IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A METHOD OF COMBINING DATA REDUCTION TECHNIQUES FOR EFFECTIVE BUG TRIAGE

Aparna S. Murtadak
Department of Computer Engg, S.N.D.C.O.E.R.C. Yeola, Dist-Nasik,
Savitribai Phule University of Pune, Maharashtra, India

DOI: 10.5281/zenodo.268686

ABSTRACT

In current scenario, software development is having challenge of handling bugs and optimizing software. Daily vast amount of bug information is generated and dealing with these software bugs is an important part in software industry to keep software upgraded. Programming associations spend enormous measure of expense on managing programming bugs as it is an inevitable step of settling bugs, which aims to adequately allot a developer to a new bug. Manual work constitutes huge time cost so to lessen this, text classification techniques are used to perform automatic bug triage. For popular programming frameworks, the number of day by day submitted bug reports is high. Triaging these arriving reports is not only time consuming but also monotonous assignment. Bug triage by using software data reduction techniques means reduction of bug data set by keeping the originality and chooses appropriate developer for a bug to fix it. Hence for these purpose two software data lessening techniques Instance selection and Feature selection are used. Basically Instance selection technique is used to eliminate bug reports which contain similar information and Feature selection technique is used to remove non-informative words from the bug data set

KEYWORDS: Bug, Bug data reduction, Bug triage, Feature selection, Instance selection, Prediction for reduction order

INTRODUCTION

Bug is a common term used for the Error, Fault, Failure or mistake occurs in computer program or system. Due to the complexity of software development, bugs are unavoidable and it is necessary to resolve all the bugs. Bug resolution, which means the diagnosis, fixing, testing, and documentation of bugs, is a significant activity in software development and maintenance. A time-consuming step of handling programming bugs is bug triage, which expects to assign a right developer to fix a bug. In traditional software development, new bugs are manually triaged by a developer i.e. a human triager. Because of the huge number of daily bugs and the absence of expertise of all the bugs, manual bug triage is expensive in time cost and low in accuracy.
In bug repository bug is maintained as bug report which contains textual description about details of a bug. A bug repository is an important database in advanced programming development. A bug repository (a regular programming storehouse, for storing the details of bugs) plays an essential part in managing programming bugs. Programming bugs are unavoidable and fixing bugs is costly in programming improvement. Programming organizations spend huge amount of expense in altering bugs. Bug triage is an essential step for bug fixing, means to assign a bug to a related developer for further handling but it is wide procedure. One of the important reasons why bug triaging is such an extensive procedure is the difficulty in determination of the most skillful developer for the bug kind. But for effective bug triage Data Reduction is necessary. There are two difficulties identified with bug information that may influence the effective use of bug repositories in programming development tasks, to be specific the vast scale and the low quality of bug data. So in this paper we work on Data reduction for effective Bug Triage i.e. how to reduce scale and improve quality of bug data.

LITERATURE SURVEY

Bug repositories are generally used for handling software bugs. A lengthy stride of handling software bugs is bug triage, which expects to allocate a right developer to fix a new bug [1]. Because of the significant number of consistently bugs and the lack of skill of the considerable number of bugs, manual bug triage is expensive in time cost and low in terms of accuracy. Cubranic and Murphy first recommend the issue of automatic bug triage to lessen the expense of manual bug triage [3]. They apply text categorization method to predict specific developer to handle the bug based on the bugs details. Jeong et al. discover that more than 37 percent of bugs have been reassigned in manual bug triage [7]. They recommend a tossing graph process to lessen reassignment in bug triage. To analyze the interrelations in bug data, Sandusky et al. build a bug report network to assess the interdependency between bug reports [10]. Also examine connections among bug reports, Hong et al. build a developer public network to analyze the relationship among developers in view of the bug data in Mozilla project [6]. This developer public network is helpful to understand the developer society and the project advancement.

Just et al. separated the reactions from the same overview to recommend enhancements to make bug following frameworks less demanding to utilize and encourage accommodation of better quality bug reports [15]. In conclusion, another study by Bettenburg et al. competed for converging of copy bug reports alongside the firsts to make more extraordinary data about the bug accessible to designers [13]. Conversely, our work concentrates on the collaboration in the middle of developers and clients with the objective of enhancing apparatus support for this communication. Sandusky and Gasser concentrated on the part of arrangement in programming issue administration and how it influences the association of data [10]. In existing work on bug repository engineers are constantly treated similarly. Be that as it may, the need of a designer assumes a critical part in the errands. For instance, a dynamic designer might make a larger number of commitments on bug settling than a dormant one; an accomplished analyzer might discover bugs with higher severities than a typical end client [14]. In this paper, we rank every one of the engineers of bug repository to help the undertakings around bug repository. We indicate the procedure of creating the engineer needs as developer prioritization. For bug information, a few different tasks exist once bugs are triaged. For e.g. severity identification which aims to distinguish the significance of bug reports for further planning in bug handling [17].

