IJCTA, 9(23), 2016, pp. 155-164

Efficient Bug Triage Using Software Reduction Techniques

Shubham Krishna Aggarwal* S.S. Pande and Emmanuel M.*****

Abstract : Apart from Software Development, organization works etc. of companies, trading with the various types of bugs during the entire software development process is a very monotonous task. A debugger working manually would be just an excess of time while looking at the speed of development now a days. Out of the total budget companies spend for their development, estimate for this bug removal process has a significant amount. Thus for the companies to reduce this manual effort on bug eradicating process in recent year various bug triage systems are in use. Bug triage process deals with inspecting the extent to which a particular bug would damage a system, and accordingly produce reports for the same. Text classification is an essential domain for bug triage process to be effective. In recent years various steps are being taken to address difficulties like data reduction in text classification for bug triage.

Keywords : Bug Trigging Process, Text Classification, Data Reduction.

1. INTRODUCTION

With increase in software development companies in recent years, software repositories are also increasing. Software repositories are nothing but the databases which companies maintain include that include heterogeneous data related to software like output, e-mails, bugs information etc. With day to day increase in scale of software repositories, the complexities in the data also increase. Various data mining techniques are available now days that help to work on such complex data. Data mining helps software repositories to get informed about various problems or likewise analytics on the same. Large the data with the companies large is the software development process and more is the need for analyzing the bugs in the data [17].

Software repositories contains very vital factor that is bug repository, which acts as database for bugs and manages software bugs. Bug repositories are such databases which contain information about software bugs in detail. Bug repositories contain report about each bug like the updates related to bugs, way to fix the bug etc.

Now a day's these bug repositories are become large in size inviting for some methods to maintain the same. With constant increase in software repositories, maintaining the quality and handling the large scale bug data is a challenge. Most of the software companies have a bug tracking system installed at their place to keep track of the same. Bug tracking system helps the developers to be updated about the bug. Many open source project are on their way of development today. They have direct involvement of users in it. Users put up various queries related to the project and the developers of the project try to satisfy their queries. Even sometimes whenever a bug or a problem occurs even user can involve in discussion with the developer and try to solve the problem. This interaction between user and developer is not a easy task though as due to increasing number of reports for developer and the increasing number of problems for user both get annoyed at certain point of time [5].

* Department of Information Technology PICT, Pune, Maharashtra, India *Email- ershubham.aggarwal@gmail.com*

** Department of Information Technology PICT, Pune, Maharashtra, India *Email- sspande@pict.edu*

*** Department of Information Technology PICT, Pune, Maharashtra, India Email- hodit@pict.edu

Following points are mostly gathered while a customer-developer interaction:

1. List of frequently asked questions in bug reports.
2. Survey of question time, response rate and time.
3. Qualitative analysis of bug reports.
4. Feedbacks for bug tracking systems.

To understand the concept an example of UNIX operating system is the best. This is a open source operating system wherein work related to bugs removal is done in terms of patch. Here,

1. A new bug is filed
2. The maintainer periodically reviews the status of the bug report, and sees for any action taken on it.

When user or developer comes up with the solution that is patch for the bug reported, it would be included in the future versions of the system.

This communication between user and developer need to be more successful, thus in recent years a lot of work has been done on this aspect. Earlier software development mainly concentrated on the maintenance, and the day to day needs. But currently this sequence has changes and lot of effort needs to be done towards bug deduction [3].

