Automatic Assessments of Code Explanations

Predicting answering times on Stack Overflow

Selman Ercan,* Quinten Stokkink,† Alberto Bacchelli‡
{*s.ercan-1, †q.a.stokkink}@student.tudelft.nl, ‡a.bacchelli@tudelft.nl
Delft University of Technology, The Netherlands

***Abstract*—Users of Question & Answer websites often include code fragments in their questions. However, large and unexplained code fragments make it harder for others to understand the question, thus possibly impacting the time required to obtain a correct answer. In this paper, we quantitatively study this relation: We look at questions containing code fragments and investigate the influence of explaining these fragments better on the time to answer. We devise an approach to quantify code explanations and apply it to ~300K posts. We find that it causes up to a 5σ (single-tail significant) increase in precision over baseline prediction times. This supports the use of our approach as an 'edit suggestion': Questions with a low score could trigger a warning suggesting the user to better explain the included code.**

I. Introduction

Stack Overflow (SO) is a popular Question & Answers website for programmers, whose questions often include exemplary code snippets to better illustrate the problem.

One of the challenges of posing a question is to explain the problem clearly. Unclear explanations could lead to a longer time to answer, because askers might have to refine their questions as potential answerers ask for clarifications. Assessing the quality of an explanation is a non-trivial task. In this paper, we adopt the view that both the natural language (NL) explanation and the code snippets act as descriptors of the problem in question and can be used to determine the quality of an explanation, or explanatory value. In this light, we investigate the relation between quality of code explanation and answering time. In particular, we test our hypothesis that a higher correlation between the NL explanation and the code fragment is related to better quality.

We define the following research questions:

RQ1: Does the correlation between the natural language explanation and the accompanying code fragment impact the time to answer a post?

RQ2: Can the quality of a code fragment explanation, with respect to human readability, be assessed automatically?

We state human readability rather than simple correlation, because the textual description and the code snippets should be two *distinct* descriptors of the same problem. Otherwise, two identical texts would result in perfect correlation.

Based on the correlation values we obtain from our analysis of questions with code fragments, we develop four predictors of answering time. Our evaluation shows that the performance of the best predictor is statistically significantly better compared to the baseline value.

II. Related work

Nasehi *et al.* [1] investigate the attributes of good answers on SO, considering high score Java-related answers and analyzing their features. They show the impact of code fragments on post quality, as concise and well explained code examples are found to be indispensable parts of good posts.

Rigby and Robillard [2] focus on extracting code elements from informal documentation and assessing their relevance. Their findings indicate that the best feature to determine the relevance of a code element is *text type*, *i.e.*, whether the element occurs in a code fragment or in text.

Lin [3] investigates the qualities of summaries, implementing and testing various algorithms. The algorithms are then assessed by comparing the results with human assessments. The relevance of these algorithms stems from our second research question.

Our approach involves a combination of the second and third papers; we extract code from questions by considering the code fragments in it, and we use an algorithm from [3] for the correlation calculation.

III. Methodology

We conducted our research in three stages: (i) We performed initial research on a small subset of relevant posts on SO; (ii) we used a more advanced algorithm (L-ROUGE) for the full set of relevant posts from the SO dump [4] and compared to the simple algorithm from the initial research; (iii) we used a state of the art algorithm on a filtered subset of the set of relevant posts and compared to the advanced algorithm from the second stage.

A. Initial approach

The initial step verifies whether there is any correlation between the answering time of a post and the relation of the NL text with the code snippet. We retrieved a sample of 300 posts randomly chosen among those with: (1) a 'java' tag, (2) at least one HTML '<code>' tag in the text, and (3) an accepted answer. We determined the answering time of a post as the difference in minutes between the posting time of the question and of the accepted answer.

We devised a simple token similarity algorithm (TSA) to assess the explanatory value of a question: It computes the overlap between the code fragment tokens and the NL ones. The more code tokens also occur as NL, the higher the explanatory value. This showed a slight positive correlation.

