www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issues 6 June 2016, Page No. 16800-16804

Bug Triage Using Dimensionality Reduction Technique And PSO Algorithm

S.Amritha[1], A.Jennifer Sagaya Rani[2]

[1]M.E. Student, Department of CSE, Parisutham Institute of Technology and Science, Tamil Nadu, India
[2]Asst. Professor, Department of CSE, Parisutham Institute of Technology and Science, Tamil Nadu, India
[1]amriajay@gmail.com,[2]jenusagay@gmail.com

***Abstract*—The process of fixing bug is bug triage that aims to properly assign a developer to a new bug. Software companies pay out most of their expenses in dealing with these bugs. To reduce time and cost of bug triaging, an automated approach is developed to predict a developer with relevant experience to solve the new coming report. In proposed approach data reduction is done on bug data set which will reduce the scale of the data as well as increase the quality of the data. Instance selection and feature selection is also used simultaneously with historical bug data. Previously, text classification techniques are applied to conduct bug triage. The problem here is to get quality bug data sets as they are of very huge in size. In the proposed system, the problems of reducing the size and to improve the quality of bug data are addressed.First, pre-processing is done to the remove unimportant attributes and to identify missing terms. Then instance selection is combined with feature selection by using Dimensionality reduction technique to simultaneously reduce data size on the bug dimension and the word dimension. By using PSO algorithm, the reduction order is determined using fitness value. It is used to produce quality bug data set. The results show that the proposed system can effectively reduce the data size and improve the accuracy of bug triage. The proposed system provides an approach to leveraging techniques on data processing to form reduced and high eminence bug data in software improvement and maintenance.**

Keywords: Bug, Bug triage, data reduction, Instance selection, Data Mining.

I. INTRODUCTION

A bug plays a vital task in handling software bugs. Various open source software projects attempt an open bug database that allows both developers and users to handle defects or problems in the software, providing possible improvements, and comment on existing bug reports.

For open source large-scale software projects, the number of daily bugs is so large which makes the triaging process very difficult and challenging [10]. Software companies spend over 45 percent of cost in fixing bugs .There are two challenges related to bug data that may affect the effective use of bug repositories in software development tasks, namely the large scale and the low quality. In a bug repository, a bug is maintained as a bug report, which records the textual description of reproducing the bug and updates according to the status of bug fixing [15].

Primary contribution of this paper is as follow:

- **Bug dimension:** Instance selection can remove unhelpful bug information, meanwhile, we can witness that the accuracy may be diminished by confiscating bug information.
- **Word dimension:** By removing unhelpful arguments, feature selection improves the accuracy of bug triage. This can recover the accuracy loss by instance selection.

In this paper, we proposed system which is used to simultaneously reduce the scales of the bug dimension and the word dimension and to improve the accuracy of bug triage. It proposes a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories. Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports, which are redundant or non-informative. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data using Dimensionality reduction. It extends the work on the generation of bug report summaries with a task-based study to evaluate the usefulness of bug report summaries in the context of a real software task that is bug report duplicate detection using Dimensionality reduction. PSO algorithm is used to calculate fitness value and predict the Efficient Retrieval result based on the fitness value. The proposed system will gives the bug report with accuracy.

II. ARCHITECTURE

1) Bug Triage

DOI: 10.18535/ijecs/v5i6.12

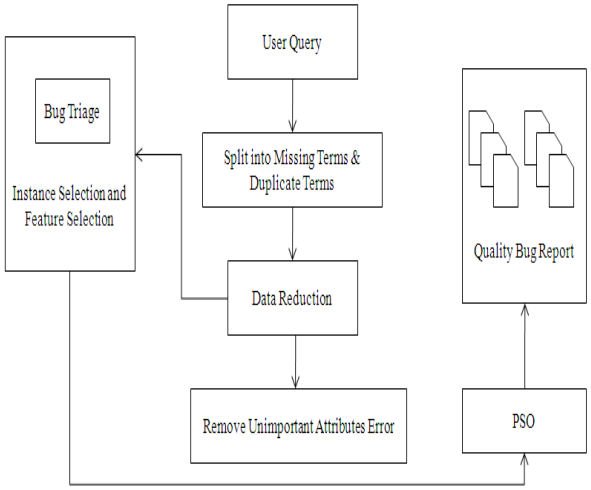

Fig. 1 Architecture of bug triaging process

Figure 1 shows the architecture of the bug triaging. Aim of bug triage is to allocate a developer for bug deception. Once a developer is allocated to a new bug report they will solve the bug or attempt to fix it. They will provide the status associated to bug whether it is corrected or not [1].

