Revisiting Textual Feature of Bug-Triage Approach

Zexuan Li
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
lizx_17@sjtu.edu.cn

Hao Zhong
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
zhonghao@sjtu.edu.cn

***Abstract*—With the increase of software users, programmers use issue tracking systems to manage bug reports and researchers propose bug-triage approaches that assign bug reports to programmers. Programmers assign bug reports often according to their descriptions. Based on by this observation, prior approaches typically use classic natural language processing (NLP) to analyze bug reports. Although the technical choice is straightforward, the true effectiveness of this technical choice is largely unknown. Taking a state-of-the-art approach as an example, we explore the impact of textual features in bug triage. By enabling and disabling the textual features of this approach, we analyze their impacts on assigning thousands bug reports from six widely used open source projects. Our result shows that instead of improving it, textual features in fact reduce the effectiveness. In particular, after we turn off its textual features, the f-scores of the baseline approach are improved by 8%. After manual inspection, we find two reasons to explain our result: (1) classic NLP techniques are insufficient to analyze bug reports, because they are not pure natural language texts and contain other elements (*e.g.*, code samples); and (2) some bug reports are poorly written. Our findings reveal a strong need and sufficient opportunities to explore more advanced techniques to handle these complicated elements in bug reports.**

I. Introduction

To support the collaboration of software teams, issue trackers (*e.g.*, JIRA [1]) have been widely used in both open source communities and commercial companies. With issue trackers, programmers and users can file and assign their found bugs. Even with issue trackers, it is still a time-consuming task to assign bug reports to programmers for fixing, since many bug reports have to be assigned [10] and some projects have many programmers [22]. In practice, even experienced programmers can disagree with some of assignments, and 25% bug reports of Eclipse are reassigned at least once [13], [20].

To handle the problem, researchers [9], [10], [15], [21], [12], [22], [8] have proposed various bug-triage approaches that automatically assign bug reports to developers, and these approaches typically reduce the bug triage problem to a classification problem. When training such classification models, the features from bug reports are inputs, and the assignees are outputs. In particular, most prior approaches focus on textural features (*e.g.*, bug descriptions) from bug reports. Although textural features shall be useful, their true effectiveness is not supported by empirical evidences. In this paper, we conduct the first empirical study to analyze the impact of textual features on a state-of-the-art approach [16]. We select this approach, because it has been evaluated on thousands of real bug reports from commercial companies and its results are quite positive. We turn off its textual features to explore the following research questions:

RQ1: What is the effectiveness of the baseline with and without textual features? The answer explores how effectively a typical bug-triage approach uses textual features.

RQ2: What's the effectiveness when textual features are combined? The answer explains how the features built up final effectiveness of the typical bug-triage approach.

RQ3: Why the effectiveness was improved, when textual features were removed? The answer explains why it is difficult to extract useful information from textual features.

II. Dataset

Table I shows our dataset. We collected those bug reports from five Apache projects in different sizes (*HBASE* [2], *CASSANDRA* [3], *SHIRO* [4], *PDFBOX* [5] and *LUCENE* [6]) as our subjects. To remove superficial bug reports, we select bug reports that are marked as "Closed" and "Fixed", but remove bug reports whose assignees or reporters are not human beings (*e.g.*, Adobe JIRA). In addition, we remove bug reports that were assigned to their reporters.

For large projects as *CASSANDRA* and *HBASE*, we select their latest 10,000 issues, and for the other projects, we extract all their issues. For each bug report, our tool extracts its details, title, description, reporter, and assignee, for latter analysis.

III. Baseline Approach

In this study, we select Jonsson *et al.* [16] as our baseline, because it is a recent approach that achieves promising results in the industrial contexts. They selected Logistic regression model as the top classifier and combined the underlying classifiers such as BayesNet, SMO, IBk, KStart and Random Forest. As they do not release their tool, we follow their steps, and build our own version also on WEKA [14]. As a minor difference, Jonsson *et al.* treat the title and the description as a single textual feature, but we handle them as two to explore their individual impacts.

