Review on Leveraging Techniques on Bug Repository to form Accurate Bug Triage

Aparna S. Murtadak, Prof. S. R. Durugkar

Abstract— Software organizations spend huge amount of cost on managing programming bugs. An unavoidable stride of fixing bugs is bug triage, which expects to effectively allocate a developer to a new bug. To diminish the time cost in manual work, text classification techniques are applied to perform automatic bug triage. In this paper, we address the problem of information decrease for bug triage, i.e., how to diminish the scale and enhance the nature of bug data. We use instance selection with feature selection at the same time to decrease information scale on the bug dimension and the word dimension. To focus the request of applying instance selection and feature selection, we extract properties from historical bug information sets and construct a predictive model for new bug information set. Outcomes demonstrate that our data reduction can adequately decrease the data scale and enhance the precision of bug triage. Our work gives a way to deal with leveraging techniques on data processing to form decreased and high-quality bug information in programming advancement and upkeep.

Index Terms—Bug data reduction, Bug triage, Data Mining, Feature selection, Instance selection

1 INTRODUCTION

Bug fixing is a significant and lengthy process in software maintenance. For a major software project, the number of daily bugs is so large that it is not possible to handle them without delaying. In recent software development, software repositories are extensive databases for storing the outcomes of software development. Conventional software analysis is not completely appropriate for the large-scale and complex information in software repositories. By using data mining techniques, mining software repositories can discover interesting information in programming repositories and solve real world programming problems. A bug repository (a typical software repository used for storing bug reports), plays an important role in handling software bugs. In a bug repository, a bug is retained as a bug report, which records the textual description of reproducing the bug and updates the status of bug fixing. The two difficult tasks related to bug data

that may affect the efficient use of bug repositories in programming development, that is the large scale and the low quality bug data. In this paper, the problem of data reduction for efficient bug triage that means how to reduce the scale and improve the quality of bug data is addressed.

- *Aparna S. Murtadak is currently pursuing masters degree program in Computer Engineering from S.N.D.C.O.E.R.C. college Yeola, Dist-Nasik, in Savitribai Phule University of Pune, Maharashtra, India E-mail: murtadakaparna@gmail.com*
- *Prof. S. R. Durugkar HOD at S.N.D.C.O.E.R.C. college Yeola, Dist-Nasik, in Savitribai Phule University of Pune, Maharashtra, India E-mail: santoshdurugkar@gmail.com*

2 LITERATURE SURVEY

Bug repositories are widely used for handling software bugs. A time-consuming stride of handling software bugs is bug triage, which expects to assign a correct developer to fix a new bug [1]. Because of the substantial number of every day bugs and the absence of skill of the considerable number of bugs, manual bug triage is costly in time cost and low in exactness. Cubranic and Murphy first propose the problem of automatic bug triage to diminish the cost of manual bug triage [3]. They apply text categorization method to forecast specific developer that should work on the bug based on the bug's explanation. Jeong et al find out that over 37 percent of bugs have been reassigned in manual bug triage [7]. They recommend a tossing graph method to diminish reassignment in bug triage.

Anvik et al examine various techniques on bug triage, including information preparation and classic classifiers [2]. Open source development projects commonly support an open bug storehouse to which both developers and clients can report bugs. The reports that show up in this repository must be triaged to figure out whether the report is one which requires consideration and if it is, which developer will be assigned the obligation of resolving the report. Extensive open source advancements are troubled by the rate at which new bug reports show up in the bug store. In paper displays a semi-automated approach expected to simplicity one piece of this procedure, the task of reports to a developer [2].

To examine the interrelations in bug data, Sandusky et al. create a bug report network to evaluate the interdependency between bug reports [14]. Also examine relationships among bug reports, Hong et al. construct a

developer public network to examine the association among developers based on the bug information in Mozilla project [6]. This developer public network is cooperative to understand the developer society and the project advancement. By drafting bug priorities to developers, Xuan et al. recognize the developer prioritization in open source bug repositories [9]. The developer prioritization can differentiate developers and aid tasks in software upkeep. To inspect the quality of bug information, Zimmermann et al. prepare questionnaires to developers and clients in three open source projects [23]. Based on the analysis of questionnaires, they distinguish what makes a good bug report and guide a classifier to recognize whether the quality of a bug report should be enhanced. Duplicate bug reports weaken the quality of bug information by delaying the cost of managing bugs.

Instance selection and feature selection are commonly used techniques in data processing. For a given data set, instance selection is to obtain a subset of relevant instances (i.e., bug reports in bug data) [4] while feature selection expects to obtain a subset of relevant features (i.e., words in bug data) [5]. In this paper join instance selection with feature selection at the same time to decrease information scale on the bug dimension and the word dimension.

