Best Answer Prediction in Community-based Question-Answering Services

Sunyam Bagga[1] and Qianyu Liu[2], Jin Guo[3]

***Abstract*— A Community-based Question-Answering (CQA) service provides a platform for users to post their questions and answer others' questions. Currently, selecting the best answer for a given question is done manually on various CQA sites which leads to a large number of questions without a marked best answer. In this paper, we propose a method to automate the process of best answer selection by utilizing supervised machine learning techniques. The most important task is that of designing features to capture different aspects of the answer. We evaluate our approach on a Stack Overflow dataset and show how it would perform in practice, tackling the issue of class imbalance that is seldom discussed in existing works.**

I. Introduction

Community-based Question-Answering (CQA) services aim to provide a collaborative environment that enables exchange of knowledge. Registered users in such services can directly contribute to the environment by asking or answering questions. They can also react to questions or answers posted by others by leaving comments, casting upvotes or downvotes. The importance and social impact of CQA services is evident from the heavy traffic observed on popular CQA sites such as Yahoo! Answers, Stack Overflow, and Quora. Since most questions receive multiple answers, it is important to evaluate the quality of the answers and identify the *best answer* among them. Moreover, other users with similar questions rely heavily on the marked *best answer* especially when they don't have enough background to evaluate the answers themselves, or there are too many answers posted.

In various CQA sites, many questions are left without a marked best answer [1]. This could be due to lack of action from the askers, or their confusion in deciding on the best answer. In such a scenario, people would hesitate to rely on any answer even if there are many reasonable answers posted. This could be detrimental to knowledge spreading. Even worse, some CQA services remove such questions after a period of inactivity leading to loss of knowledge. A more reliable way to mark the best answer is to use the answer with the most upvotes since more upvotes normally implies a higher level of agreement among community-members. This work, therefore, aims to recommend the answers that could potentially receive the most upvotes. Given a question and a set of candidate answers, our work identifies the best answer by utilizing features from various perspectives: the textual description of the answers, the credibility of the answerers, the similarly between the answers and the questions and the similarity among competing answers.

We work with Stack Overflow[1] which is a domain-specific CQA site that caters to programming related questions and involves users ranging from novice programmers to experienced developers. Our method is able to identify the best answer with an accuracy of 70.1%. The techniques presented in this work can be applied to other CQA sites with minimal effort. We also release our code[2] to encourage future research in this direction.

II. Related Work

There have been several studies on CQA services in the past. Some previous studies [2] [3] took a qualitative approach, investigating factors that influence the selection of best answers. They performed a large scale analysis on CQA platforms, with little focus on building an actual predictor. Adamic et al. [2] collected one month of Yahoo! Answers activities, and examined the diversity and quality of questions and answers on the platform. They attempted to predict whether an answer would be selected as the best answer using four features and reported accuracy ranging from 69.2% to 72.9% depending on the category forum. Blooma et al. [3] conducted a study to investigate the factors that influence the selection of the best answer in Yahoo! Answers by analyzing three kinds of features: social, textual and content-appraisal features. Furthermore, Sahu et al. [4] aimed to predict best answers from only question-answer relationship and user profile, and achieved an accuracy of 69.1%. They considered the answerer's score and her "expertise" by examining the question-tags associated with her answers. In a related study, Xiang et al. [5] proposed an attentive deep neural network to predict whether an answer is "good". They used a labeled dataset of "good", "bad", and "potential" answers to evaluate their model and reported a F1-score of 58.4%. Lastly, Tian et al. [6] reported an accuracy of 72.3% using a set of 16 features covering answer content, answer context, and question-answer relationship. The features used in our work incorporates all the major categories from previous work. In addition, we adopted word embeddings using Word2Vec [7] to better capture the semantic information while calculating textual-related features, and introduced user-profile features. Word2Vec captures the semantic relationship between words since it represents them by encoding the contexts of their surrounding words into a continuous-valued vector.