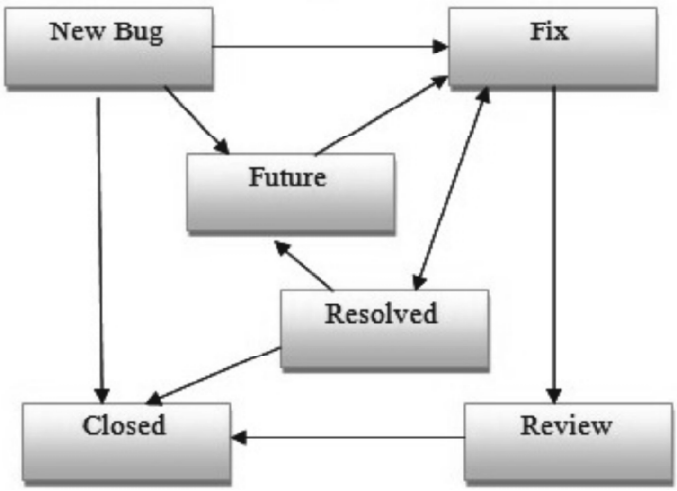

Fig. 1. Bug Triage Process.

Timely changes in the software codes these days are not as interesting as changes that fix the bugs. The bug fixer has all the old buggy data and the new buggy that which helps to properly resolve the issue.

In recent years lot of efforts are being taken to automate the bug tracking process. Triaging a bug is to decide upon certain parameter that helps to resolve the occurred bug.

Automatic Bug Triaging process is active area for research, which ensures that bug report

- has a sufficient information for developer to work.
- is not already been resolved
- is properly being files for respective project

Each report contains a substantial amount of information in the form of pre-defined fields, free-form text, attachments, and report activity logs [16].

Automatic bug triage system is evolved to reduce the manual work for the same. Existing systems try to map the bug report to a document so that bug triage problem becomes a text classification problem for more effective and automated bug triaging on the software systems [6].

Bug repositories (also known as *bug tracking systems*) are deployed in software projects for the storage and management of bugs. A bug in bug repositories is recorded as a *bug report*, which is occupied with the information of a software difficulty. Founded on the bug reports, developers can collect and reproduce bugs for

bug fixing. In practical software engineering, many software jobs are examined on bug repositories, e.g., conveying bugs to correct developers to reduce the time of bug fixing, briefing a long bug report into a short abstract, identifying duplicate bugs to avoid repetitive procedures and characterizing issues of bug fixing [14].

2. LITERATURE SURVEY

Various concepts related to bug triage process and different methods use to make the process more efficient are discussed below:

1. Reducing The Effort Of Bug Report Triage: Recommenders For Development- Oriented Decisions

John Anvik, C. Murphy [15] proposes a key collaborative hub for many software projects is a database of reports relating both bugs that essential to be fixed and fresh features to be added. This database is frequently called an issue tracking system or bug repository. The practice of a bug repository can progress the development process in a number of ways including tracking the work that needs to be done, allowing developers who are geographically distributed to communicate about project development tracking the evolution of the project by the number of outstanding bug reports and improving the quality of software produced.

To reduce the time and effort expended by triagers, it uses an approach in which a triager is presented with a list of suggestions from recommenders for various triage decisions. In this way, human involvement in triage is transformed by moving the triager's role from gathering information prior to making a decision to confirming a suggestion from the recommender. This article presented a machine learning-based approach to creating recommenders that assist with development-oriented decisions. This creates three different kinds of development-oriented recommenders: a developer recommender that suggests which developers might fix a report, a component recommender that suggests to which product component a report might pertain, and an interest recommender that suggests which developers on the project might be interested in following the report.

2. Advances In Instance Selection For Instance Based Learning Algorithms

Henry Brighton, Chris Mellish, Kluwer [2] proposes the simple nearest neighbor classifier suffers from the undiscerning storage of all presented training instances. With a large database of instances classification answer time can be slow. When blaring instances are existing classification accuracy can suffer. Drawing on the large body of pertinent work approved out in the past 25 years, here the principle tactics to solving these problems are reviewed. By deleting instances, both problems can be improved, but the criterion used is typically expected to be all surrounding and effective over many domains. So it claims against this position and presents an algorithm that opponents the most successful existing algorithm. When evaluated on 30 different problems, neither algorithm consistently outperforms the other constancy is very hard.

