NAME: MOHAMMAD REZA ARDESTANI DATE: DEC 3, 2022

What Makes a Good Commit Message?

Yingchen Tian, Yuxia Zhang, Klass Stol, Lin Jiang, Hui Liu

PROBLEM(S) ADDRESSED

The outline of research questions (RQs) is written as a question in the paper title. Later in the paper, it has been decomposed into three questions. Firstly, what is the degree to which mal-written commit messages exist? Secondly, what are common patterns across well-written commit messages? Thirdly, can we train a classifier system to automatically discriminate between good and low-quality commit messages? These three questions, throughout the paper, will be broken down into sub-questions elaborated in the Proposed Solution section.

MOTIVATION

Based on an examination of previous studies, there is no work on defining what makes a commit "good" and no work on analyzing the quality of commit messages. Having found a high level of understanding of what makes a "good" commit, we need a lower level of understanding by extracting recurrent patterns. Unlike current state-of-the-art (SOTA) methods, this paper, by incorporating semantic and thematic analysis, does not rely only on the syntax of messages. Finally, building a commit classifier, answering RQ3, which can distinguish between "good" and "poor" commit messages would be helpful for both developers, who write messages, and researchers who construct high-quality commit-message datasets. Moreover, previous studies are using uncurated collected and superficially preprocessed datasets for their automatic commit generator systems. They, however, can revivify their system and rectify their result by using this paper's method for reaching high-quality datasets.

PROPOSED SOLUTION

A data collection and preprocessing step is carried out at the beginning. The authors chose five active and popular java projects from GitHub amongst 100 initially selected projects which were sorted by their "star" and prioritized if they had previously been used in automatic commit message generation studies. Spring-boot, Retrofit, Okhttp, Junit4, and Dubbo are the final selected projects for studying their messages. Since this paper focus on English and manually written messages automatically generated messages are filtered out leaving us with a collection of ca. 30K messages.

Next, the authors use the aggregated dataset to answer RQs. Regarding the first RQ, there is no consensus upon a metric of high-quality messages. To construct this metric, authors reviewed 46 germane papers, looked in on the top 50 pertinent online records from OSS, and asked for the opinions of 30 experienced OSS developers on it. Finally, the authors concluded that a message should recap the change (noted as What) and the reason for the change (noted as Why). Depending on whether these two key components exist in a commit, commits get classified into four categories. Authors did not classify messages as "Not What" or "No Why" in cases where "what" or "why" are easy to infer, common sense, or trivial. Moreover, they have decided to accept links to an issue report or pull request as a valid way of providing "why" information. To find the distribution of the four aforementioned categories, clustered random sampling is used and ca. 5% of initial commit messages (95% confidence, 5% margin of error) is selected. Then, two authors independently and manually labeled the selected commits with 0.91 Kappa (Cohen's Kappa coefficient of agreement) and with 85% accuracy when their result was validated with experienced developers.

A thematic approach is adopted for identifying the attributes of well-written messages, having both "why" and "what", to address RQ2. The analysis is summarized as follows. (1) read and examined 252 well-written commits, (2) generated initial codes organized in a systematic way, (3) categorized codes with similar meanings and identified initial themes, (4) Merged similar themes or added sub-themes, and (5) came up with the final set of themes and resolved any conflicts between two independent examiners. During the above-mentioned analysis, authors noticed the effect of types of maintenance activities on the ways developers articulate "why' and "what" in commits; hence, the authors added another layer of classification upon themes. This layer considers if a commit is corrective, adaptive, or perfective. Lastly, two authors, who independently investigated commits, reached a 0.92 Kappa coefficient (high level of consistency).

Regarding the third RQ, authors trained two separate models, one for identifying whether a message contains "why" and one for "what", which they can later combine to classify a message into four different classes we have. Before training and testing, they have data preparation (cleaning, making dataset balance, and vectorization) and model selection. To clean commits, we modify all tokens that are not "natural language". URLs, for example, are replaced by "<X url>", where x indicates the type of the URL, or newlines are replaced with "<enter>". To redress the balance of the imbalance dataset authors have tried three widely used over-sampling methods including random sampling with replacement, synthetic minority oversampling, and the adaptive synthetic to prepare the data and selected the method that rendered hider accuracy in each classifier. As the last step of preprocessing, they vectorized the messages and then embedded them with Bidirectional Encoder Representations from Transformers (BERT). Finally, eight models are selected for modeling.