PROPOSED SYSTEM

The objective of this paper is to address the issue of data reduction for effective bug triage. Data reduction for bug triage hopes to construct a small scale and high quality bug data by removing bug reports and words, which are not enlightening and repetitive in nature [11]. Instance selection and Feature selection techniques are used for bug data reduction to make reduced bug data set. Instance selection removes bug reports which are duplicates, it reduces the data scale and by removing uninformative words feature selection improves the accuracy of bug triage. Proposed framework is assessing the reduced bug information by two criteria which are size of an information set and the accuracy of bug triage.

Figure indicates the system architecture of the proposed system. It indicates description of bug data reduction for effective bug triage. The input to the system is in the form of bug data set. Bug data set contains all the details of bugs and each bug has bug report that includes details about bug. For implementation of proposed system bug data set of Eclipse which is large open source project is being used. In figure first step is applying predicted reduction order of Instance selection and Feature selection to the data set for data reduction. Second step is extraction of attributes from bug data set and training classifier. Then For more accuracy Agglomerative clustering technique is used to generate different clusters before classification. Then classification performs on a cluster which new bug is belonging. For better bug triage novel hybrid approach of Naive Bayes and KNN is used in our proposed system instead of single classifier. Output of proposed system is in the form of predicted list of developers to a bug.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

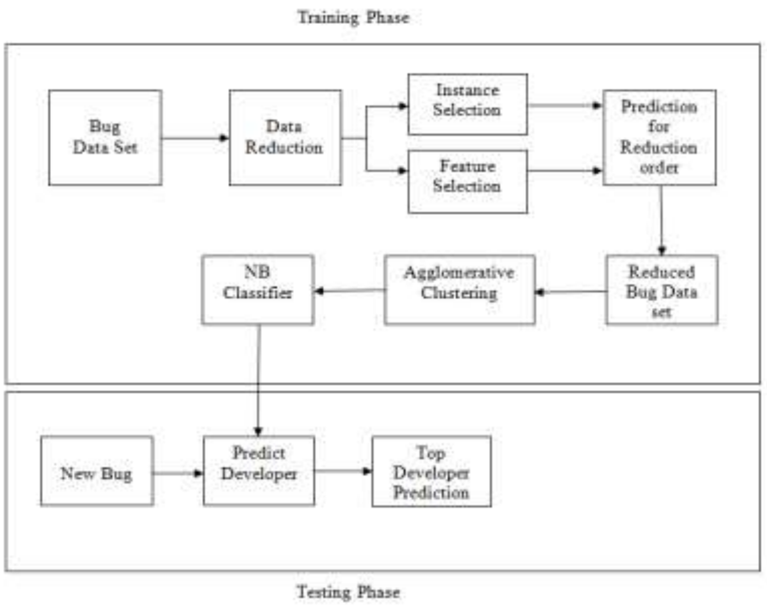

Fig 1: System Architecture

IMPLEMENTATION DETAILS

A. Instance Selection Algorithm:

Input: Bug Data set

Output: The set of selected bug reports

Processing steps:

```
Step 1: Read All Bug Reports.
Step 2: Calculate total unique words (U) in all bug reports
Step 3: Calculate TF value for each word for each document
Step 4: Create feature vector of each bug report as fd1, fd2..... fdn.
Step 5: Apply Pearson correlation score calculation
            for(i=fd1   to    fdn)
{
                     for(j=i+1 to fdn)
{
                     Calculate distance using Pearson correlation score calculation
}
Step 5.1: Apply KNN to find top similar document to current document as set (ks)
Step 5.2: Remove all ks set from corpus
}
Step 6: Repeat step 5 for all Bug reports.
```

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

B. Feature Selection Algorithm:

Input: Bug Data set

Output: The set of selected words

Processing steps:

```
Step 1: Read all bug reports from input bug data set.
Step 2: Calculate unique words array U.
Step 3: Calculate term frequency of each unique word for every bug report.
Step 4: Re-rank all words in U.
Step 5: Calculate zero count for each word in all reports.
Step 6: Remove all words having information gain less than T.
            Where T is user define threshold value.
Step 7: Calculate performance of Feature selection algorithm
Step 8: Display updated new bug data.
```

