What Defines a Good Stack Overflow Answer Post, An Automated Post Rater

Github Link: https://github.com/YanpeiTian/CS229

1. Abstract

Major Q&A sites for professional and enthusiast programmers, like Stack Overflow, have gained phenomenal popularity during recent years. Programmers are relying on these sites to seek inspiration, find solutions and help others. However, many of the question owners do not select their most preferred answer. Therefore, it would be very inconvenient for other viewers to quickly identify useful information, especially for some unpopular questions with zero vote. This project attempts to address this issue by building an automated agent that can identify "good" answers. A feature extractor is first used to preprocess the data. Then, we apply logistic regression, SVM, and neural network to train our model. With our implementation, we can get ~70% prediction accuracy when measured against the accuracy metric[1].

2. Introduction

As there are many desired qualities that define a good answer. We built our own feature extractor to gauge our data. The input to our feature extractor is the raw data obtained from the Stack Overflow online public archive. We used SQL to query related questions, answers, and user information from it.

Then we preprocessed the data to obtain relevant features, such as comment count, post length, user reputation, user profile views, total user upvotes, total user down votes, the number of code words used in the description, the length of code blocks, hyperlinks (reference), and edit, etc. Other than the manually selected features mentioned above, we also considered the overall style and fluency of an answer. We trained a unigram and a bigram language model using exemplary text that matches the overall writing style of a good answer post. Lastly, we used the BERT[2] embedding model to extract the contextual meaning for the question-answer pairs to augment our feature epace.

Then, the extracted features were fed into learning models (e.g. logistic regression, SVM, neural networks)[3]. The prediction accuracy is then defined as the percentage of correctly rated posts measured by whether the post is marked as "accepted" by the question owner.

3. Related work

Extensive research has been conducted on predicting best answers on general Q&A sites (e.g. Quora and Yahoo! Answers) and recent work has been trying to identify the characteristics of the good programming-related answers on Stack Overflow. According to a qualitative study [2], for answers to java questions, textual explanations for code examples are as important as the code examples themselves. However, many previous studies have only used relatively "shallow" language features to predict answer acceptance. For example, [3] and [4] used body length (e.g. number of characters, words, and/or sentences) and unit length (e.g. words per sentence, characters per word, Flesch-Kincaid score) as measurements for the elaborateness (or conversely, conciseness) and readability of answer to predict answer acceptance. Other language features, such as similarity between question and answer (e.g. cosine similarity) and polarity (sentiment), have also been utilized by researchers to build the prediction model [5,6]. Apart from the textual contents, users reputation, technical

[1] Our definition of a "good" answer: The answer marked as "accepted" by the person who asked the question.
[2] BERT: Bidirectional Encoder Representations from Transformers is a technique for NLP pre-training developed by Google.
[3] Models used: Logistic Regression, SVM, Simple Neural Network, Hierarchical Neural Network.

contents (e.g. code, hyperlinks), and metadata (e.g. comments, response time) are also identified to be indicative of answer quality and have been extracted as features for the prediction models [7,8]. Although previous research has created a thorough set of relevant non-language features, enough attention has not been paid on the code, an important part for programming answers. Most studies simply used the existence of code snippets (binary feature) [5,7] or the length of block code (e.g. word or line count) [6,9]. Nevertheless, as [2] pointed out, providing inline documentation (e.g. inline code) and step-by-step solution using multiple code chunks can also improve the quality of answers, both of which have not been studied quantitatively.

In terms of the models, logistic regression and decision tree models have been used by many studies while few work has tested the performance of neural network models. Many studies reached an accuracy of around 70% on balanced datasets [4,10]. Only one study [9] has utilized neural network models on a set of features ranging from code, textual, non-textual, and user features to predict acceptance of answers to java questions and achieved a prediction accuracy of 70.9% which is comparable to random forest model (accuracy 71.7%).