B. L-ROUGE

After verifying a correlation between explanatory value and answering time, we maximized the set size using the same three filters of the initial phase. The resulting set (dubbed: FULL set) contains 301,480 posts. Figure 1 shows a scatter plot of the TSA compared to the post answering time.

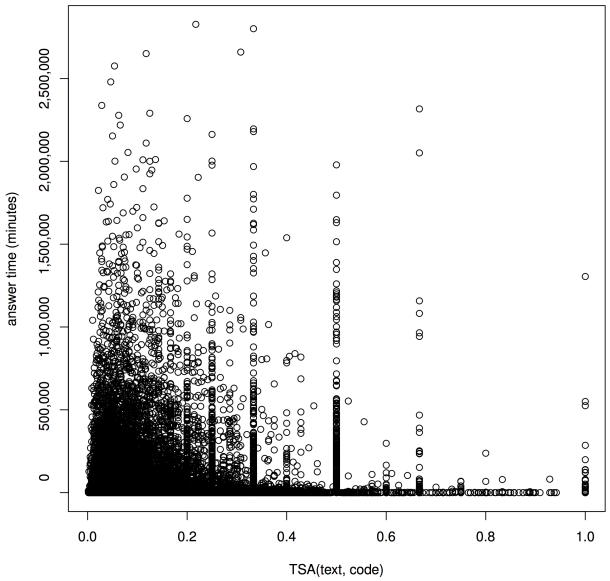

Fig. 1. Token correlation of the code and the explanation vs. the time to answer a post in minutes for the full data set

To find a metric with a denser scatter of data points (which would lead to better predictability), we investigated metrics used to score automatically generated NL summaries: The *ROUGE* metrics [3]. The advantage of using the *ROUGE* metrics is that they have a high correlation with human scores. This implies that the correlation scores which the *ROUGE* metrics supply give a good indication of human explanatory value between a summary and its text.

Most *ROUGE* metrics were created to compare NL with NL, thus we focused on those that did not include NL sentence structure. Among them, we found the *L-ROUGE* metric [3] to be the most appropriate. Important criteria for this selection were that the metric rewards (1) mentioning more elements from the code fragments and (2) following the code structure in the explanation. It also had one of the best average precisions spanning the data sets we consider.

We use the code snippet of a post as a candidate summary and the NL part as the reference text. Resulting *ROUGE* scores should indicate how well the code snippet summarizes the problem explanation. Figure 2 shows the scatter plot of the L-ROUGE algorithm compared to the post answering time.

C. Lists of terms

The last approach we utilized was to replace code fragments with lists of terms. The intuitive reason behind this is that a list of important terms better captures the essence of a code fragment than using it at face-value. We chose the following terms:

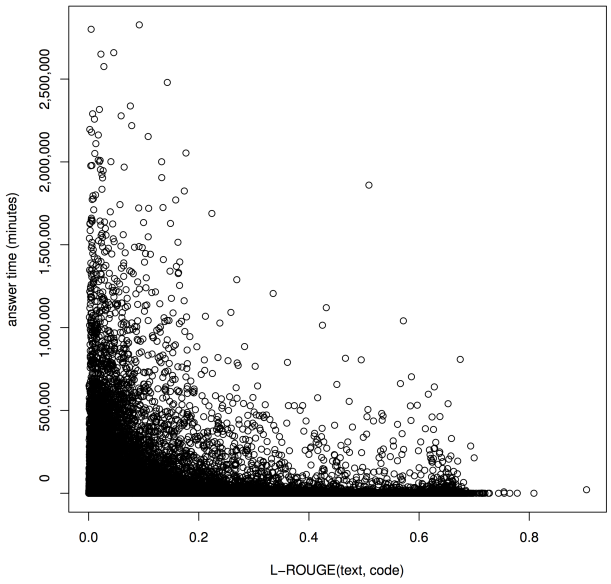

Fig. 2. L-ROUGE score of the code and the explanation versus the time to answer a post in minutes for the full data set

import types, method types, method names, field types, field names, variable types, and variable declarations. We created a custom Java-like grammar using ANTLR4 [5], which was fine-tuned for the aforementioned terms. Using this grammar we also filtered out all Java keywords to make sure maimed code would not contaminate the terms. Not all posts of the original FULL set could be parsed in this way though. Of the original 301,480 posts only 246,301 posts had a non-empty list of terms. Manual inspection revealed that excluded posts were mostly configuration files (*e.g.*, Maven *pom.xml* files). This second set, for which the Island Parser could discover any terms at all, was dubbed the TERM set.