A common theme we encountered in related studies concerns the issue of class imbalance. Since there is only one

[1]https://stackoverflow.com/

[2]https://github.com/sunyam/BestAnswer_Prediction/

BestAnswer for each question, majority of the answers from CQA datasets are *NonBestAnswer*s. However, some studies [6] [4] [8] did not address this issue and only reported accuracy, whereas other studies [2] [3] [5] that reported precision/recall began with a balanced dataset. In most cases, they undersampled the majority class at the time of data collection, during which only the best answer and one or two random answers for each question were selected. Clearly, such an evaluation does not reflect the real-world scenario. Therefore in this paper, we work toward explicitly addressing this issue during evaluation.

III. Proposed Approach

A. Dataset

We used Stack Overflow's *pythonQuestions*[3] dataset, publicly available on Kaggle. It contains questions tagged with the "python" that were asked between August 2, 2008 and October 19, 2016. The dataset consists of 607,282 questions and 987,122 answers. We discarded questions that had less than 3 answers (including unanswered questions) which leaves us with 105,258 questions and 411,789 answers. Although examining questions with fewer than 3 answers would not be difficult, these questions do not provide sufficient answer-to-answer relationship. Out of the 411,789 answers, 272,129 belong to the class *NonBestAnswer* and the remaining 139,660[4] belong to the class *BestAnswer*.

B. Word Embeddings

To compute word embeddings, we trained a Word2Vec model on a combination of two datasets: *pythonQuestions* and *stackSample*[5]. We preprocessed all posts in our dataset using standard methods, i.e. removing tags, tokenizing and removing stop words and punctuations. The pipeline is shown in Fig. 1. We then train a 300-dimensional word embedding model with the preprocessed dataset using python's Gensim library [9]. We used a *context window* of size 10, and ignore all words with total frequency lower than 10. After experimenting with both skip-gram and continuous bag-of-words (CBOW), we found the performance to be very similar but slightly better with the CBOW model. Therefore, we only report the results for the CBOW case in this paper.

C. Features Description

We consider the following sets of features to predict the best answer for a given question:

- f_A captures the quality of the answer.
- f_U represents information about the user who answers the question.
- $f_{A \leftrightarrow Q}$ measures the relevance of the answer to the question.
- $f_{A \leftrightarrow A}$ captures how the answer in consideration competes with other answers to the same question.

[3] www.kaggle.com/stackoverflow/pythonquestions

[4] Note that for a question, there could occasionally be more than one best answer if they have the same number of upvotes.

[5] www.kaggle.com/stackoverflow/stacksample

The complete feature set is presented in Table I. In this section, we discuss in detail only those features that are not self-evident from the table.

1) f_A (Answer Content): This set of features aims to transform the answer content into the feature vector. f_4, called ans_len, is calculated by counting the number of characters in the answer. f_5 and f_6 represent the readability of the answer. We believe that an answer that is easily readable is more likely to be selected as the best answer. As done in [6], we measure readability using:

$$f_5 : readability\_1 = \max_i L_i \tag{1}$$

$$f_6 : readability\_2 = \frac{1}{N} \sum_{i=1}^{N} L_i \tag{2}$$

where L_i is the length of the i^{th} paragraph in terms of number of characters, and N is the total number of paragraphs in the answer.

2) f_U (User's Information): The two features in this set capture information about the user who wrote the answer A_i. We consider two aspects of the answerer: *ownerScore_A* describes how good she is at answering other questions, and *ownerScore_Q* describes how good she is at asking meaningful questions. The rationale behind this is that a user who is good at answering and asking questions will be more likely to write a best answer. Note that these values are calculated using the *pythonQuestions* dataset, and are not real-time Stack Overflow user reputation.

3) $f_{A \leftrightarrow Q}$ and $f_{A \leftrightarrow A}$: These two categories consist of 7 features in total ($f_9 - f_{15}$). f_{10} captures the 'similarity' between the answer and question, and f_{11}, f_{12}, f_{13} capture the 'similarity' among the competing answers. As can be seen in Fig. 1, we calculate the similarity using the standard cosine similarity metric that computes the cosine of the angle between two vectors. To convert a post into a real-valued vector, we take the average of the word embeddings of each word in that post. Hence in this way, each answer and question is represented as a 300-dimensional vector of real values.