2) Data Reduction

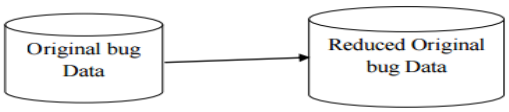

Fig. 2 Data Reduction

Figure 2 shows that the data is reduced from original bug data. Here we are decreasing the bug recordsby means of instance and feature selection so that we acquire low scale as well as eminence data. In our work, to protect the manual labor charge of developers, the data reduction for bug triage has two goals decreasing the data scale and enhancing the accuracy of bug triage.

In disparity to exhibiting the textual content of bug information in prevailing work (e.g., [1], [6], [9]), we target to augment the data set to construct a preprocessing methodology, which can be applied before an prevailing bug triage approach. We describe the two goals of data reduction as follows.

a) Decreasing the Data Scale

We reduce scales of data sets to protect the manual labor charge of developers.

Bug dimension. As mentioned above, the aim of bug triage is to allocate developers for bug deception. Once a developer is allocated to a new bug report, the developer can inspect historically fixed bugs to custom a solution to the existing bug report [6], [14]. For example, historical bugs are checkered to detect whether the new bug is the replica of a prevailing one [25]; moreover, existing results to bugs can be searched and applied to the new bug [12]. Thus, we consider reducing replica and raucous bug reports to reduce the number of historical bugs. In practice, the manual labor charge of developers (i.e., the cost of examining historical bugs) can be avoided by subsiding the number of bugs based on instance selection.

Word dimension. We use feature selection to remove noisy or replica words in a data set. Based on feature selection, the reduced data set can be controlled more easily by automatic methods (e.g., bug triage approaches) than the original data set. In addition to bug triage, the reduced data set can be further used for other software tasks after bug triage (e.g., severity identification, time prediction, and reopened bug analysis).

b) Enhancing the Accuracy

Accuracy is an essential evaluation measure for bug triage. In our work, data reduction discovers and removes noisy or duplicate information in data sets.

Bug dimension. Instance selection can remove unhelpful bug information; meanwhile, we can notice that the accuracy may be reduced by removing bug reports.

Word dimension. By removing unhelpful words, feature selection recovers the accuracy of bug triage.

III. METHODOLOGY AND MEASURES

1) Instance Selection

- Instance selection approaches related with data mining jobs such as classification and clustering It's a non-trivial procedure of finding usable, new, hypothetically valuable, and at last comprehensible patterns in Original bug Data Reduced
- Original bug selecting a subset of records to attain the original determination of a data mining application as if the entire records are used.
- The ultimate result of instance selection is model independent.

P (Ms) ≅ P (Mw)

Evaluation measures:

- **Direct Measure**
  - Keep as much similarity as possible amongst a selection of data and the original data.
  - Ex) Entropy, moments, and histograms.
- **Indirect Measure**
  - For example, a classifier can be used to checker whether instance selection outcomes in enhanced, equal, or worse analytical precision.
  - Conventional evaluation methods in sampling, classification, and clustering can be castoff in

evaluating the performance of instance selection.
- Ex) Precision, recall.

2) Feature Selection

- It chooses a smallest set of features such that the chance of distribution of dissimilar classes specified the values for those features is as close as possible to the original distribution given the values of all features [1].
- Decrease # of patterns in the patterns, understand at ease.
- Create new features that can capture the essential data in a records set much more proficiently than the original features.
- Use the least illustration which is adequate to resolve the task.

Heuristic methods:

- Step-wise forward selection
- Step-wise backward elimination
- Create new features that can detect the significant facts in a records set much more proficiently than the original features
- Three general methods:
- Feature extraction
- domain-specific
- Mapping data to new space

3) PSO algorithm

PSO is prepared with a set of random elements (solutions) and then pursuit for optimum by apprising generations. Elements move over the result space, and are appraised according to some fitness condition after each time period. In every repetition, each element is updated by resulting two "best" values. The first one is the best result (fitness) it has attained so far (the fitness value is also stored). This value is called elements best value. Another "best" value that is traced by the PSO is the best value attained so far by any element in the population. This second best value is a global best. When an element takes part of the population as its topological neighbors, the second best value is a local best. Neighborhood bests permit parallel consideration of the search space and reduce the vulnerability of PSO to subsiding into local minima.

v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[])…… (a)

present[] = persent[] + v[]…….. (b)

where,
v[] is the particle velocity,
persent[] is the current particle (solution).
pbest[] and gbest[] are well-defined as specified before.
rand () is a random number between (0,1).
c1, c2 are learning factors(usually c1 = c2 = 2.)

