answers.

There were some limitations in our approach that should be addressed. Our method of selecting the answer with the highest score to be the best answer may not actually select the most helpful or highest quality answer in some cases. For example, answers posted earlier may receive higher scores than later questions with better content as shown in our SHAP feature importance calculations. Future work could explore different learning approaches for the task of best answer prediction, such as pairwise approaches where each training instance contains the question as well as one decidedly good answer and one decidedly bad answer [6]. Also, while we did include the number of code snippets in the answer body as a feature, the specific contents of each snippet were discarded from body text during preprocessing. Future work could incorporate additional features based on code embeddings [1] in answers, as the code could provide insights into what makes a good answer.

Future work could also explore the use of text content as opposed to just the numerical and shallow text features. We provide an initial exploration of word and sentence embeddings for this problem in Appendix 8.3. In particular, we used word embeddings to train a QA-LSTM [17]. We also attempted to use sentence embeddings to assess question-answer cosine similarity, train simple models (e.g. linear regression), and fine-tune a pre-trained BERT masked language model for classification and regression [3]. Despite these attempts, our initial results are not very promising (see Appendix 8.3), as none of the models using word or sentence embeddings outperformed the naive baselines. We hypothesize that this may stem from the dataset having questions from many diverse topics, as opposed to focused on just a specific type of question (e.g. Python programming help). Further, fine tuning large models such as BERT requires a large dataset, which is another limitation that could be addressed in future work.

7 Attributions

All group members contributed equally to this work. With daily virtual meetings in December, the team had engaging communication throughout the project, from brain-storming, researching, coding, presentation to final report writing. The detail tasks attribution are listed below:

Xiang:

- Generated Numerical features from the question answer dataset, including: Word count, character count, average character length, polarity, subjectivity, grade level, Dale chall level, reading ease metric, as well as corresponding features for questions.
- Trained and fine-tuned CNN and LSTM models. Fine-tuned MLP model.
- Covered the model and result section in Presentation. (Slide pages 15-19)
- Wrote section 4 model, Appendix 8.2 model structures, as well as part of section 5 results, part of section 6 discussion, and part of abstract in the report.

Ming:

- Performed exploratory data analysis including visualizations.
- Prepared the questions and answers dataset for feature extraction, including dataset filtering as well as data and text preprocessing.
- Extracted chronological ordering features, cosine similarity features, and some of the user history features from the dataset.
- Trained and fine-tuned linear regression, decision forest, MLP, and RNN LSTM models.
- Prepared and presented the dataset section and part of the model section in the presentation.
- Wrote section 3 Data, Appendix 8.1 model features, as well as parts of section 4 Models, section 5 Results, section 6 Discussion, and the abstract for the report.

Karthik:

- Extracted user-based features such as the cumulative answer score, average answer score, number of questions, and number of answers.
- Trained and fine tuned the CNN-LSTM model.
- Explored word embeddings and their use in training the QA-LSTM model.
- Explored sentence embeddings from pre-trained BERT model for cosine similarity and simple model training.
- Fine-tuned BERT model with current dataset and different Python-specific dataset.
- Presented setup, motivation, and prior work during project presentation.
- Write up of introduction, related work, parts of the discussion and abstract, and Appendix 8.3.

References

[1] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

[2] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. Moving to stack overflow: Best-answer prediction in legacy developer forums. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–10, 2016.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Hanyin Fang, Fei Wu, Zhou Zhao, Xinyu Duan, Yueting Zhuang, and Martin Ester. Community-based question answering via heterogeneous social network learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 122–128, 2016.

[5] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[6] Tuan Lai, Trung Bui, and Sheng Li. A review on deep learning techniques applied to answer selection. In *Proceedings of the 27th international conference on computational linguistics*, pages 2132–2144, 2018.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Lei Li, Daqing He, Wei Jeng, Spencer Goodwin, and Chengzhi Zhang. Answer quality characteristics and prediction on an academic q&a site: A case study on researchgate. In *Proceedings of the 24th international conference on world wide web*, pages 1453–1458, 2015.

[9] Jun-Wei Lin, Tzu-Chi Lin, and Peter Schaedler. *Predicting the Best Answers for Questions on Stack Overflow*. 2018.

[10] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26:3111–3119, 2013.

[13] Jamshid Mozafari, Afsaneh Fatemi, and Mohammad Ali Nematbakhsh. Bas: An answer selection method using bert language model. *arXiv preprint arXiv:1911.01528*, 2019.