In Fig. 1, Post A would be the question Q and Post B would be the answer A_i when calculating f_{10}; both posts A and B would be answers when calculating f_{11}, f_{12}, f_{13}. Concretely, these features are calculated using the following equations:

$$f_{10} : QA\_sim = sim(A_i, Q) \tag{3}$$

$$f_{11} : ave\_ans\_sim = \frac{1}{num(A_{k \neq i})} \sum_{k \neq i} sim(A_i, A_k) \tag{4}$$

$$f_{12} : min\_ans\_sim = \min_{k \neq i} sim(A_i, A_k) \tag{5}$$

$$f_{13} : max\_ans\_sim = \max_{k \neq i} sim(A_i, A_k) \tag{6}$$

Hence after combining the four categories, our feature vector is a 15-dimensional vector which we then feed to our classifier.

TABLE I: Set of features for a candidate answer A_i to the question Q. Set of other answers to Q are denoted by $\{A_k\}$.

Category	Index	Name	Description
f_A	1	url_tag	Number of URL tags present in A_i.
	2	pic	Number of illustration figures present in A_i.
	3	code	Number of code snippets present in A_i.
	4	ans_len	This measures the length of answer A_i.
	5,6	readability	This captures whether A_i is easy to read. See (1) and (2).
f_U	7	ownerScore_A	Number of upvotes (score) the answerer of A_i accumulated by answering other questions.
	8	ownderScore_Q	Number of upvotes (score) the answerer of A_i accumulated by asking questions.
$f_{A \leftrightarrow Q}$	9	timeSlot	Difference between Q's creation time and A_i's creation time.
	10	QA_sim	Similarity between A_i and Q.
$f_{A \leftrightarrow A}$	11	ave_ans_sim	The average of similarities between A_i and $\{A_k\}$.
	12	min_ans_sim	The minimum of similarities between A_i and $\{A_k\}$.
	13	max_ans_sim	The maximum of similarities between A_i and $\{A_k\}$.
	14	competitor_num	Number of other answers $\{A_k\}$ to question Q.
	15	ans_index	The order that A_i was created. For example, it was the 2nd answer to the question Q.

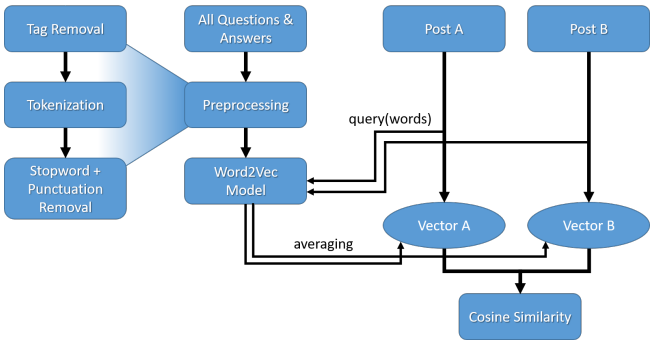


Fig. 1: Pipeline to compute similarity between two posts.

D. Classification

We used Random Forest [10] to predict whether a given answer, represented by the feature vector described above, is a best answer or not. This is essentially a binary classification task with labels: *BestAnswer* and *NonBestAnswer*. The major reason we selected this algorithm is because it works well with large datasets [11] and provides an efficient way to compute feature importances.

Each question has a set of answers, among which only one belongs to the class *BestAnswer* and the remaining belong to the class *NonBestAnswer*. This leads to a class imbalance issue where there are a whole lot of examples for one class and far less number of examples for the other class. In this work, we experiment with two different techniques to deal with this issue:

- Undersampling: As suggested in [12], we kept all the *BestAnswer* vectors and randomly sampled the same number of vectors from the *NonBestAnswer* class for training. This results in a training set of size equal to 2 times the number of original *BestAnswer* samples.
- Oversampling: We used Synthetic Minority Oversampling Technique (SMOTE), which creates new examples of minority class using interpolation and the k-nearest neighbors (See [13] for more details). This results in a training set of size equal to 2 times the number of original *NonBestAnswer* samples.