4. Dataset and Features

We scrape all the posts generated by users with tag “Python” on Stack Overflow [11] during the period of 01/01/2018 - 05/01/2018 as our training and validation data, and posts generated by users on 06/01/2018 as test data. In total, we have 46287 training and validation data points, and 486 testing data points. Labels are identified by whether the answer was accepted by the question owner. The question posts with no accepted answer are excluded to ensure all data is labeled. Since different features have relatively different scales (e.g. feature “user reputation” can go as high as ~100000, while feature “edit” would only be either 0 or 1), we normalized all features to make them relatively comparable for logistic regression, SVM models.

Summary of features we use in the training process:
BERT Embedding: contextual word embeddings to the answer and question post bodies.
UnigramCost: Text fluency measured by unigram.
BigramCost: Text fluency measured by bigram.
parsed_CommentCount: how many comments are followed to this answer.
parsed_BodyLength: how many words the answer has.
parsed_UserViews: how many people have viewed this user’s profile.
parsed_UserUpVotes: how many upvotes this user has earned for all of his previous answers.
parsed_UserDownVotes: how many downvotes the user has earned for all of his previous answers.
InlineCode: how many times code is used within text description.
BlockCode: how many code blocks are used to illustrate the solution.
BlockCodeLine: how many lines of code are used in all code blocks.
Hyperlink: how many reference hyperlinks this answer contains.
Edit: whether this answer has been edited.

Details about n-gram features: An n-gram sequence model is a function that, given n consecutive words, provides a cost based on the negative log likelihood that the n-th word appears just after the first n-1 words. The cost will always be positive, and lower costs indicate better fluency.

Details about BERT feature sets: BERT (Bidirectional Encoder Representations from Transformers) is a new breakthrough method in NLP (Natural Language Processing) released in 2018 by Google researchers. It is a type of transformer encoder stack which utilizes the idea of transfer learning. The researchers pre-trained the BERT model on large datasets (e.g. Wikipedia webpages) with given tasks. Then the trained models could be used for other NLP applications to generate word embeddings that encode contextual information. [12]

The pre-trained model being used in this project is “bert-base-uncased”, it is being configured as 12-layer, 768-hidden, 12-heads, and 110M parameters. [1]

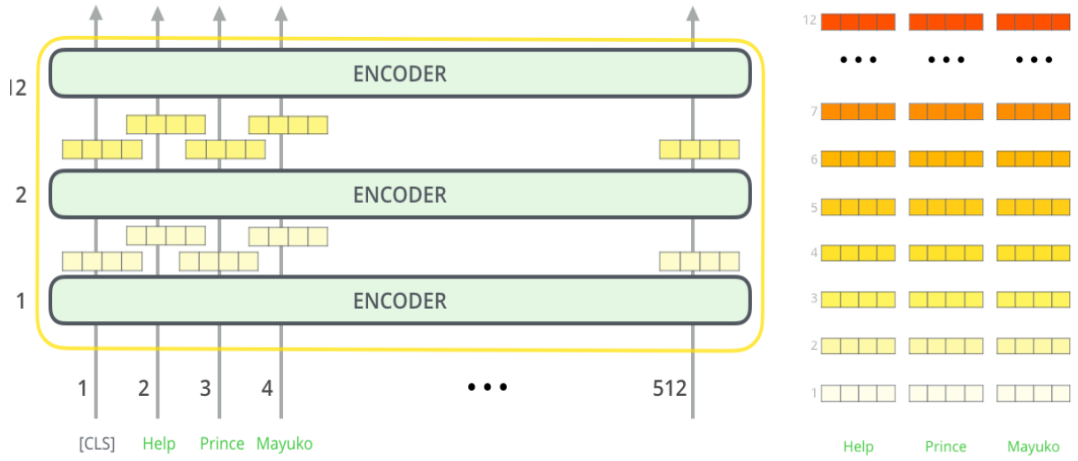


Figure 1. Bert model layer structures [12]

Each word in a text would have a vector with length of 768 in every encoder layer. There are a total of 12 such layers for the “base” model. All of those parameters together contribute to the whole contextual representation of a text. However, using all of them would be too computationally expensive, different pooling strategies (shown in Figure 2) has been explored to condense useful information [12].