First we repeated the L-ROUGE algorithm for this new set with just the raw code snippets, to compare the impact of the set filter. The overall distribution, with respect to the structure of the scatter plot (not reported), did not change significantly. Secondly, we used the list of terms, as outputted by our tools, with the L-ROUGE algorithm. Figure 3 shows the scatter plot of this term based strategy. The plot appears denser.

IV. RESULTS

Our work so far has given us four metrics: (1) TSA, (2) L-ROUGE on code snippets (L-ROUGE$_{code}$) using the FULL set, (3) L-ROUGE$_{code}$ using the TERM set and (4) L-ROUGE on term lists (L-ROUGE$_{terms}$) using the TERM set. To investigate the relationship between these metrics and answering time, we implement the former as predictors for the latter. When implementing them we strived for (1) full applicability and (2) simplicity; when necessary, we sacrificed precision to comply with these characteristics.

Given the shapes of the scatter plots, based on our metrics we define the following generic predictor, or estimator E:

$$P(post) : E(post_{text}, post_{code}) \geq S \rightarrow post_{answertime} < T$$

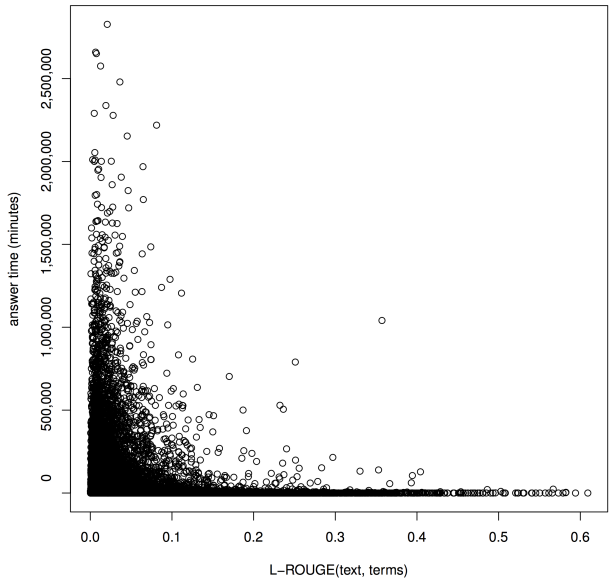

Fig. 3. L-ROUGE score of the terms and the explanation versus the time to answer a post in minutes for the data set with recognized terms

To achieve the best possible precision, we optimized the predictors locally to achieve optimal precision around the "bending" points in the scatter plots by tuning S (± 0.005) for different values of T. We used T values corresponding to one day, one week and one month. The resulting scores S, precisions and recalls for these times can be seen in Table I.

We now define our baseline against which we will compare our predictors. We define it as the predictor that, for any value of T, predicts that a given post will be answered within T. This corresponds to the notion that, without any further insight into the data, no better predictions can be made. Obviously the baseline will have 100% recall; since it labels all posts with "will be answered within T", it cannot fail to include any posts that are answered within T. The added value of our predictors therefore depends on the increase in precision they provide.

It is noteworthy that the four predictors have a higher precision than the baseline in all cases except for the token similarity predictor for $T =$ one month. This makes the TSA predictor unreliable, as it has better precision[1] than the L-ROUGE$_{code}$ predictor for the full set for predicting whether the post will be answered within a day, but is worse than the baseline for predicting whether the post will be answered within a month. This suggests that using TSA as a guideline will actually negatively impact the answer time of a post.

As the time in which the post is supposed to be answered increases from a day to a month, the increase in precision (P$_{inc}$) becomes smaller for all predictors. We can explain this as follows: As the maximum time for a post to be answered reaches infinity, having any predictor predict that it will be answered within this amount of time is equivalent to stating flat-out - without using predictors - that it will be answered in this amount of time. This is a side effect of a data set in which all posts eventually get answered.

