Best Answer Prediction on Stack Overflow Data Set

Taikai Takeda
0949784
taikait@uci.edu

Weicheng Yu
11842901
weicheny@uci.edu

Xingwei Liu
16942125
xingweil@uci.edu

Abstract

Our project aims to automatically predict the most popular answer from a set of user-provided answers to a specific technical questions in Stack Overflow. We will use two different type of features: answer style and QA correlation. In terms of model, we concentrate on simple ones such as logistic regression because scalability matters. Our main contributions are 1) improved an accuracy score by adding various features, 2) found effective features that reflect characteristics of this specific data set.

1 Introduction and Problem Statement

For many people, Q&A websites like Quora have become the go-to places when they run into problems. These websites usually keep track of how effective an answer is by counting the number of votes it receives from users. Higher votes usually means better quality. Reading some of the most popular answers on Stack Overflow, we wonder if there is a pattern in them. The motivation for this project was the idea of being able to ”guess” the most popular answer to a problem without any support of knowledge base.

We will concentrate on Stack Overflow QA data set because it is technical QA. Most of previous work is focused on non-technical QA, for example, Baidu Zhidao (Wang et al., 2010) and Yahoo! Answers (Surdeanu et al., 2011; Qiu and Huang, 2015). We believe that there are some inherent qualities of an answer to a technical question that make it popular among users. Our hypothesis is that a good answer should provide adequate information, be it text, images, hyperlinks, or code blocks. In addition, the quality of writing should also be an important factor because people tend to favor elegantly written answers over casually written ones. By experimenting different machine learning techniques, we hope to discover preferences of answers among Stack Overflow users.

2 Related Work

This task is called Community Question Answering (cQA). A lot of work has been done in this domain, which shows that it is quite an interesting problem in NLP. Some of the difficulties include retrieving answers for a new question from previous question-answer pairs (Zhou et al., 2016). Another difficulty comes from learning to rank answer candidates given a question, which is closely related to what we will do in this project.

Typically, only the best or the most popular answer is important for Q&A. Hence, it is possible to not rank every answer.The traditional way to tackle this problem is to combine some features that seem to be useful for this task (Surdeanu et al., 2011). Recently, some other approaches have been tested out using Neural Network, which models correlation between questions and answers (Tan et al., 2015; Qiu and Huang, 2015).

3 Data Sets

Internet Archive[1] provides archives questions and answers from StackOverflow. The original file was in XML format. Every node in the file represents an entry of either a question or an answer. Every entry has the following fields, which are ID, number of up votes, parent ID,

[1] http://archive.org/download/stackexchange

number of children, and the HTML formatted content.

3.1 Preprocessing Data

As is usually the case, the data downloaded from the Internet are not organized in the format we want. We have to do some preprocessing in order to make the data easier to use.

3.1.1 Data Format

The difficulty with the data-set is that is it was organized in a flat structure. As discussed above, the file we retrieved contains an array of entries of either question or answer in creation order. In this project, we need our data-set to be in pair of question-answer-set format, so we first preprocess the data into JSON format where keys are question ID and values are answers. Because the whole file is nearly 40GB, to save memory, we write a JSON file when 10,000 questions are processed.

3.1.2 Data Reformatting

We first wrote a python script to divide the file into 111 smaller files organized by answer numbers. From `Figure 1`, we can see that the number of questions with a certain number of answers decrease rapidly as the number of answers increase. By organizing our data this way, we can easily pick out the data containing questions with a specific number of answers. One thing that is worth mentioning is that all the text contents of questions and answers are stored in HTML format, which means that there are many HTML tags in our data. Because we might need the pure text information, we wrote a number of utility functions to remove HTML tags in our data using Python regular expression and json libraries. Then we can feed the processed data to the NLTK library and start the feature extraction process.