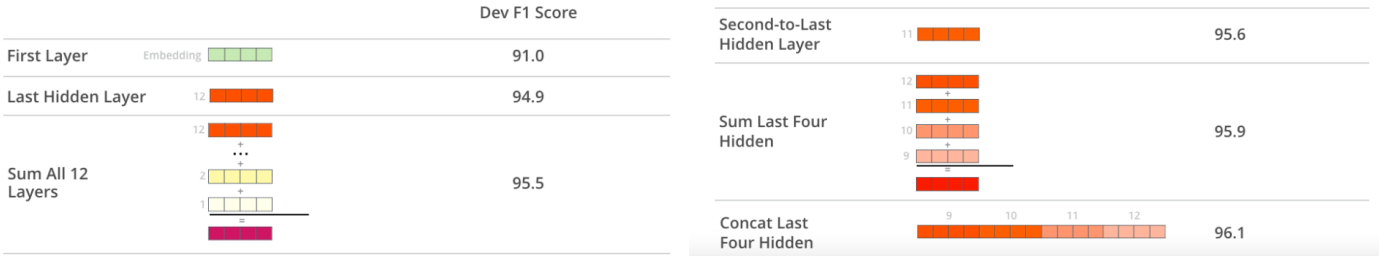


Figure 2. Bert model performance with different pooling strategies [12]

The last four layers has shown to be containing most of the information; hence the second to last layer is selected in this project in considering both performance and computational complexity. For the second to last layer, all of the word vectors are again being averaged to obtain a final overall representation of the whole text. Therefore, the total final output from the BERT model would be a 768×1 float vector with values ranges around -2~2. Both question and answer text bodies are passed into BERT to obtain an embedding vector. A 768×2 vector is generated for each sample.

5. Methods

Because the automated rater rates each question as “will be accepted” and “will not be accepted” we need a classification algorithm to train the model.

The most fundamental algorithm for binary classification is Logistic Regression, so we started with it to have our first trial. Logistic regression uses a sigmoid function to convert a continuous linear regression algorithm into a discrete model. It predicts the probability of $P(y=1|x;\theta)$. The complete hypothesis expression of it is:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

by assuming that all training examples are generated independently, the log likelihood function can be obtained as:

$$l(\theta) = \sum_{i=1}^{n} y^{(i)} log h(x^{(i)}) + (1-y^{(i)}) log(1-h(x^{(i)}))$$

Then gradient descent and Newton’s method can usually be used to maximize this log likelihood.

Our second attempt to solve this problem is SVM. SVM is a classification algorithm that seeks a decision boundary that maximizes the geometric margin, effectively giving very confident predictions. This can be formulated as a constrained optimization problem as follows:

$$max_{\gamma,w,b}\gamma,\ s.t.\ y^{(i)}(w^T x^{(i)} + b) \geq \gamma,\ i = 1, ..., n\ and\ ||w|| = 1$$

As a foray to improve the performance of our model, we also used neural network as the training algorithm. A neural network is formed by layers of neurons. Each neuron consists of a weight vector and an activation function. A neuron takes a vector input, computes the weighted average and transforms the result according to its activation function. A layer is formed by stacking multiple neurons. And, lastly, a neural network is formed by stacking multiple layers.

To solve the imbalance feature space issue (768×2 BERT features compared with 13 other features), we also use a hierarchical neural network architecture. The hierarchical architecture first condenses the BERT features from 768×2 dimension to 70 dimension. Then the 70-dimensional condensed BERT vector is concatenated with our original 13 feature vector to form the input to the final neural network.