4) Graph Module

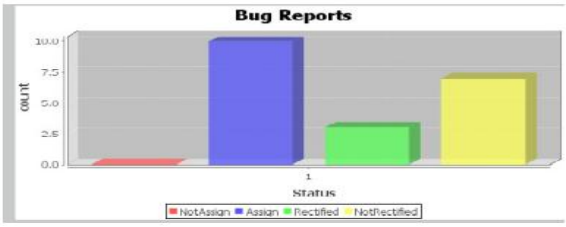

Fig. 3 Graph of bug reports

Figure 3 shows four parts as follow:

1) Firstly it will display how many bugs are not allocated to any developer. It will give ample status about the bugs to the supervisor so that he will come to recognize which bugs are not allocated yet.

2) Secondly it will display how many bugs are not allocated to any developer. It will give ample status about the bugs to the supervisor so that he will come to recognize which bugs are not allocated.

3) Thirdly it will display how many bugs are corrected by the developers. It will give ample status about the bugs to the supervisor so that he will come to recognize which bugs are corrected entirely.

4) Fourthly it will display how many bugs are not corrected by the developers. It will give ample status about the bugs to the supervisor so that he will come to know which bugs are not corrected yet.

IV. RELATED WORK

In this section, we analyze prevailing work on modeling bug data, bug triage, and the quality of bug data with defect prediction.

1) Modeling Bug Data

To explore the associations in bug data, Sandusky et al. [20] form a bug report system to inspect the dependence among bug reports. In addition to learning associations among bug reports, Hong et al. [8] construct a developer social network to inspect the relationship among developers based on the bug data in Mozilla project. This developer social network is supportive to understand the developer's civic and the project growth. By recording bug priorities to developers, Xuan et al. [28] recognize the developer prioritization in open source bug sources. The developer prioritization can differentiate developers and help tasks in software maintenance.

To explore the quality of bug data, Zimmermann et al. [31] plan surveys to developers and users in three open source projects. Based on the enquiry of surveys, they illustrate what makes a good bug report and train a classifier to find whether the quality of a bug report should be enriched. Identical bug reports deteriorate the quality of bug data by deferring the cost of managing bugs. To identify

replica bug reports, Wang et al. [25] propose a natural language processing method by matching the execution facts; Sun et al. [22] propose an identical bug detection method by enhancing a recovery function on multiple features.

To enhance the quality of bug reports, Breu et al. [5] have manually investigated 600 bug reports in open source projects to search for unnoticed data in bug data. Based on the analysis on the quality between bugs and requirements, Xuan et al. [26] handover bug data to requirements databases to increase the lack of open data in requirements engineering.

In this paper, we also emphasis the quality of bug data. In contrast to prevailing work on studying the features of data quality (e.g., [5], [31]) or aiming on identical bug reports (e.g., [7], [9]), our work can be used as a preprocessing technique for bug triage, which both increases data quality and decreases data scale.

2) Bug Triage

Bug triage targets to allocate an suitable developer to fix a new bug, i.e., to decide who should fix a bug. Cubrani c and Murphy [6] suggest the difficulties of automatic bug triage to decrease the charge of manual bug triage. They put on text classification techniques to guess related developers. Anvik et al. [1] inspect multiple methods on bug triage, including data preparation and typical classifiers. Anvik and Murphy [2] prolong above work to decrease the effort of bug triage by creating development-oriented recommenders.

Jeong et al. [9] discover that over 37 percent of bug reports have been reallocated in manual bug triage. They suggest a tossing graph technique to decrease-allocation in bug triage. To sidestep low-

quality bug reports in bug triage, Xuan et al. [27] train a semi-supervised classifier by joining unlabeled bug reports with labeled ones. Park et al. [18] alter bug triage into an optimization delinquent and offer a collaborative filtering methodology to decreasing the bug- fixing time.

For bug data, numerous other tasks happen once bugs are triaged. For example, severity identification [14] targets to identify the status of bug reports for further scheduling in bug management; time expectation of bugs [29] models the time cost of bug deception and expects the time rate of specified bug reports; reopened-bug analysis [20], [30] finds the wrongly fixed bug reports to evade deferring the software release.

In data mining, the difficulties of bug triage narrate to the problems of expert finding (e.g., [6], [24]) and ticket routing (e.g., [16], [19]). In contrast to the broad areas in professional outcome or ticket routing, bug triage only focuses on allocating developers for bug reports. Moreover, bug reports in bug triage are relocated into forms (not keywords in expert result) and bug triage is a kind of content-based classification (not sequence-based in ticket routing).