After reviewing the principle approaches grouped them into three classes: early schemes, current additions, and the formal of the art. The degree to which each class of algorithm realizes unintrusive storage reduction approximately mirrors this chronological order. It perceives that ICF algorithm and Wilson and Martinez' RT3 algorithm attains the highest degree of instance set reduction as well as the retention of classification accuracy: they are close to attaining unintrusive storage reduction. The degree to which these algorithms perform is quite impressive: an average of 80% of cases is removed and classification exactness does not drop significantly. The comparison providing here is important as, considering the number of approaches; few consistent comparisons have been finished.

3. A Review Of Feature Selection Methods On Synthetic Data

Veronica, Bolon-Canedo, Noelia Sanchez-Marono, Amparo Alonso-Betanzos [17] works on numerous artificial datasets are employed for this purpose, directing at reviewing the performance of feature selection methods in the occurrence of a falcate number or irrelevant features, clamor in the data, severance and interface between attributes, as well as a small ratio between number of samples and number of features. Seven filters, two implanted methods, and two wrappings are applied over eleven synthetic datasets, tested by four classifiers, so as to be able to choose a vigorous method, paving the way for its application to actual datasets.

This work analyses numerous feature selection methods in the literature and orders their performance in an artificial measured experimental scenario, contrasting the skill of the algorithms to select the related features and to discard the extraneous ones without permitting noise or redundancy to frustrate this process. A scoring measure will be presented to compute the degree of matching between the output given by the algorithm and the known ideal solution, as well as the classification accuracy. Finally, real experiments are presented in order to plaid if the conclusions extracted from this theoretical study can be deduced to real scenarios.

4. Automatic Bug Triage Using Text Categorization

D. Cubranic and G.C. Murphy [6] proposes a scheme to apply machine learning techniques to support in bug triage by using text categorization to expect the developer that should work on the bug built on the bug's description. It validates the approach on a collection of 15,867 bug reports from a big open-source project. This evaluation shows that this model, using supervised Bayesian learning, can appropriately predict 30% of the report assignments to developers. It includes investigation of using machine learning, and in specific text categorization, to "cutout the triage man" and automatically allocate bugs to developers based on the description of the bug as arrived by the Bug's submitter. The method would require no changes to the way bugs are presently submitted to Bugzilla, or to the way developers knob them once the bugs are assigned. The profit to software development teams would be to free up developer resources currently ardent to bug triage, while assigning each bug report to the developer with appropriate proficiency to deal with the bug.

Table 1. Various Concepts Related To Bug Triage Process.

Topic	*Description*
A good Bug Report	To make a better bug report following points need to be noticed - Observations regarding what developers expect and what reporters provide. - Tools that look after the qualitative analysis of the bug reports [5].
Characterizing and Predicting Reopening of Bugs	When a bug is fixed if it reopens again in some time following issues need to be addressed for the same - Actual quality of bug triage process - Looking at the issues that are not fixed - Searching areas that need better tools - Efficient bug triage process - Planning for bug triage process taking reopening of bugs into account [1].
Information needs in Bug Reports	To work on bug reports properly there must be proper informative communication between the developer and the user of the system [3].
Instance selection	Works on obtaining a subset or relevant instances that is bug reports in bug data [2].
Feature selection	Works on obtaining a subset of relevant features that is words in bug data [4].

Table 2. Various Methods and Techniques Used For Improving Bug Triage Process

Topic	*Description*
Improving Bug Triage with Bug Tossing Graphs	If a bug report is assigned to a developer, same can be reassigned to other developer this process is called bug tossing. This process mainly uses graphical model based on Markov Property. This model has 2 distinct characteristics: - Discovers developer networks and team structures. - Helps to assign better developers to bug reports [11].