IV. Early Result

We use our tool to assign the bug reports in our dataset with and without textural features. We use 10-fold cross-validation to ensure the reliability of our results on the following RQs.

RQ1: What is the effectiveness of the baseline with and without textual features? Table II shows the effectiveness with and without textual features (baseline vs no texts). We

TABLE I: Our dataset

Project	Issues	Bug reports	Developers
CASSANDRA	10,000	2,290	434
HBASE	10,000	2,265	347
LUCENE	8,653	1,923	70
PDFBOX	4,441	2,016	16
SHIRO	659	256	7
Total	33,753	8,750	863

TABLE II: The effectiveness (baseline vs no texts)

Model	**Setting**	**Precision**	**Recall**	**F-score**	**AUC**
BayesNet	Baseline	0.533	0.485	0.508	0.941
	No texts	**0.669**	**0.669**	**0.669**	**0.984**
SMO	Baseline	0.768	0.701	0.733	0.987
	No texts	0.737	**0.730**	0.733	0.987
IBk	Baseline	0.537	0.466	0.499	0.725
	No texts	**0.615**	**0.607**	**0.611**	**0.887**
KStar	Baseline	0.021	0.021	0.021	0.512
	No texts	**0.626**	**0.618**	**0.622**	**0.981**
RandomForest	Baseline	0.317	0.289	0.302	0.843
	No texts	**0.681**	**0.671**	**0.676**	**0.987**
Stacking	Baseline	0.891	0.673	0.767	0.985
	No texts	**0.953**	**0.725**	**0.824**	**0.986**

find that except for SMO, the models obtain better results without textual features. For their meta-classifier, after we disable textual features, we improve its f-score by 8%. In summary, removing textual features improves the effectiveness.

RQ2: What is the effectiveness when textual features are combined? Table III shows the results of comparing the combinations of textual information and nominal features reflecting different meanings. We find that the AUC with textual features only (0.667) is lower than the combination of nominal features (0.853). The effectiveness is remarkably improved by combining nominal features. We try to replace the nominal features with the best combination of textural contents, and the results show that all the replacements lead to lower AUC values. In summary, the combinations of the textual features do not improves its effectiveness either.

RQ3: Why the effectiveness was improved, when textual features were removed? After some inspections and discussions, we come to two explanations:

1. The classic textural analysis techniques are insufficient to handle bug reports. Bug reports are the combination of natural languages and structural contents, *e.g.* code samples, stack traces, and patches. However, all the existing approach use natural language processing techniques to handle bug reports, including Vector Space Model (VSM) and TF-IDF *et al.*. These techniques do not understand the true meanings of structural contents. As a result, these techniques overestimate the importance of code elements and such structural contents can become noises and even harm the effectiveness.

2. Some bug reports have no useful textural descriptions to determine assignments. For example, LUCENE-4689 [7] has a description "Just updating the eclipse project name from lucene_solr_branch_4x to lucene_solr_4_1 on the new branch." They introduce version change without informative natural language texts to explain the symptoms and reasons. Even experienced programmers also have to think over the reason behind and make a careful decision. As a result, it is challenging to determine who shall fix the bug.

TABLE III: The combination of features

Items	Features	AUC
Textual features	Summary	0.539
	Description	0.554
	Comment	0.645
	Summary + Description	0.595
	Summary + Comment	0.667
	Description + Comment	0.661
	Summary + Description + Comment	0.661
Textual & Nominal features	Summary + Description + Component + Version	0.652
	Summary + Description + Commenter + Reporter	0.778
Nominal features	Commenter + Reporter + Version + Component	0.853

commenter: people who made comments; component indicates which component is buggy; version denotes the releases of the buggy code

In summary, the inspection leads to two conclusions: some bug reports are poorly written without informative natural language texts, while well-written bug reports contain structural contents which call for more advanced techniques.