[1]The next subsection discusses how much better this precision is. It turns out the difference in precision is not significant.

TABLE I
RESULTS BY SET, TIME, AND EVALUATOR

Set	T	E	S$_{opt}$	R	P	P$_{inc}$
Full	day	baseline	—	1.000	0.889	—
		TSA	0.16	0.263	0.900	0.011
		L-ROUGE$_{code}$	0.20	0.123	0.899	0.009
	week	baseline	—	1.000	0.952	—
		TSA	0.14	0.312	0.953	0.001
		L-ROUGE$_{code}$	0.20	0.123	0.959	0.007
	month	baseline	—	1.00	0.975	—
		TSA	0.14	0.311	0.974	-0.001
		L-ROUGE$_{code}$	0.18	0.143	0.980	0.004
Term	day	baseline	—	1.000	0.892	—
		L-ROUGE$_{code}$	0.25	0.099	0.903	0.012
		L-ROUGE$_{terms}$	0.03	0.293	0.903	0.012
	week	baseline	–	1.000	0.955	–
		L-ROUGE$_{code}$	0.24	0.106	0.963	0.009
		L-ROUGE$_{terms}$	0.03	0.291	0.961	0.007
	month	baseline	–	1.000	0.977	–
		L-ROUGE$_{code}$	0.24	0.105	0.981	0.004
		L-ROUGE$_{terms}$	0.03	0.291	0.980	0.003

TABLE II
STANDARD SAMPLE DEVIATION VALUE AND PRECISION INCREASE IN $\widetilde{s_n}$

Set	E	T	s$_N$	p$_{inc}$ ($\widetilde{s_n}$)
Full	TSA	day	227.005	1.565
		week	1131.623	0.549
		month	3326.014	-0.750
	L-rouge$_{code}$	day	239.058	1.119
		week	1165.604	3.092
		month	3284.045	5.819
Term	L-rouge$_{code}$	day	235.087	1.423
		week	1151.011	3.701
		month	3146.475	4.962
	L-rouge$_{terms}$	day	237.987	1.373
		week	1151.619	3.091
		month	3137.567	4.372

A. Statistical significance

To illustrate the statistical significance of differences in precision increase (last column of Table I), we present them in multiples of the standard deviation. The standard deviation for a predictor $P_{S,T}(post)$ with time interval T is calculated as the corrected sample standard deviation of the subset of answering times from the set for which $P_{S,T}(post)$ is applicable, which have an answering time of less than T. The resulting values for s_N per predictor can be seen in the fourth column of Table II.

The next step in determining the significance of the precision differences is to determine the change in precision of the predictors as the standard deviation changes. This means that the difference in precision values must be determined for a difference in T of s_N. We calculate the significance interval $\widetilde{s_N}$ (precision change per s_N shift in T) as follows:

$$\widetilde{s_N} = |(1 - \frac{Precision(E_{S,T})}{Precision(baseline)}) - (1 - \frac{Precision(E_{S,T+s_N})}{Precision(baseline)})|$$
$$= |\frac{Precision(E_{S,T+s_N})}{Precision(baseline)} - \frac{Precision(E_{S,T})}{Precision(baseline)}|$$

The $\widetilde{s_N}$ value can be used to express the precision increases resulting from increasing the prediction time T with the standard deviation (or rather corrected standard sample deviation). The resulting values can be viewed in the last column of Table II; only those increases in precision which lie more than a value of 1 apart are significantly different.

With this definition in terms of the standard deviation, we see that the TSA predictor is only a significant improvement over the baseline for the time span of a day. All of the other predictors show a significant improvement over the baseline. The only significant difference between the predictors, however, is the precision difference of the L-ROUGE$_{code}$ predictor and the L-ROUGE$_{terms}$ predictor. It shows the month prediction of the L-ROUGE$_{code}$ predictor for the full set is significantly better than the L-ROUGE$_{terms}$ predictor.