IV. Evaluation

We report accuracy, F1-score, precision, recall for three cases: (1) *As-is* where we make no changes to the training set, (2) *Undersamp* where we undersample our training set, and (3) *Oversamp* where we oversample our training set. Note that for all three cases, we do not modify the test set (imbalanced) since that reflects how the classifier would perform in a real-world scenario. Random Forest was implemented using Python's scikit-learn library [14]. We tuned the hyperparameters using a 10-fold cross-validated randomized search over a predefined range of values for different parameters. For this task, Random Forest works best with 1000 decision trees[6].

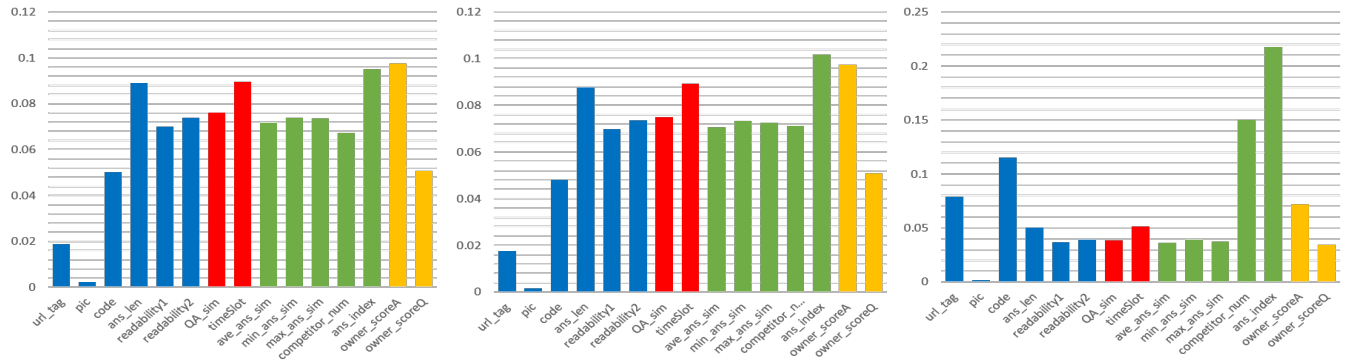
Class imbalance is an important issue that we addressed in our study to ensure that the evaluation results reflect how the model would perform in practice. We also want to note that since none of the previous works explicitly address this issue, they either used a balanced test-set or only reported accuracy. For this reason, we believe that directly comparing our results against existing work is not a fair comparison.

We split our dataset into training (80%) and test (20%) sets. The prediction results of Random Forest on the test data for all three cases is shown in Table II. As can be seen in the table, the best accuracy we get is 70.1% for the *As-is* and *Oversamp* case which is comparable to the work by Tian et al. [6]. Since Tian et al. do not report any other metrics, we can not compare them. Moreover, these results are not directly comparable since they use a different dataset (not publicly available) and make a different best-answer assumption.

When working with imbalanced data, accuracy is not the most suited metric to measure performance. Our model has low recall in the *As-is* case since it misclassified many minority-class posts as majority-class posts. In comparison, having a balanced training set (in *Undersamp* and *Oversamp*) contributed to a noticeable improvement in recall and F1-score.

Finally, we used scikit-learn's *feature_importances_* attribute which uses gini impurity [15] to measure importance.

[6]The complete set of hyperparameters for Random Forest and Word2Vec can be found in the README of our GitHub repository.



(a) As-is training set (b) Undersampled training set (c) Oversampled training set

Fig. 2: Distribution of feature importances for the three cases: (a) *As-is* (b) *Undersamp* (c) *Oversamp*.