Evaluation & Results

After manually classifying 5% of the commits dataset, authors found the distribution of commit categories in the five selected projects and they also further classified "Neither Why nor What" into five different themes. All projects, except Retrofit, have more commits of type "Why and What" than any other type, and this type range in ratio from 42.2% to 82.3%. The average ratio of the "Neither Why nor What" type is ca. 4% with the following themes: single-word messages, submit-centered messages, scope-centered messages, redundant messaged, and irrelevant messages.

Authors addressed the second RQ with their thematic analysis outlined in the previous section. Five main themes were found for explaining "why" and four themes for describing "what". Five main themes of "why" are: Describe Issue (directly explaining the motivation of a code change), Illustrate Requirement (the origin of requirements that demanded a change), Describe Objective (mentioning the purpose of a change), Imply Necessity, and Missing Why (not providing any reason, e.g. when it is easy to infer). Four main themes of "what" are: Illustrate Function (explaining from a functional perspective), Describe Implementation Principle, Missing What, and Summarize Code Object Change. As mentioned previously, authors also investigated the commit distributions with respect to different maintenance types. From "why" themes, Describe Issue, Imply Necessity, and Missing Why are the most frequent themes in Corrective, Adaptive, and Perfective maintenance types respectively. From "what" themes, summarize code object Change is the most frequent amongst all maintenance types.

To build a commit message classifier, authors enlisted the help of eight common classification technics (Bi-LSTM, LSTM, Logistic Regression, MLP, Random Forest, Gradient Boosting, KNN, Decision Tree). Then, they used ten-fold cross-validation to validate the performance of eight methods and report the average accuracy of classifiers across ten folds. As mentioned before, two different types of classifiers are trained; C-Why (whether "why" is mentioned) and C-What. These two classifiers later get combined and construct C-Good (whether both "what" and "why" are mentioned). The average accuracy of Bi-LSTM on C-Why, C-What, and C-Good are 84.7%, 91.0%, and 75.9% respectively. This method outperforms all other methods.

Lastly, authors mentioned the values they added to the community and threats to the validity of their research. Developers can benefit from articulated taxonomy and the built classifiers by using them to write better commits and researchers can use the tools to collect high-quality commit messages for their datasets. There are some concerns about the validity of the results, namely, uncertainty about whether all automatic commits are filtered out, the subjective process of commit labeling, the relatively small dataset, a limited number of projects studied, and not using all the potential of format-related metrics in commits.

Reflections on Learning

- As mentioned in this paper, developers tend to avoid writing "why" in their commits. I realized I am one of them. This can definitely help me in writing better commits next time, thereby communicating effectively with other developers of the team.
- I also added Cohen's Kappa coefficient technic to my collection and keep that in mind for similar research I will conduct.
- Three over sampling technics were mentioned in this paper. Two of them were new to me. I explored them to my satisfaction
- Result of the RQ1 was quite unexpected for me. I didn't know even top projects are writing commits poorly a considerable number of times.
- I had a misunderstanding of k-fold cross validation until I read this paper. I thought we have to report the best accuracy during k-fold validation. This paper, however, reports the average accuracy of the k-fold validation which is more error-tolerant.

Directions

- Their approach is prone to the overfitting problem on account of using too small dataset. One way to rectify this is by using self-supervised technics as a future direction. In this way, we can generate more labeled records.
- Alongside the previous idea for automatically generating more labeled data, we can use semi-supervised technics, namely, "pre-train and fine-tuning" technic. For interested researchers, perhaps the authors, this website[1] has a thorough coverage of this issue, when there is not enough labeled data.
- Writing is a complex cognitive task and no wonder that developers fail to write their commits well. We can aid this process by generating a few commit suggestions and then asking the developers to choose the best one, and perhaps modify it before pushing. It is in this manner that we can have both the power of machines and the accuracy of humans since developers get facilitated by an initial version of the commit and get encouraged to well formulate that at the next step. This hypothesis can be verified or rejected in future works.

[1] https://lilianweng.github.io/posts/2021-12-05-semi-supervised/