Topic	*Description*
Detect Duplicate Bug Reports using Natural Language & execution Information	Whenever a bug is filed for a certain system, a triage process if the same bug already exists, if yes, it triggers a duplicated bug existence. To identify such duplicate reports various natural language algorithms are used.Basic Approach: - Evaluate NL-S that is similarities between old and new bug reports. - Evaluate Execution Information based of both old and new bug reports [9].
Code Change based Bug Prediction	This is based on machine learning domain. Machine learning classifiers are a good way to predict the bugs according to changes made in source code. The classifiers are trained according to the history of the existing software and then predict the changes occurring in new bug reports [8].
Cost Aware Triage Algorithm	This algorithm has two concepts: - treating bug triage problem as recommendation problem optimizing both accuracy and cost - Adopt content based collaborative filtering combining the existing CBR systems with collaborative filtering recommender [7].
Automatic Bug Triage Using Semi Supervised Classification	This uses labeling approach for bug triaging process. It uses classifier to convert unlabeled bug reports to labeled bug reports [6].

3. PROPOSED SYSTEM

In bug triage, a bug data set is transformed into a text matrix with two dimensions, specifically the bug dimension and the word dimension. In this work it influences the combination of instance selection and feature selection to produce a reduced bug data set. It swaps the original data set with the reduced data set for bug triage. Instance selection and feature selection are widely used procedures in data processing. For a given data set in a certain application, instance selection is to find a subset of relevant instances (*i.e.*, bug reports in bug data) while feature selection goals to get a subset of relevant features (*i.e.*, words in bug data). This pays the combination of instance selection and feature selection. To discriminate the orders of applying instance selection and feature selection, it gives the following signification. In fig 4.1 given an instance selection algorithm IS and a feature selection algorithm FS, FS - IS is used to denote the bug data reduction, which ûrst applies FS and then IS; on the other hand, IS - FS denotes ûrst applying IS and then FS.

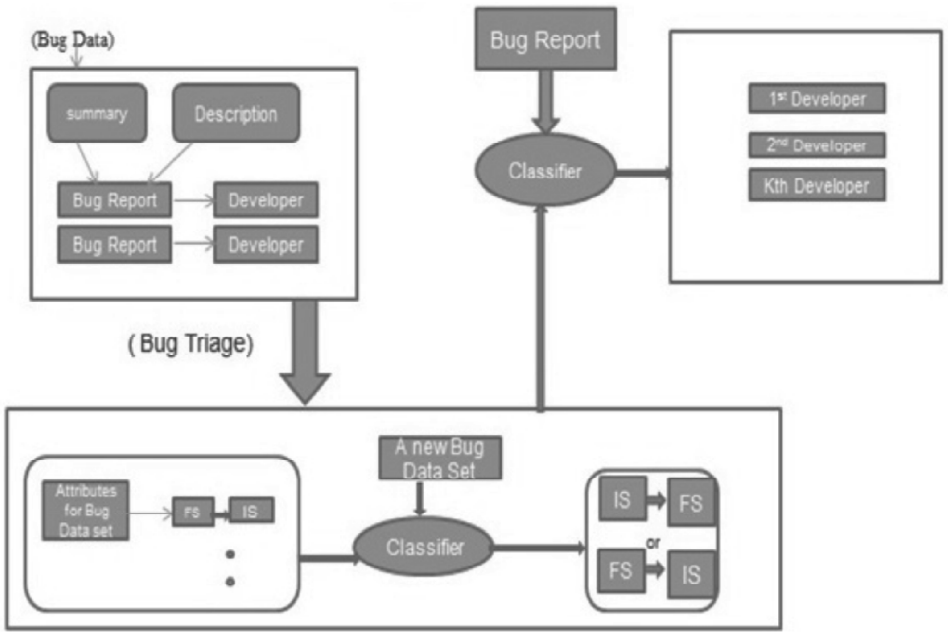

Fig. 2. Architecture Diagram.