3.2 Statistical Information

# of Question	10789362
# of Answer	17650222
Max # of Ans	518
Avg # of Ans	1.64

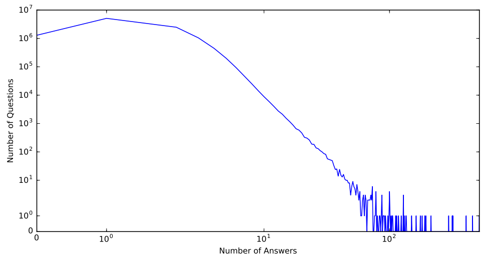


Figure 1: **Answer Number Distribution**: This figure shows the distribution of question numbers against questions of specific number of answers.

3.3 Normalization

Another problem of the data-set is that the number of votes vary greatly depending on the attributes of the questions. For example, there are many questions that are very basic but are commonly asked. These questions tend to have best answers with a very large amount of up votes, because more people are concerned with the questions and they tend to have well-written answers. However, very technical and geeky questions tend not to get so much attention as there is a smaller amount of people who know and care about them. Hence, it is possible that even the highest voted answers have only a few up votes. The discrepancy creates problem in the actual training process because it will favor more commonly asked questions. Therefore, we need to regularize the votes so that all questions are trained under the same scale. The way we regularize the vote is to rescale the votes for each question by setting the highest voted answer to 1. Other answers will be set proportionally.

3.4 Train/Test Data

To simplify the problem, we use the data-set of questions that have same number of answers. When train models, we splite the data-set by 75/25 into train and test data. Also, for hyperparameter optimization, we use the training set and do 3-fold cross validation to test the accuracy of different parameter configuration.

4 Description of Technical Approach

Overall predictive model structure is illustrated in Figure 2. Generally speaking, there are two major components in it: features and models. Here, we will explain them in detail.

4.1 Preliminary

Here, we will introduce notations. Let question set be $Q = \{q^1, \cdots, q^N\}$ where N is the number of questions in data set. Similarly, let answer set be $A = \{a^1, \cdots, a^N\}$ where each a^i is sequence of answer candidates $a^i = (a^i_1, \cdots, a^i_M)$. We fixed the number of answer candidates M because we cannot directly compare the result with different M. Target is denoted as $T = \{t^1, \cdots, t^N\}$. Each t^i is $M-$length sequence of answer score, $t^i = (t^i_1, \cdots, t^i_M)$ where $t^i_j \in [0, 1]$. True label set Y is $Y = \{y^1, \cdots, y^N\}$ where $y^i \in \{1, \cdots, M\}$ is label of the best answers $y^i = \arg\max_j t^i_j$, i.e. y^i-th answer $a^i_{y^i}$ is the best answer for q^i. Our goal is to obtain good prediction function $f : Q \times A \to \{1, \cdots, M\}$. Evaluation measure that shows goodness of this predictor is explained in the next subsection.

4.2 Evaluation Measure

We employ Re-ranking Precision at 1st position (P@1) as an evaluation measure, since we are interested mainly in the best answer. We divide the number of correctly predicted answers by the total number of QA pairs. The fraction represents the accuracy of our prediction for the best answer.

$$P@1 = \frac{\#\{\text{correctly re-ranked into the first position}\}}{\#\{\text{all QA pairs}\}}$$

More formally,

$$P@1 = \frac{\sum_{i=1}^N I(f(q^i, a^i) = y^i)}{N}$$

where I is indicator function. Clearly, this measure largely depends on the number of answer candidates M, so we used fixed $M \in \{2, 5, 10, 15, 20\}$ for evaluation purpose.

4.3 Features

In terms of extracting features, we tried two different approaches: one is to analyze answer documents independently, and the other is to capture correlation between questions and answers. Since these two factors are orthogonal, we combine both features using these two different approaches and feed them to predictive models. Here, we will explain these two approaches separately.