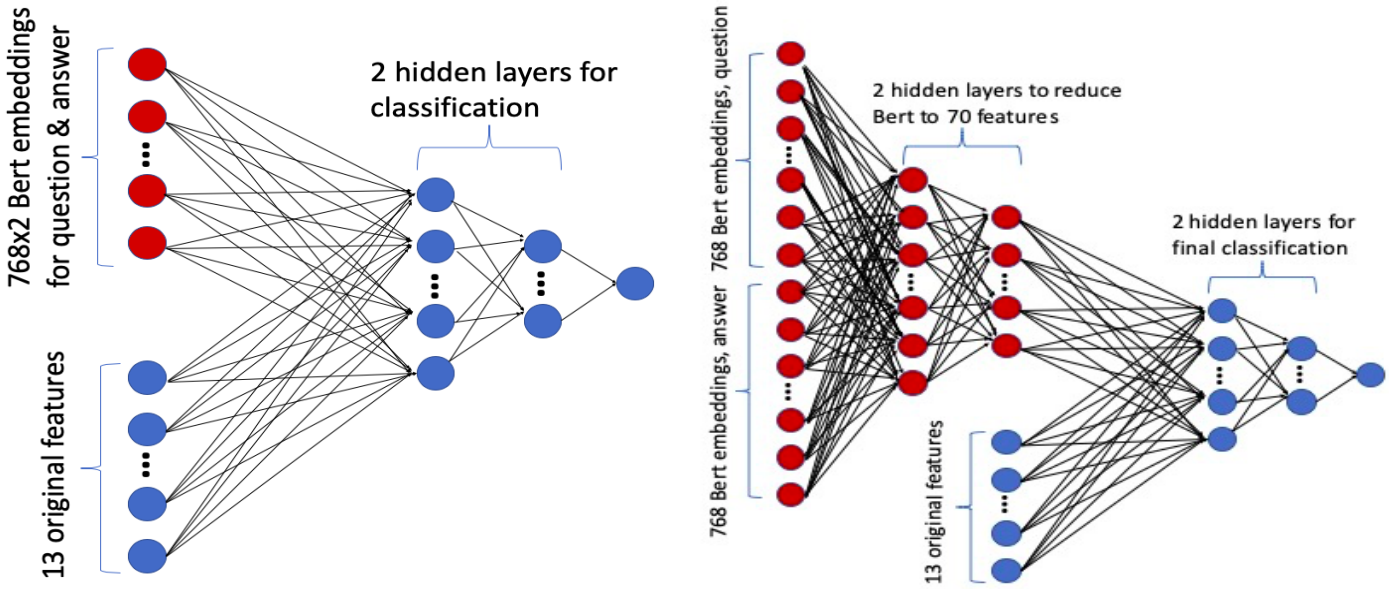


Figure3: Simple Feed-forward and Hierarchical Neural Network Architecture

6. Experiments/Results/Discussion

Since the dataset (excluding posts without accepted answers) is quite balanced, accuracy is used as the primary measure of evaluation. The following table illustrates training and testing results for different models:

Model	Training Accuracy	Test Accuracy
Logistic Regression w/o BERT	0.67	0.65
Logistic Regression w/ BERT	0.67	0.65
SVM w/o BERT	0.65	0.67
SVM w/ BERT	0.79	0.68
Simple NN w/o BERT	0.75	0.70
Simple NN w/ BERT	0.98	0.61
Hierarchical NN w/ BERT	0.67	0.70