This also checks the blank bug reports through the data reduction. A blank bug report denotes to a zero-word bug report, whose words are removed by feature selection. Such blank bug reports are finally removed in the data reduction then they offers none of information. The removed bug reports and words can be viewed as a type of noisy data.

ICF (Iterative Case Filtering) : ICF is an instance selection algorithm based on the k-Nearest Neighbor algorithm (kNN). The process of ICF consists of two steps, namely noise filtering and instance condensing.

CH (CHI Square Algorithm) : Chi-squared tests are erected from a sum of squared errors, or through the sample variance. Test statistics that follow a chi-squared distribution arise from a hypothesis of independent normally distributed data.

$$\lambda^2 = \sum(i = 1 \text{ to } k)(Oi - Ei)2/Ei$$

where, Oi = Observed frequency

Ei = Expected frequency

If two distributions are exact alike, $\lambda^2 = 0$ (But generally due to sampling errors, λ^2 is not equal to zero)

Random Forest Algorithm

Random forests is a collaborative learning method for classification, regression and other tasks, that works by building a horde of decision trees at training time and outputting the class that is the mode of the classes classification or regression of the individual trees. Random forests correct for decision trees practice of over fitting to their training set.

Algorithm Steps :

- Let the number of training cases be M, and the number of variables in the classifier be N.
- The number '*n*' of input variables are used to determine the decision at a node of the tree; m should be much less than N.
- Choose a training set for this tree by choosing M times with replacement from all M available training cases. Use the remaining cases to predict the error of the tree, by predicting their classes.
- For each and every node of the tree, randomly choose 'n' variables on which to base the decision at that node. Calculate the best split based on these 'm' variables in the training set.
- Each tree is fully grown and not lopped.

Table 3. ICF Algorithm.

ICF(T)	**11. compute coverage(*x*)**
1. Perform Wilson Editing	**12. progress = false**
2. for all *x* belongs to T do	**13. for all *x* belongs to T do**
3. if x classified incorrectly by *k* nearest neighbors then	**14. if reachable(*x*) > coverage(*x*) then**
4. flag *x* for removal	**15. flag x for removal**
5. for all *x* belongs to T do	**16. progress = true**
6. if *x* flagged for removal then *t* = *t*- (*x*)	**17. for all *x* belongs to T do**
7. iterate until no cases flagged for removal:	**18. if *x* flagged for removal then T = T –(*x*)**
8. repeat	**19. until not progress**
9. for all *x* belongs to T do	**20. return TWhere *x* is the current bug report which is compared by all otherAnd T is the collection of all bug reports**
10. compute reachable(*x*)	

4. EXPERIMENTAL WORK

Dataset Loading

In this experiment Dataset is being collected from the Eclipse bug data site named 'Bugzilla' in xml format which is being loaded in the sql database in order to make it appropriate for applying the preprocessing steps to obtain the desired textual data.

Table 4. Summary of Fetched Bug Dataset.

Bug ID	*Opening Time*	*Reporter ID*	*Update When*	*Update What*
334345	120920130546	104577	110420140732	Proxy settings don't work
335612	130620140323	196867	170820140138	Ditch the xpinstaller
326410	231120140757	108201	190320150619	Pages constantly load
329246	180720141426	948905	260920150754	Bug in the xml window
345797	141120141729	161026	290520151929	File/Import does not find Netscape Communicator

Application of Preprocessing Steps on Dataset

In this Experiment those Bug reports are chosen, which are fixed and duplicate (based on the items status of bug reports). Moreover, in bug repositories, several developer shave only fixed very few bugs. Since bug triage aims to predict the developers who can fix the bugs, it follows the existing work to remove unfixed bug reports, *e.g.*, the new bug reports or will-not-fix bug reports.

Table 5. An Overview of Attributes for a Bug Dataset.