4.4 Answer Style Analysis

We assume that good answers exhibits a similar style. For example, an answer with lots of casual and informal words is not likely to be favored by users. In contrast, an answer with correct grammar and formal vocabulary is likely to get a vote up. Hence, even if we do not know what the original question is, we can find a ”better” answer. Here, we are going to model the preference of answers in the following ways:

- **BoW:**
  Use the basic Bag of Words as a feature to train our models.
- **Sentence Counts Heuristic:**
  Choose the best answer based on the number of sentences an answer has. The answer with the most counts is the best.
- **HTML Tags:**
  Since the answers we get are embeded in HTML, we can choose either to pass in the whole HTML objects as input or to extract only text answers as input.
- **Heuristic:**
  We build some simple heuristic methods to explore possible features for our machine learning models. Those heuristic functions are based on number of characters, number of words, number of sentences, and average sentence length. For some of them, we also have an additional version that up-weighs code block's length (UWC) to capture the importance of the code blocks.

4.5 Correlation between Question and Answer

Correlation between questions and answers is another concern. A good answer should responds directly to the question. Therefore, modeling this correlation should be helpful to find the best answer. However, modeling the correlation is not trivial. The first approach is to compute a similarity score between question and answer documents. We vectorize them with BoW, and project them onto hidden space using Latent Semantic Analysis (LSA). Then, we compute cosine similarity of them and use it as a feature.

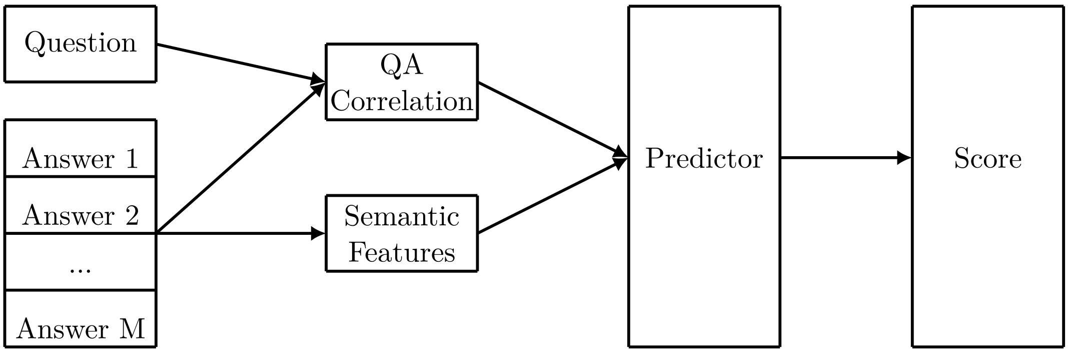


Figure 2: **Predictive Model**: Answer style and QA correlation are two different type of features. With these features, predictor produces predicted scores. An answer with maximum score is predicted as the best answer.

4.6 Predictive Models

We picked some models because characteristics of the model can make a difference. Also, we tried both regression models and classification models because binarizing the score might affect our results. For classification models, target value is binarized with threshold set to 1.0, which means that only the best answers are labeled as 1. Indeed, there would be the room to optimize this threshold, so we will do it in hyper parameter tuning, which is planned to be done in 9th week. We did not choose non-linear models because it takes much more time to optimize (e.g. RBF-SVM).

5 Software

We share all the code on GitHub [2]

The primary software we use is Python version 3.5. Our code are majorly written in python with NLTK as tokenization library and scikit-learn as the machine learning library. *Main.py* runs the predictor and produce the accuracy results. *Preprocess.py* divides data by the number of answers. *Others* are some helpers to transform data and process text.

We wrote all other Python code for this project by ourselves.

6 Experiments and Evaluation

Here, we will show our experiment results.

6.1 Accuracy Scores

We added various features to the basic models. First, we added QA correlation feature, then mixed heuristic features. Here, we will show result for each one.

The accuracy result of adding QA correlation and html tags is shown in `Table 1`. We can say from this result 1) the number of answers significantly influences the accuracy as expected. 2) For a given model and a specific answer configuration(whether we keep the HTML tags and whether we take QA correlation into consideration), keeping the HTML tags in the tokenization phase significantly improves the accuracy results in all models. 3) Logistic regression and ridge regression produce slightly more accurate predictions under the same configuration. 4) The first three models work relatively well and all produce close results. However, linear regression exhibits some deficiencies in the prediction. 5) Taking QA distance into consideration does not produce a distinguishable difference in the prediction result.