Table 1: Training and Testing Results

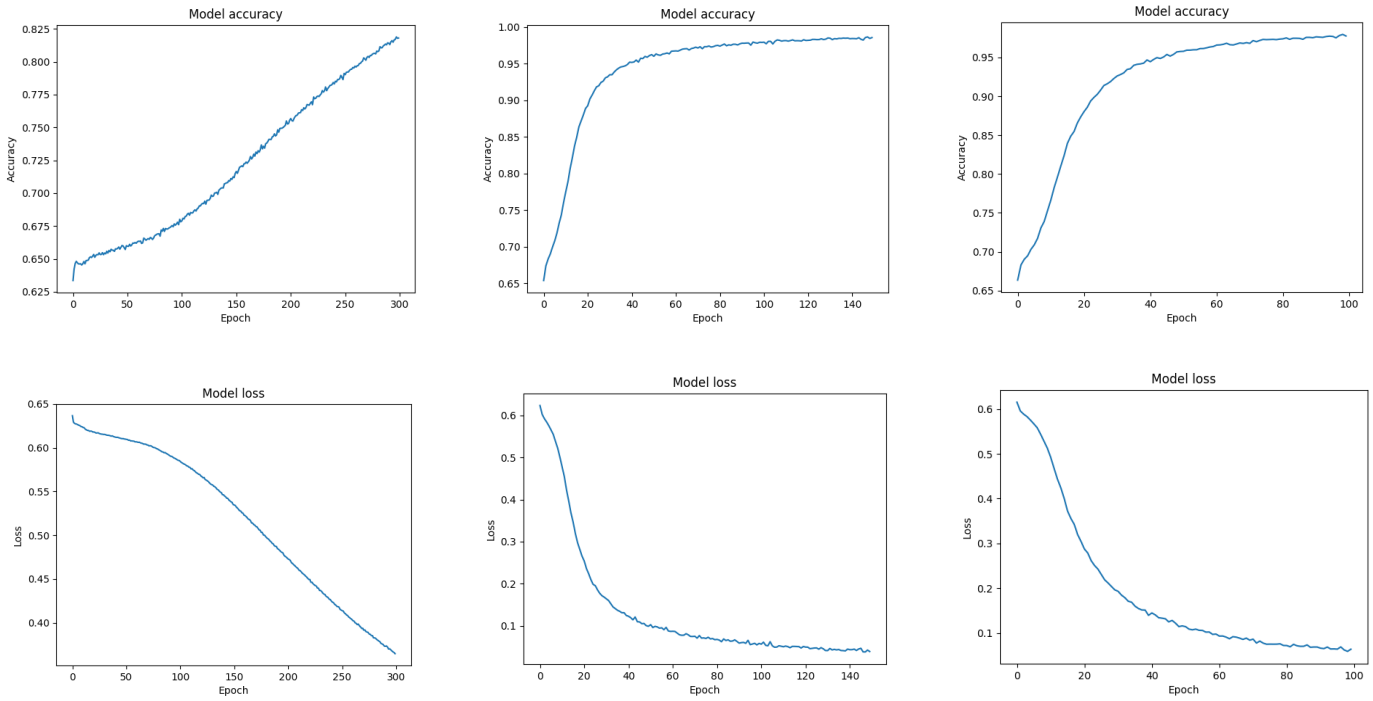
		Actual Value	
		Positives	Negatives
Predicted Value	Positives	87 True Positive	39 False Positive
	Negatives	111 False Negative	249 True Negative

Table 2: One Example of Confusion Matrix for Hierarchical Neural Network with BERT

Result Analysis:

1. Logistic Regression: The training and testing performance are the same for logistic regression w/o BERT and w/ BERT. Our hypothesized explanation is as follows: Due to the 'shallowness' of logistic model, the learning algorithm cannot effectively use the information encoded by the BERT embedding;
2. SVM: Incorporate the BERT embedding will increase the prediction performance of the model. And the overall performance of SVM is also better than logistic regression due to the increased model capacity;
3. Neural Networks: We achieve our highest prediction accuracy by using neural networks. Another interesting observation we have is the overfitting issue with model 'Simple NN w/ BERT'. If we just combine the 768×2-dimensional BERT feature and the 13-dimensional original feature, the simple feed-forward neural network will heavily overfit, justifying the necessity of the hierarchical neural network architecture.

Training Curves:



Simple NN w/o BERT | Simple NN w/ BERT | Hierarchical NN w/ BERT

Figure 4. Training and test accuracy graphs with different models

7. Conclusion and Future Work

With BERT model, N-gram, and 13 other engineered features, the highest prediction accuracy for "good" post can be achieved is around 70%. This result meets the initial expectation. Since the models do not have any contextual prior knowledge about the topics (knowledge of Python in this case), inferring correctness of the answers from the question itself would be difficult. For future work, other models, such as various decision trees could be tested. In addition, more useful features need to be explored to incorporate the prior domain knowledge into the model.