Index	*Attribute Name*	*Description*
B1	#Bug reports	Total number of Bug reports.
B2	#words	Total number of Bugs in all the Bug reports.
B3	Length of Bug Reports	Average number of Bugs of all the Bug reports.
B4	#Unique words	Average number of Unique Bugs in each Bug Reports.
B5	Ratio of sparseness	Ratio of sparse terms in the text matrix. A sparse term refers to a word with zero frequency in the text matrix.
B6	Entropy of severities	Entropy of severities in Bug reports. Severity denotes the importance of Bug reports.
B7	Entropy of Products	Entropy of Products in Bug reports. Product denotes the sub object.
B8	Entropy of Components	Entropy of Components in Bug reports. Component denote the sub-sub project.
B9	Entropy of priorities	Entropy of priorities in Bug reports. Priority denotes the level of Bug report.
B10	Entropy of words	Entropy of words in Bug reports.

Fetching Data from GUI Interface

In this experiment, data is being fetched from the database and been applied the preprocessing steps, that is data extraction is taken placed. Now the data is prepared for feature extraction *i.e.* word dimension reduction is applied on the data in order to reduce the words attributes from the description.

Table 6. Bug Report Status with Full Description

Bug ID	*Status*	*Description*
1047685	Resolved Fixed	Description David McKnight 2009-07-24 12:37:33 EDT with the fix for Bug 277141, RSE reuses a previous System edible remote file if one.....
1053467	Closed Fixed	Description Mindan xu 2009-07-16 03:54:38 EDT created attachment 142488 report design description: Context object content.....

Reduction of Instances and Attributes after Applying ICF-CH

In this experiment algorithm firstly ICF is applied on the dataset to reduce the bug dimension or to decrease the vertical size of the dataset. Afterwards on the reduced dataset CH is applied to reduce the word dimension or to decrease the horizontal size of the dataset.

In the given example, initially dataset contains 1000 bug reports which contain 8527 featured attributes. On applying Instance selection on it the reduced number of instances is came to be 649, and then Feature selection is applied on these instances which reduced the attributes count to 24.

Table 7. Instances and Attributes Count Before and After Reduction

Property	*Statistics*
Database	Database 1
Method	ICF—CH
Instances Before	1000
Instances After	649
Attributes Before	8527
Attributes After	24

4. RESULTS AND DISCUSSION

The order of reduction used is first instance selection then feature selection *i.e.* firstly the dataset passes through ICF algorithm which reduce the dataset vertically and the it goes to Chi square algorithm which reduce the no. of key features or reduce horizontally.

Then the prediction is made through the random forest classifier which allocates the appropriate bug reports to the appropriate developer.