C. Naive Bayes Algorithm:

Input: Query Bug, Trained dataset

Output: Recommended Developers

Processing steps:

```
Step 1: Read training data set.
Step 2: Identify distinct labels in training data.
Step 3: Calculate prior probability of each label (y1, y2, y3..... yn) (Developer)
Prior probability=no of yn type labels/total no of labels;
Step 4: Read new bug report for label prediction as X
Step 5: Calculate likelihood of X for y1, y2, y3...yn
Likelihood of X to y1=no of y1 in vicinity of X/total noY1;
Step 6: Calculate Posterior probability of X being Y1
Posterior probability x=prior probability Y1*likelihood x to Y1;
Step 7: Repeat steps 5-6 for all labels y1, y2...yn.
Step 8: select label with highest probability.
```

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

RESULT ANALYSIS

In this paper, two data pre-processing techniques are used for data reduction such as Instance Selection and Feature Selection. To perform the experiments, I have set up the dataset for bug data reduction from open source project Eclipse and Mozilla. Below figure 2 and 3 shows result of Bug Triage accuracy with Instance selection and Feature selection on Eclipse and Mozilla bug data set in term of Precision, recall and F1 in comparison with Base paper result. And figure 4 shows result of Bug Triage accuracy in percentage with Data reduction and without data reduction

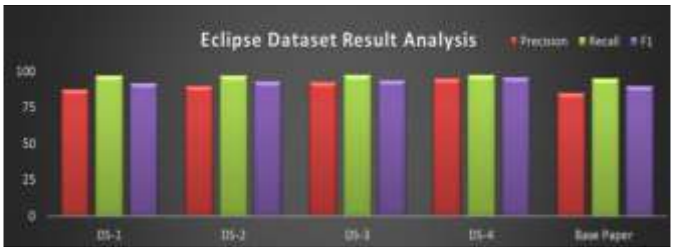

Fig 2: Result on Eclipse data set

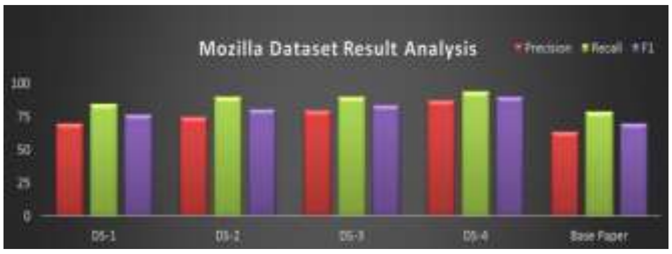

Fig 3: Result on Mozilla data set

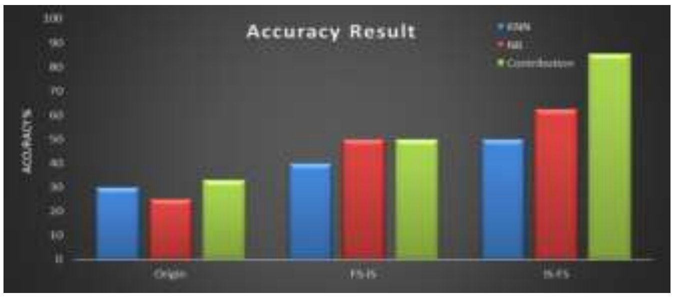

Fig 4: Accuracy of Bug Triage with Data reduction

CONCLUSION

In this paper, I have presented an approach to assignment of a bug report to a developer with the software data reduction technique Feature selection along with instance selection to enhance quality of bug data for Accurate Bug Triage. For better bug triage novel hybrid approach of Naive bayes and KNN is used in our proposed system instead of single classifier.

ACKNOWLEDGEMENT

I am very much thankful to all authors those are mentioned in the references. I would like to thanks my guide Prof. S. R. Durugkar for his constant support and motivation. Also I would like to thanks PG coordinator and HOD of computer department for their support in presenting this paper.

REFERENCES

[1] Jifeng 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. Cubranic 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] 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.
[10] 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.
[11] 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.
[12] 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.
[13] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" IEEE Trans. Softw. Eng., vol. 36, no. 5, pp. 618–643, Oct. 2010.
[14] S. Artzi, A. Kiezun, 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.
[15] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," in VL/HCC, pages 82–85, 2008.
[16] J. W. Park, M. W. Lee, J. Kim, S. W. Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems," in Proc. 25th Conf. Artif. Intell., Aug. 2011, pp. 139–144.
[17] 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.