Discussion: Although we find that textual features do not have positive contributions to the state-of-the-art bug assignment approach, we disagree the superficial conclusion that researchers shall not use NLP in their tools. In contrast, there are a strong need and many opportunities to explore better techniques to handle natural language texts in software engineering. As NLP techniques are proposed to handle pure natural language texts, it is not strange that these techniques cannot achieve good results to handle software engineering documents. In other research topics, when researchers analyze software engineering documents [24], [11], [23], they propose techniques to classify code fragments and textual contents. After the contents of bug reports are classified, it is feasible to explore more effective techniques for bug triage. In addition, removing obsolete bug reports [19] can also improves the effectiveness of textual features.

More details of our study are listed on our project website: https://github.com/lizx2017/textMyth

V. Work Plan

Our plan to extend this work is as follows:

1. Identifying the key features of bug triage. While this study show that textual features are not effectively used, we will use feature selection techniques (*e.g.*, [17]) to explore which features were effectively used by the prior approaches.

2. Exploring more advanced techniques to handle textual contents. We plan to explore the effectiveness of textual features with more advanced techniques. For example, we will explore deep learning techniques [18] for their effectiveness.

3. Exploring the impacts on individual projects. Our study explores our research questions on a large dataset, but the impacts on a specific project can be overwhelmed by other projects. We plan to conduct our study on individual projects to explore impacts on specific projects.

Acknowledgment

We appreciate the anonymous reviewers for their insightful comments. Hao Zhong is the corresponding author. This work is sponsored by the National Key R&D Program of China No.2018YFC083050.

References

[1] https://issues.apache.org/jira.

[2] https://issues.apache.org/jira/projects/HBASE.

[3] https://issues.apache.org/jira/projects/CASSANDRA.

[4] https://issues.apache.org/jira/projects/SHIRO.

[5] https://issues.apache.org/jira/projects/PDFBOX.

[6] https://issues.apache.org/jira/projects/LUCENE.

[7] https://issues.apache.org/jira/browse/LUCENE-4689.

[8] S. N. Ahsan, J. Ferzund, and F. Wotawa. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *Proc. ICSEA*, pages 216–221, 2009.

[9] J. Anvik. Automating bug report assignment. In *Proc. ICSE*, pages 937–940, 2006.

[10] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proc. ICSE*, pages 361–370, 2006.

[11] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza. Content classification of development emails. In *Proc. ICSE*, pages 375–385, 2012.

[12] P. Bhattacharya and I. Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *Proc. ICSM*, pages 1–10, 2010.

[13] P. Bhattacharya, I. Neamtiu, and C. R. Shelton. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *The Journal of Systems & Software*, 85(10):2275–2292, 2012.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[15] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proc. ESEC/FSE*, pages 111–120, 2009.

[16] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21(4):1533–1578, 2016.

[17] R. Kohavi and G. H. John. Wrappers for feature subset selection. *AI*, 97(1-2):273–324, 1997.

[18] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong. Applying deep learning based automatic bug triager to industrial projects. In *Proc. ESEC/FSE*, pages 926–931. ACM, 2017.

[19] Z. Li and H. Zhong. An empirical study on obsolete issue reports. In *Proc. ASE*, page to eappear, 2021.

[20] G. Murphy and D. Cubranic. Automatic bug triage using text categorization. In *Proc. SEKE*, 2004.

[21] M. M. Rahman, G. Ruhe, and T. Zimmermann. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In *Proc. ESEM*, pages 439–442, 2009.

[22] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2016.

[23] F. Zampetti, A. Serebrenik, and M. Di Penta. Automatically learning patterns for self-admitted technical debt removal. In *Proc. SANER*, pages 355–366, 2020.

[24] H. Zhong and Z. Su. Detecting API documentation errors. In *Proc. OOPSLA*, pages 803–816, 2013.