3 PROPOSED SYSTEM

The objective of paper is to address the problem of data reduction for effective bug triage, i.e., how to reduce the bug information to save the work cost of developers and improve the quality to facilitate the process of bug triage. Data reduction for bug triage expects to build a small-scale and high-quality set of bug data by removing bug reports and words, which are not informative and redundant. In proposed system, we join existing techniques of instance selection and feature selection at the same time to decrease the bug dimension and the word dimension. The reduced bug data contain less bug reports and fewer words than the original bug data and also provides similar information over the original bug data. After that applying clustering technique on resulted reduce bug data set to generate different domain wise clusters. Text classification is used in bugs data set to classify all bugs into different classes. Proposed system is evaluating the reduced bug data by two criteria which are size of a data set and the correctness of bug triage.

3.1 Architecture of Proposed System

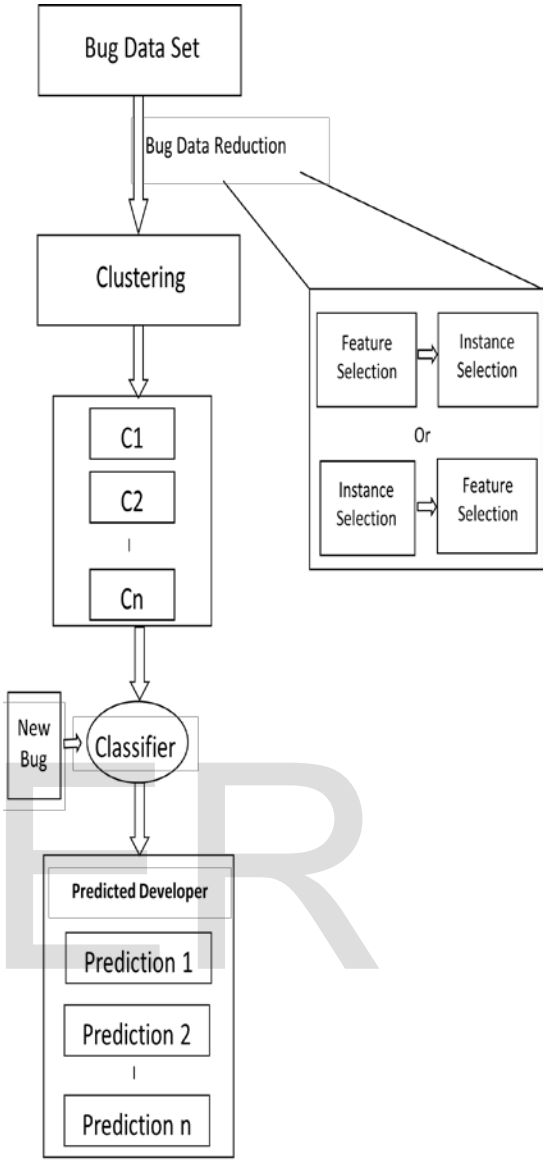

Fig. 1 Block diagram of system architecture

Figure shows illustration of reducing bug data for bug triage. In this figure input is bug data set. Bug data set views as a text matrix. Each row of text matrix indicates one bug report and each column of text matrix indicates one word. Instance selection and Feature selection techniques are applying on bug data set to reduce the scale of bug data. Instance selection technique is used to remove bug reports and Feature selection technique is used to remove non-informative words. After that applying clustering on resulted reduced bug data set to make different domain wise clusters. In next step Naïve Bayes classifier is used to make classification of bugs data set in to different classes. Finally bugs are assigned to specific developers.

3.2 Flow of Proposed System

1. Take input dataset
2. Apply Data reduction
3. Apply clustering
4. Extract Features-Stored into training set
5. New bug reported
6. Classifier will take two inputs, new bug and training set
7. Output of classifier is predicted set of developers

3.3 Feature Selection

Feature selection is a standard technology to decrease the features of huge data sets in machine learning. The number of variables (or features) gathered in a dataset is typically relatively large and some of these features are not informative or can't provide high differentiating power. The objective of feature selection algorithm is to remove the irrelevant and redundant words from the selected dataset, thus optimising the performance of the classification and/or clustering algorithms. In addition, for a particular dataset, feature selection can aid to realize which features are important as well as how they are associated. Feature selection can be defined as the procedure of choosing a smallest subset of m features from the original dataset of n features (m is less than n), so that the feature space (i.e. the dimensionality) is optimally reduced according to the evaluation criteria: The classification accuracy does not significantly reduce and The resulting class distribution, given only the values for the selected features, is as similar as possible to the original. A feature selection algorithm generally consists of four steps which are subset generation, subset evaluation, stopping criterion, and result validation. Subset generation is a search procedure which creates subsets of features for evaluation. Each subset generated is verified by some particular evaluation criterion and evaluated with the earlier best one with respect to this criterion. If a new subset is found to be better, then the earlier best subset is replaced with the new subset. In this paper, Feature selection technique is used to remove the words in bug reports which are redundant and non-informative.