We also added heuristic features. We set up some mixed models that use the output of heuristic functions as additional features to logistic regression model. We tested them in 10-answer data-set. The result is shown in `Table 2`. Here, we show how adding heuristic features affects the result. There is also no difference between up-weighing code block and the normal weighing method. This is because logistic regression using bag of words method already extracts the information of code blocks from text. In addition, incorporating heuristic functions to machine learning models only increases less than one percent of test accuracy.

[2] https://github.com/bigsea-t/best-answer-prediction

# Ans	LSVM	LoR	RiR	LiR	
2	61.23	**61.28**	61.19	59.45	(!tag, !cor)
	69.17	***69.28***	68.97	*66.96*	(tag, !cor)
	69.06	**69.15**	*69.01*	66.89	(tag, cor)
	61.51	**61.54**	61.49	59.60	(!tag, cor)
5	30.81	**30.96**	30.62	27.32	
	43.47	43.29	42.96	40.37	
	43.50	*43.38*	*43.12*	40.44	
	30.81	**30.83**	30.76	27.52	
10	17.52	18.05	**18.49**	15.49	
	30.49	30.15	***30.83***	*27.40*	
	30.69	*30.20*	30.35	26.57	
	18.54	18.49	**19.36**	15.34	
15	12.83	13.35	**16.75**	13.09	
	22.77	22.51	**23.04**	*21.73*	
	20.68	***24.08***	*23.56*	19.63	
	13.61	**15.18**	13.87	11.26	
20	7.76	**8.62**	6.90	6.03	
	14.66	13.79	***15.52***	*13.79*	
	16.38	*14.66*	14.66	12.07	
	8.62	6.03	**8.62**	**8.62**	

Table 1: **P@1 accuracy scores for Linear SVM, Logistic Regression, Ridge Regression and Linear Regression**: **bold** numbers mean the best result of four models given a number of answers on a feature processing configuration. *italicized* numbers mean the best result of a model among all feature processing configurations on a number of answers. `!tag` means removing tags. `tag` means keeping tags. `col` means using cosine distance between answers and questions in latent semantic space as a feature (q-a correlation). `!col` means not using q-a correlation as a feature.

Methods	Default	UWC
Original	32.22 30.20	N/A
# Char	32.48 30.45	32.03 30.83
# Word	32.41 30.98	N/A
# Sent	32.48 31.46	32.46 31.56
# W/S	32.28 30.45	N/A

Table 2: **Heuristic mix-model accuracy:** The train and test accuracy of mixed model of logistic regression and heuristic methods

6.2 Hyper-Parameter Optimization

We exploited random search on Hyper-Parameter Optimization (Bergstra and Bengio, 2012) rather than grid search. We tried 50 iterations for every model's parameter space and chose the 5 best configurations from each model according to their validation score. We tested on the 10-answer data-set with all selected configurations on our models. The result is shown in `Table 3`.

From this result, we conclude that because of the simplicity of linear models, the hyper parameters do not affect the result in a significant way. Only linear regression models have noticeable improvement when applied hyper parameter optimization. The difference in terms of the parameter choice is influenced by the intercept parameter, which means whether the data are centralized. In addition, we propose that the reason of discrepancy is a result of the data distribution, which is closer to origin in hyper space. Details about the parameter type and range and the result are in code.

6.3 Overfitting

In `Figure 3`, we show the train and test error for different number of answers. We see that the performance of our model is relatively stable for answers with number of questions under 15, and train error and test error are close. However, as we evaluate on questions with a larger number of answers, the performance deteriorates drastically, with test accuracy rapidly approaching 0 and train accuracy rapidly approaching 1. As we have shown in `Figure 1`, the number of questions shrinks exponentially in response to the increase of number of answers. As we have shown in the dataset section, the average number of answers a question has is 1.64. Therefore, the increasing gap between train and test error, we believe, is the consequence of the over-fitting effect, which is caused by the reduced size of dataset.