[14] Emily Pitler and Ani Nenkova. Revisiting readability: A unified framework for predicting text quality. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, page 186–195, USA, 2008. Association for Computational Linguistics.

[15] Bhaskar Sen, Nikhil Gopal, and Xinwei Xue. Support-bert: Predicting quality of question-answer pairs in msdn using deep bidirectional transformer. *arXiv preprint arXiv:2005.08294*, 2020.

[16] Clayton Stanley and Michael D Byrne. Predicting tags for stackoverflow posts. In *Proceedings of ICCM*, volume 2013, 2013.

[17] Ming Tan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*, 2015.

[18] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30(5):981–997, 2015.

8 Appendix

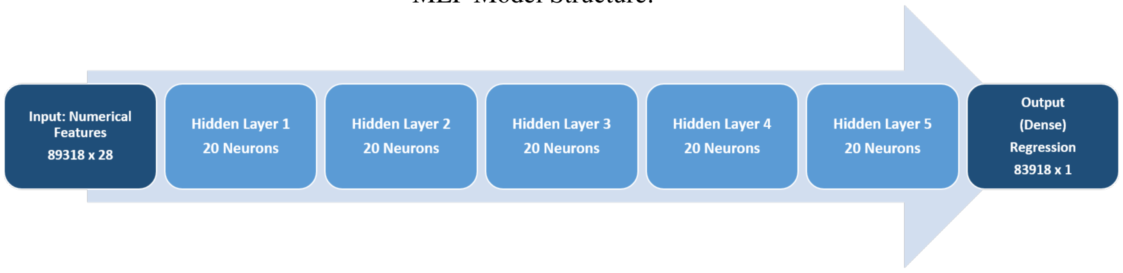
8.1 Model Features

Category	Feature
Shallow linguistic	Word count Character count Average word length Number of code snippets Number of links
Vocabulary	Cosine similarity to question Mean cosine similarity to other answers Dale-Chall readability estimate Flesch–Kincaid grade level Flesch–Kincaid reading ease Polarity Subjectivity
User history	Cumulative answer score from other answers by user Average answer score from other answers by user Total answers posted by user Total questions posted by user
Chronological	Number of answers posted before Number of answers posted after
Question	Question word count Question character count Question average word length Question link count Question Flesch–Kincaid grade level Question Flesch–Kincaid reading ease Question Dale-Chall readability estimate Question polarity Question subjectivity

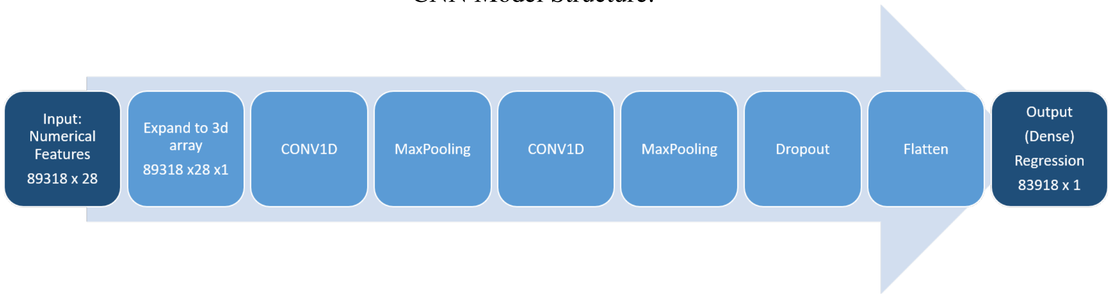
8.2 Deep Learning Model Structures

The 4 deep learning model structure is summarize in the following 4 figures. With several hidden layers and thousands of trainable features, they all perform relatively well with 36-40% per question accuracy. The full implementation is provided in Models_Training.ipynb in code submission.