TABLE II: Classification results for the three cases: *As-is*, *Undersamp*, and *Oversamp*.

	As-is	*Undersamp*	*Oversamp*
Precision	60.1%	48.9%	57.4%
Recall	33.6%	71.5%	43.2%
F1-score	43.1%	58.1%	49.3%
Accuracy	70.1%	65.3%	70.1%

Figure 2 shows the feature importances for the *As-is*, *Undersamp*, and *Oversamp* case. It can be seen in the figure that some features are clearly more important than other features. Particularly, *pic* is the least important feature for all three cases. The distribution is almost identical for the *As-is* and the *Undersamp* case. All the similarity features that utilize word embeddings are equally important (8%). The two most important features in both these cases are owner_scoreA and ans_index (about 10%). This result demonstrates that the answerers' reputation for answering other questions and when the answers are posted could be more reliable indicators for predicting answers receiving more upvotes. This finding also confirms the "Matthew effect" ("for to everyone who has, more will be given") discussed in many social network analysis [16]. It is surprising to see that the distribution drastically changes for the *Oversamp* case. The importance of ans_index shoots up to 22% and all the similarity features drop to less than 5%. This is perhaps due to new synthetic training examples created by SMOTE.

V. Conclusions and Future Work

We studied the problem of selecting the best answer given a question and a set of candidate answers. To the best of our knowledge, this is the first work that aims to predict the community-decided best-answer as opposed to the best-answer selected by the asker. Our evaluation on a relatively large Stack Overflow dataset shows that some features are considerably more important than others. Unlike some other related work, we report precision, recall, and F1-score which can be used as a baseline for future work in this field. The results show that this seemingly trivial task is hard and textual similarities, in particular, did not seem to improve the performance of the classifier.

In future, we would like to incorporate more features based on question-tags and comments made by other users. We also plan to predict the "goodness" of the answers by formulating this task as a ranking problem where each of the candidate answers are ranked instead of hard binary classification.

References

[1] L. Yang, S. Bao, Q. Lin, X. Wu, D. Han, Z. Su, and Y. Yu, "Analyzing and predicting not-answered questions in community-based question answering services." in *AAAI*, vol. 11, 2011, pp. 1273–1278.

[2] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman, "Knowledge sharing and yahoo answers: everyone knows something," in *WWW*. ACM, 2008, pp. 665–674.

[3] M. J. Blooma, A. Y.-K. Chua, and D. H.-L. Goh, "Selection of the best answer in cqa services," in *ITNG*. IEEE, 2010, pp. 534–539.

[4] T. P. Sahu, N. K. Nagwani, and S. Verma, "Selecting best answer: An empirical analysis on community question answering sites," *IEEE Access*, vol. 4, pp. 4797–4808, 2016.

[5] Y. Xiang, Q. Chen, X. Wang, and Y. Qin, "Answer selection in community question answering via attentive neural networks," *IEEE Signal Processing Letters*, vol. 24, no. 4, pp. 505–509, 2017.

[6] Q. Tian, P. Zhang, and B. Li, "Towards predicting the best answers in community-based question-answering services," in *ICWSM*, 2013.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[8] C. Shah and J. Pomerantz, "Evaluating and predicting answer quality in community qa," in *SIGIR*. ACM, 2010, pp. 411–418.

[9] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *LREC Workshop on New Challenges for NLP Frameworks*. ELRA, 2010, pp. 45–50.

[10] L. Breiman, "Random forests," *Maching Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[11] M. Zakariah, "Classification of large datasets using random forest algorithm in various applications: Survey," *International Journal of Engineering and Innovative Technology*, vol. 4, pp. 189–198, 09 2014.

[12] F. Provost, "Machine learning from imbalanced data sets 101," in *AAAI workshop on imbalanced data sets*, 2000, pp. 1–3.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[14] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[16] A. Marin and B. Wellman, "Social network analysis: An introduction," *The SAGE handbook of social network analysis*, vol. 11, 2011.