6.4 Effective Features

Here, we will show which features are highly effective in terms of prediction. We only show the result about logistic regression because all the model have similar accuracy result. We

Rank	LSVM	LoR	RiR	LiR
default	32.27	32.22	31.06	29.28
	30.49	30.15	30.83	27.40
1	**32.14**	32.12	32.11	**32.07**
	30.69	30.11	30.88	**30.54**
2	32.20	**32.19**	32.12	32.07
	30.35	**30.20**	30.93	30.54
3	32.20	32.24	32.11	30.27
	30.35	30.15	30.88	27.78
4	32.20	32.17	**32.11**	30.27
	30.35	30.06	**30.98**	27.78
5	11.92	31.91	32.12	N/A
	11.47	29.72	30.88	N/A

Table 3: **Hyper-Parameter Optimization Result**: The result of hyper-parameter optimization. Rank shows the rank of accuracy on validation set, e.g. rank 1 is the best configuration according to the random search. 'default' is the score before optimization. The first row of each rank is training accuracy, and the second is test accuracy. The best test accuracy results are bold.

can easily interpret this model by looking at the weight: larger weight means positive features , and smaller weight means negative features. Here are resulting top-10 most effective and least effective features:

Positive

edit, note, update equivalent, jsfiddle, finally, fact, instead, documentation, fix

Negative

answers, think, tried, hope, maybe, answer, thought, guess, said, mentioned

Generally, positive features include *formal* words such as "finally" and "fact", and words that indicate *edit frequency* of an answer such as "edit" and "update" (Figure 4 is typical example of an answer which include the word "edit"). This result reveals that formal and eagerly edited answers are likely to be the best answers. On the other hand, negative features are more *ambiguous* words such as "maybe" and "guess" and words that *refer another answers* such as "answers" and "said" (Figure 5 is typical example of an answer which include the word "answers"). This shows that being ambiguous and referring another answer have negative effect in receiving more votes.

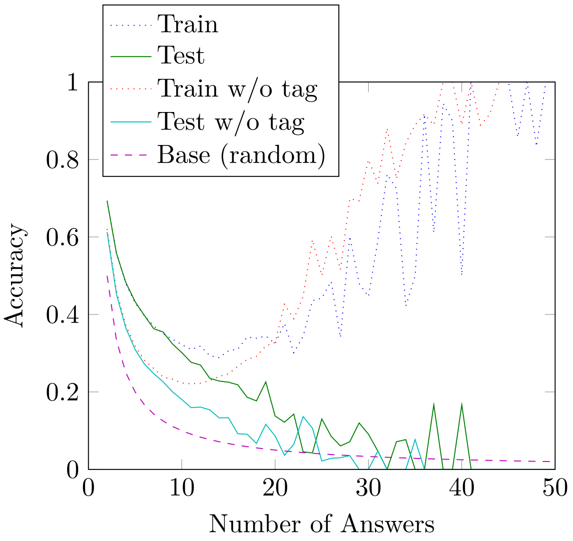
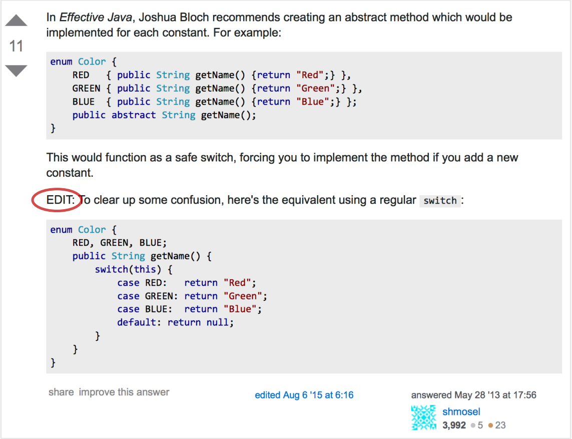


Figure 3: **Dynamics of Accuracy**: The performance of Logistic Regression model on data-sets with different number of answers (without q-a correlation as feature).