3.4 Instance Selection:

As data increases the requirement for data reduction also increases for effective data mining. Instance selection is one of effective means to data reduction. The objective of Instance Selection algorithm is to diminish the size of a dataset while still sustaining the integrity of the actual dataset. In many cases, generalization correctness can raise when noisy instances are eliminated and when decision borders are smoothed to more closely equal the true core function. These instances can also be considered as outliers (or bad data). Specifically, outliers are those data points which are extremely unlikely to arise given a model of the data. In this paper, Instance selection technique is used to reduce the number of instances means bug reports by removing noisy and redundant bug reports. Due to removing a set of instances from a dataset the response time of classification decisions decreases, as some instances are examined when a query instance is presented. This purpose is primary when working with huge database and has limited storage.

4 Conclusion

Bug triage is a costly stride of programming upkeep in both work cost and time cost. In this paper, we join feature selection with instance selection to decrease the size of bug data sets and also enhance the data quality. To decide the request of applying instance selection and feature selection for a new bug data set, we extract attributes of each bug information set and prepare a predictive model based on recorded information sets. Our work gives a way to deal with utilizing systems on information handling to shape reduced and high-quality bug information in programming improvement and support.

References

[1] J. Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu," Towards Effective Bug Triage with Software Data Reduction Techniques" *IEEE transactions on knowledge and data engineering, vol. 27, no. 1, january 2015.*

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" *in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.*

[3] D. _Cubrani_c and G. C. Murphy, "Automatic bug triage using text categorization," in Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng., Jun. 2004, pp. 92–97.

[4] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," Knowl. Inform. Syst., vol. 35, no. 2, pp. 249–283, 2013.

[5] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," J. Mach. Learn. Res., vol. 3, pp. 1157–1182, 2003.

[6] Q. Hong, S. Kim, S. C. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in Proc. 27th IEEE Int. Conf. Softw. Maintenance, Sep. 2011, pp. 323–332.

[7] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with tossing graphs," in Proc. Joint Meeting 12th Eur. Softw. Eng. Conf. 17th ACM SIGSOFT Symp. Found. Softw. Eng., Aug. 2009, pp. 111–120.

[8] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in Proc. 34th Int. Conf. Softw. Eng., 2012, pp. 25–35.

[9] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36, no. 4, pp. 474–494, Jul./Aug. 2010.

[10] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," ACM Trans. Soft. Eng.

Methodol., vol. 20, no. 3, article 10, Aug. 2011.
[11] C. C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1 21, 2013.
[12] K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, Aug. 2006, pp. 43–50.
[13] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," IEEE Trans. Knowl. Data Eng., vol. 24, no. 6, pp. 1146–1150, Jun. 2012.
[14] R. J. Sandusky, L. Gasser, and G. Ripoche, "Bug report networks: Varieties, strategies, and impacts in an F/OSS development community," in Proc. 1st Intl. Workshop Mining Softw. Repositories, May 2004, pp. 80–84.
[15] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.
[16] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in Proc. ACM Conf. Comput. Supported Cooperative Work, Feb. 2010, pp. 301–310.
[17] V. Bol_on-Canedo, N. S_anchez-Maro~no, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," Knowl. Inform. Syst., vol. 34, no. 3, pp. 483–519, 2013.
[18] V. Cerver_on and F. J. Ferri, "Another move toward the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule," IEEE Trans. Syst., Man, Cybern., Part B, Cybern., vol. 31, no. 3, pp. 408–413, Jun. 2001.
[19] B. Fitzgerald, "The transformation of open source software," MIS Quart., vol. 30, no. 3, pp. 587–598, Sep. 2006.
[20] M. Grochowski and N. Jankowski, "Comparison of instance selection algorithms ii, results and comments," in Proc. 7th Int. Conf. Artif. Intell. Softw. Comput., Jun. 2004, pp. 580–585.
[21] A. E. Hassan, "The road ahead for mining software repositories," in Proc. Front. Softw. Maintenance, Sep. 2008, pp. 48–57.
[22] S. Kim, K. Pan, E. J. Whitehead, Jr., "Memories of bug fixes," in Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng., 2006, pp. 35–45.
[23] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schr€oter, and C. Weiss, "What makes a good bug report?" IEEE Trans. Softw. Eng., vol. 36, no. 5, pp. 618–643, Oct. 2010.
[24] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng., May 2010, pp. 481–490.
[25] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in Proc. 7th IEEE Working Conf. Mining Softw. Repositories, May 2010, pp. 1–10.