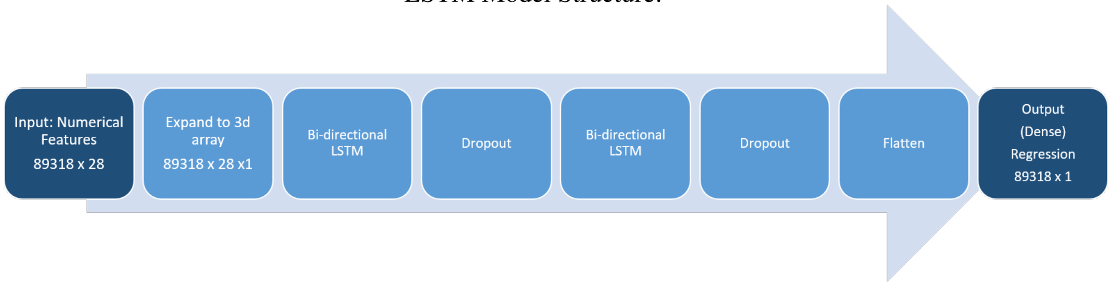
MLP Model Structure:



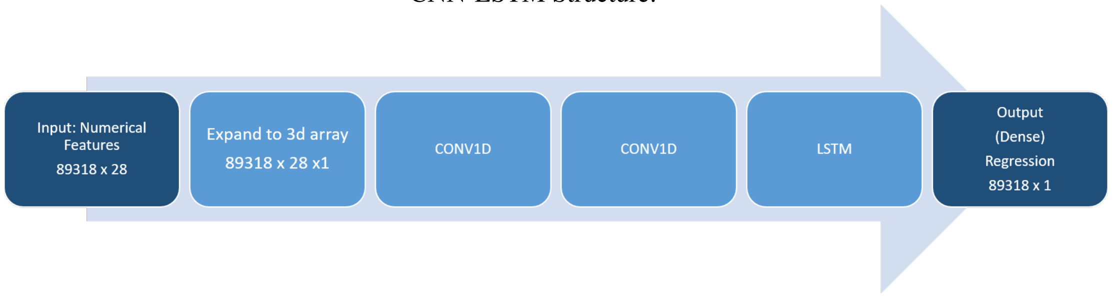
CNN Model Structure:



LSTM Model Structure:



CNN-LSTM Structure:



8.3 Training Models with Word and Sentence Embeddings

We also explored the use of advanced text representations beyond the simple Bag-of-Words approach found in some previous work. Here, we recap our attempts with word and sentence embeddings.

8.3.1 Word Embeddings

As word embeddings have showed great promise in best answer prediction in other domains we employed a similar approach to our Stack Overflow dataset. We implemented a QA-LSTM model where question-answer pairs were converted into a dense representation using word2vec embeddings of size 300. Then, each of the question-and-answer embeddings were separately fed into a bidirectional LSTM layer and finally summed to classify whether the pair contains the “best” answer, similar to how it has been used to predict a “correct” answer. This model was trained for 10 epochs with class weights to deal with the imbalanced dataset (only one best answer amongst many answers), but we did not find it capable of making accurate predictions as it has a lower accuracy than the naïve baselines (about 22%).

8.3.2 Sentence Embeddings

Next, we explored the use of the large, uncased pre-trained BERT model to generate paragraph embeddings for the question-answer pairs. These embeddings were generated individually (for question and answer).

After generating these embeddings, we tried a few different approaches. First, we calculated the cosine similarity between questions and their answers. This did not yield a better accuracy than the baseline naïve approaches (15.9%). We also explored combining cosine similarity with the other features described in the "Data" section but also did not see an improvement. Next, we tried to use the questions and answer embeddings (size 1024) as features to train simple regressors but did not see an improvement, even when combining them with the other numerical features described earlier. Lastly, we generated embeddings for question-answer pairs together, where a question makes up the first 256 words while an answer makes the last 256 words. This approach also did not outperform the naïve baselines, with or without the inclusion of other numerical features.

Since the pre-trained BERT model did not perform very well, we attempted to fine tune a both a base, uncased and a large, uncased model on our dataset instead. First, we attempted to create question-answer pairs through tokenization and experimented with different lengths, settling at 512 for the combined question-answer pair with the large model. We applied this fine-tuned model for classification and regression. For classification, we attempted both unweighted as well as weighted training to account for imbalanced data. In the regression approach, we trained the model on normalized scores for each answer to a particular question, where best answers have a score of 1 while other answers are below 1. However, even with these approaches, the performance did not yield an improvement above the naïve baselines. For instance, the regression approach led to an accuracy of only 23.7%.

We hypothesize that the poor performance may be due to the training set being too small to enable BERT learn a domain-specific problem such as best answer prediction on technical question-answer pairs. Further, the dataset contains questions from many different topics and programming languages, making it very difficult to attain a generalized understanding of what makes a good answer. Lastly, unlike problem domains where the correct answer is clearly different from an incorrect answer, on Stack Overflow answers can be good to varying extents and this is subjective, which may be hard to learn for the BERT model.