Interestingly, if we choose to keep the HTML tags, we find some additional top features ”href”, ”hr”, ”code”, and ”li”, which have higher scores than the previously mentioned words. Our assumption is that rich text elements increase credibility of answers because these links usually redirects to websites that are highly credible or focuses on a very specific domain of problems.

7 Discussion and Conclusion

In this project, we explored some feature extraction methods and predictive models for the Q&A problem taken from StackExchange. We digged into every part of our model building process and tried different approaches in selecting functions and parameters. As a result, we were able to generate a large number of results reflecting how each selection affects our model accuracy. According to our results, on the one hand, choosing different predictive models does not yield a significantly different result. On the other hand, the ways in which we do feature extraction have a significant impact on the accuracy result. We also find that correlation between Q&A using latent semantic analysis does not affect a lot. We conclude that question answer correlation, at least on vocabulary space, does not make a differ-



In *Effective Java*, Joshua Bloch recommends creating an abstract method which would be implemented for each constant. For example:

```
enum Color {
    RED   { public String getName() {return "Red";} },
    GREEN { public String getName() {return "Green";} },
    BLUE  { public String getName() {return "Blue";} };
    public abstract String getName();
}
```

This would function as a safe switch, forcing you to implement the method if you add a new constant.

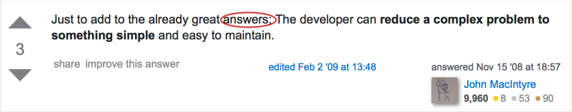
EDIT: To clear up some confusion, here's the equivalent using a regular `switch`:

```
enum Color {
    RED, GREEN, BLUE;
    public String getName() {
        switch(this) {
            case RED:   return "Red";
            case GREEN: return "Green";
            case BLUE:  return "Blue";
            default: return null;
        }
    }
}
```

edited Aug 6 '15 at 6:16

answered May 28 '13 at 17:56
shmosel
3,992 ● 5 ● 23

Figure 4: **Typical High-voted Answer**: Example of preferred answer with the word "edit". This word is an indicator of updated answer.



Just to add to the already great answers; The developer can **reduce a complex problem to something simple** and easy to maintain.

edited Feb 2 '09 at 13:48

answered Nov 15 '08 at 18:57
John MacIntyre
9,960 ● 8 ● 53 ● 90

Figure 5: **Typical Low-voted Answer**: Example of preferred answer with the word "answers". It shows the world "answer" refers other answers.

ence in terms of answer popularity. Because of the fact that choosing good feature extraction technique yields much better result, in order to improve the performance of our model, we can try more advanced feature extraction techniques of document such as paragraph2vec or POS information.

In addition to getting accuracy results, we also obtain some effective features of this specific data set. These features give us insights of what a good answer looks like. Being able to interpret some of our results is actually an advantage of choosing simpler model over complex model since it only reveals more obvious traits of our data-set. It also has the advantage of having less scalability issue than complex models.

References

James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

Xipeng Qiu and Xuanjing Huang. 2015. Convolutional Neural Tensor Network Architecture for Community-based Question Answering. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*.

Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. 2011. Learning to Rank Answers to Non-Factoid Questions from Web Collections. *Computational Linguistics*, 37(2):351–383.

Ming Tan, Bing Xiang, and Bowen Zhou. 2015. LSTM-based Deep Learning Models for non-factoid answer selection.

B Wang, Xiaolong Wang, C Sun, Bingquan Liu, and Lin Sun. 2010. Modeling semantic relevance for question-answer pairs in web social communities. *Proceeding ACL '10 Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, (July):1230–1238.

Guangyou Zhou, Yin Zhou, Tingting He, and Wensheng Wu. 2016. Learning Semantic Representations with Neural Networks for Community Question Answering Retrieval. *Knowledge-Based Systems*, 93:75–83.