3) Data Quality in Defect Prediction

In our work, we discourse the difficulties of data reduction for bug triage. To our understanding, no prevailing work has examined the bug data sets for bug triage. In an associated problem, defect prediction, some effort has focused on the data eminence of software defects. In contrast to multiple-class classification in bug triage, defect expectation is a binary class classification problem, which targets to predict whether a software piece (e.g., a source code file, a class, or a module) contains errors according to the mined features of the piece.

In software production, defect prediction is a kind of work on software metrics. To enhance the data quality, Khoshgoftaar et al. [11] and Gao et al. [7] inspect the methods on feature selection to manage imbalanced defect records. Shivaji et al. [23] suggests a structure to inspect multiple feature selection algorithms and eliminate noise features in classification-based defect prediction. Besides feature selection in defect prediction, Kim et al. [13] present how to measure the noise confrontation in defect prediction and how to perceive noise data. Moreover, Bishnu and Bhattacherjee [4] practice the defect records with quad tree based k-means clustering to support defect prediction.

In this paper, in contrast to the above effort, we report the difficulties of data reduction for bug triage. Our work can be observed as an extension lead of software metrics. In our work, we calculate a value for a set of software pieces while prevailing work in software metrics calculate a value for an individual software piece.

V. CONCLUSIONS

Bug triage is an exclusive step of software maintenance in both manual labor charge and time rate. In this paper, we syndicate feature selection with instance selection to decrease the scale of bug data sets as well as enhance the data quality. To decide the order of applying instance selection and feature selection for a new bug records set, we abstract features of each bug records set and train an analytical model based on historical records sets. We empirically examine the data reduction for bug triage in bug sources of two huge open source projects, namely Eclipse and Mozilla. Our work affords a method to leveraging procedures on data processing to form reduced and high-quality bug data in software development and maintenance.

In future work, we plan on enlightening the effects of data reduction in bug triage to discover how to formulate a high quality bug data set and block a domain-specific software task. For predicting reduction orders, we plan to pay efforts to find out the potential correlation between the

features of bug data sets and the reduction orders.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Impact of data sampling on stability of feature selection for software measurement data," in Proc. 23rd IEEE Int. Conf. Tools Artif. Intell., Nov. 2011, pp. 1004–1011.

[8] 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.

[9] 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.

[10] 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

[11] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in Proc. 22nd IEEE Int. Conf. Tools Artif.
Intell., Oct. 2010, pp. 137–144.

[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] 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.

[14] 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.

[15] Mamdouh Alenezi and Kenneth Magel, Shadi Banitaan "Efficient Bug Triaging Using Text Mining" © 2013 academy publisher

[16] G. Miao, L. E. Moser, X. Yan, S. Tao, Y. Chen, and N. Anerousis, "Generative models for ticket resolution in expert networks," in Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, pp. 733–742.

[17] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design of bug fixes," in Proc. Int. Conf. Softw. Eng., 2013, pp. 332– 341.

[18] 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.

[19] Q. Shao, Y. Chen, S. Tao, X. Yan, and N. Anerousis, "Efficient ticket routing by resolution sequence mining," in Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2008, pp. 605–613.

[20] 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.

[21] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in Proc. 17th Working Conf. Reverse Eng., Oct. 2010, pp. 249–258.

[22] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng., 2011, pp. 253–262.

[23] S. Shivaji, E. J. Whitehead, Jr., R. Akella, and S. Kim, "Reducing features to improve code change based bug prediction," IEEE Trans. Soft. Eng., vol. 39, no. 4, pp. 552–569, Apr. 2013.

[24] J. Tang, J. Zhang, R. Jin, Z. Yang, K. Cai, L. Zhang, and Z. Su, "Topic level expertise search over heterogeneous networks," Mach. Learn., vol. 82, no. 2, pp. 211–237, Feb. 2011.

[25] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in Proc. 30th Int. Conf. Softw. Eng., May 2008, pp. 461–470.

[26] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone based multilevel algorithm," IEEE Trans. Softw. Eng., vol. 38, no. 5, pp. 1195–1212, Sept./Oct. 2012.

[27] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng., Jul. 2010, pp. 209–214.

[28] 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.

[29] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: An empirical study of commercial software projects," in Proc. 35th Int. Conf. Softw. Eng., May 2013, pp. 1042–1051.

[30] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in Proc. 34th Int. Conf. Softw. Eng., Jun. 2012, pp. 1074–1083.

[31] 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.