9. Contributions

Yanhao Jiang: Project Proposal, Project Milestone, Bert Model, Logistic Regression Implementation, Neural network implementation, Final Report, Poster.

Yanpei Tian: Project Proposal, Project Milestone, Language Model, Logistic Regression Implementation, SVM Implementation, Simple neural network implementation, Final Report.

Chunyue Wei: Project Proposal, Project Milestone, Feature Selection, Hierarchical neural network implementation, Final Report, Poster.

10. References/Bibliography

[1] Google-Research. “Google-Research/Bert.” *GitHub*, 18 Oct. 2019, https://github.com/google-research/bert.

[2] Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. 2012. What makes a good code example?: A study of programming QandA in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)* (pp. 25-34). IEEE.

[3] Gkotsis, G., Liakata, M., Pedrinaci, C., Stepanyan, K., and Domingue, J. 2015. ACQUA: automated community-based question answering through the discretisation of shallow linguistic features. *The Journal of Web Science*, *1*(1), 1-15.

[4] Calefato, F., Lanubile, F., and Novielli, N. 2016. 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* (p. 13). ACM.

[5] Tian, Q., Zhang, P., and Li, B. 2013. Towards predicting the best answers in community-based question-answering services. In *Seventh International AAAI Conference on Weblogs and Social Media*.

[6] Gantayat, N., Dhoolia, P., Padhye, R., Mani, S., and Sinha, V. S. 2015. The synergy between voting and acceptance of answers on stackoverflow, or the lack thereof. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (pp. 406-409). IEEE Press.

[7] Calefato, F., Lanubile, F., Marasciulo, M. C., and Novielli, N. 2015. Mining successful answers in stack overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (pp. 430-433). IEEE Press.

[8] Xu, S., Bennett, A., Hoogeveen, D., Lau, J. H., and Baldwin, T. 2018. Preferred Answer Selection in Stack Overflow: Better Text Representations... and Metadata, Metadata, Metadata. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text* (pp. 137-147).

[9] Omondiagbe, O. P., Licorish, S. A., and MacDonell, S. G. 2019. Features that Predict the Acceptability of Java and JavaScript Answers on Stack Overflow. In *Proceedings of the Evaluation and Assessment on Software Engineering* (pp. 101-110). ACM.

[10] Burel, G., He, Y., and Alani, H. 2012. Automatic identification of best answers in online enquiry communities. In Extended Semantic Web Conference (pp. 514-529). Springer, Berlin, Heidelberg.

[11] Stack Exchange Data Explorer. “Query Stack Overflow.” Stack Exchange, 12 Dec. 2019, https://data.stackexchange.com/stackoverflow/query/new.

[12] Alammar, J. 2019. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). [online] Jalammar.github.io. Available at: http://jalammar.github.io/illustrated-bert/ [Accessed 11 Dec. 2019].

[13] Mccormickml.com. 2019. BERT Word Embeddings Tutorial · Chris McCormick. [online] Available at: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/ [Accessed 11 Dec. 2019].

[14] Mccormickml.com. 2019. BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick. [online] Available at: https://mccormickml.com/2019/07/22/BERT-fine-tuning/ [Accessed 11 Dec. 2019].

[15] Scikit-learn.org. 2019. *scikit-learn: machine learning in Python — scikit-learn 0.22 documentation*. [online] Available at: https://scikit-learn.org/stable/ [Accessed 12 Dec. 2019].

[16] Keras.io. 2019. *Home - Keras Documentation*. [online] Available at: https://keras.io/ [Accessed 12 Dec. 2019].

[17] Pytorch.org. 2019. *PyTorch*. [online] Available at: https://pytorch.org/ [Accessed 12 Dec. 2019].