Some literature suggests a difference of 5σ for a difference to count as significant. If one adheres to this strict form, only the L-ROUGE$_{code}$ predictor for the time of a month provides a significant improvement over the baseline.

V. Discussion

Only one predictor significantly outperforms the others: The L-ROUGE$_{code}$ predictor on the FULL set using T equal to a month. However, the reason that the TERM set was introduced over the FULL set, was the contamination by other languages than Java (such as XML-based configuration files). This suggests that the L-ROUGE$_{code}$ predictor is even better for predicting answering time of languages other than Java.

When it comes to predictions for confirmed Java code, the best performer also seems to be the L-ROUGE$_{code}$ predictor. A possible problem for this predictor is its recall value. In the context of imposing a minimum score for a certain metric as a guideline for posts, we can view the recall of its predictor as the "evidence" of its efficacy. In fact, the recall is the fraction of posts for which the predictor precision is evaluated. While the L-ROUGE$_{terms}$ predictor does not offer as great a precision increase as the L-ROUGE$_{code}$ predictor, it does have more than twice the amount of data points supporting its precision increase. In other words, the chance that using the L-ROUGE$_{terms}$ predictor will have any effect is higher than the chance that the L-ROUGE$_{code}$ predictor will have effect.

The last thing to keep in mind when evaluating the values in Table I is the complexity of the algorithms. The TSA approach uses simple $O(n^2)$ matching, the L-ROUGE algorithm is NP-hard and the parser for terms lies in $O(EXP)$.

VI. Future Work

There is still opportunity for investigation of the applicability, adaptability and deployment of the method of this paper.

The first promising area of research would be to test the L-ROUGE algorithm on languages other than Java. The results of Table II showed that the L-ROUGE predictor performed significantly better when used with language contamination. Even though this improvement has been shown, for which other languages this holds is unknown. For example, the L-ROUGE algorithm might perform good for code snippets with configuration languages associated with Java, but fail for other major languages like Ruby (although this is not expected).

The second area of research one could expand upon is the correlation algorithm. This paper has chosen L-ROUGE for its ease of implementation and good correlation with human scores. It is, however, not the only correlation algorithm which correlates well with human scores.

Lastly, the actual deployment of the guideline as given by this paper should be investigated. Even though this paper has shown that the explanatory value of a code snippet plays a significant role in the answering time of a question, other factors might influence the answering time of posts. In the worst case there could even be negative correlation, which would originate from the false negatives of the predictors (which is why the predictors with higher recall values are less likely to possess such a negative correlation).

VII. Conclusion

In this work we analyzed the relationship between questions' explanatory value (considering both NL text and code snippets as information sources) and their answering time. We designed a procedure for assessing the explanatory value of questions regarding accompanying code, and we tested it by correlating these values with answering times. The chosen method, called L-ROUGE, involves the calculation of the longest common subsequence between the code and the text.

Our results show that the predictors utilizing the L-ROUGE algorithm provide a statistically significant improvement over the baseline value. These support the use of the explanatory value as a predictor for the answering time of a question.

We also investigated Java term parsers in this context, where the outputted list of terms from code snippets replaces the code snippets themselves for the correlation algorithm. We verified whether these parsers provide any improvement for the answering time. Our results show that, as the answering time reached a month, this term strategy performed significantly worse than simply using the code snippet itself for correlation.

The main contribution of this paper is an initial "guideline" to lower the answering time of a post. The guideline entails that the L-ROUGE score, as implemented in this paper, of an explanation in natural language and a corresponding code snippet should be larger than or equal to 0.2. This guideline can be implemented as a real-time 'edit suggestion'.

References

[1] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 25–34.

[2] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 832–841.

[3] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, 2004, pp. 74–81.

[4] A. T. T. Ying, "Mining challenge 2015: Comparing and combining different information sources on the stack overflow data set," in *Proceedings of MSR 2015*, 2015, p. to appear.

[5] T. J. Parr and R. W. Quong, "Antlr: A predicated-ll (k) parser generator," *Software: Practice and Experience*, vol. 25, no. 7, pp. 789–810, 1995.