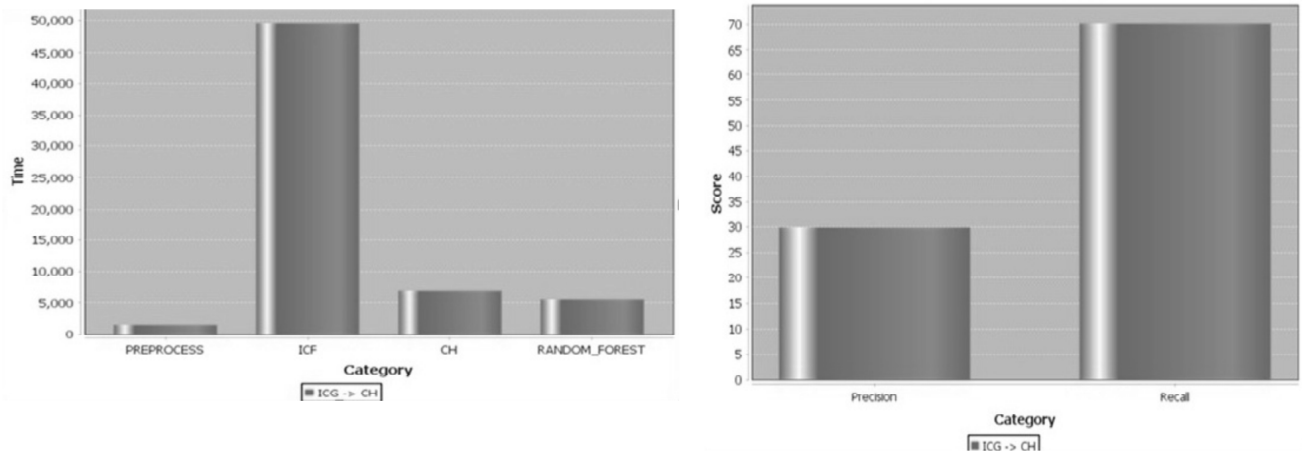

Fig. 3. Evaluation Time & Precision Recall Graph.

Fig. 3 shows the time required to perform by the different modules to complete its execution phase on the dataset. It can be seen from the graph, ICF takes most of the time to complete its processing on the dataset since it compares all the reports by itself to reduce the vertical dimension of it and also shows that nearly 29% of the reports are correctly predicted to the developers in our system. It outperforms the overall efficiency of the Bug triage system in compare to the current system.

5. CONCLUSION AND SCOPE FOR FUTURE WORK

A bug repository plays an significant role in handling software bugs. Software bugs are unavoidable and fixing bugs is lavish in software development. In a bug repository, a Bug is preserved as a bug report, which chronicles the textual description of reproducing the bug and apprises according to the status of bug fixing. A bug repository affords a data platform to support many types of tasks on bugs, e.g., fault prediction, bug localization and resurrected bug analysis. Due to the daily-reported bugs, a huge number of new bugs are kept in bug repositories. There are two challenges related to bug data that may disturb the effective use of bug repositories in software development tasks, specifically the big scale and the short quality. On one pointer, due to the daily-reported bugs, a large number of new bugs are kept in bug repositories. On the other hand, software techniques agonize from the low quality of bug data. Hence, most researches are going in bug triaging systems.

Existing systems have disadvantages that in those systems fresh bugs are manually triaged by an expert developer. Due to the large number of regular bugs and the lack of expertise of all the bugs, manual bug triage is expensive in time charge and low in accuracy. On the other pointer, software techniques in those systems suffer from the little eminence of bug data.

This Dissertation work combines feature selection with instance selection which is used to reduce the scale of bug data sets as well as recover the data quality. To control the order of smearing instance selection and feature selection for a new bug data set, it excerpt attributes of each bug data set and train a predictive model built on historical data sets. It empirically investigates the data reduction for bug triage in bug repositories of two great open source projects, namely Eclipse and Mozilla. This work delivers an approach to leveraging techniques on data processing to form abridged and high-quality bug data in software development and preservation. In future work, plan is on refining the results of data reduction in bug triage to discover how to prepare a high quality bug data set and challenge a domain-speciûc software task.

6. REFERENCES

1. 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, pp. 1074-1083, 2012.
2. H. Brighton and C. Mellish, "Advances in Instance Selection for Instance-Based Learning Algorithms," Data Mining Know. Discovery, Vol. 6, Issue No. 2, pp. 301-310, 2002.
3. 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, pp. 301-310, Feb. 2010.
4. Y. Yang and J. Pedersen, "A comparative study on feature selection in text categorization," in Proc. Int. Conf. Mach. Learn., pp. 412-420,1997.
5. 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, Issue No. 5, pp. 618-643, Oct. 2010.
6. D. Cubranic and G.C. Murphy, "Automatic bug triage using text categorization," in Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng., pp. 92-97, Jun. 2004.
7. 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., pp. 139-144, Aug. 2011.
8. 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, Issue No. 4 , pp. 552-569, Apr. 2013.

9. 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., pp. 461-470, May 2008.

10. 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, Issue no. 5, pp. 1195-1212, Sept./Oct. 2012.

11. 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., pp. 111-120, Aug. 2009.

12. Q. Hong, S. Kim, S. C.Cheung, and C. Bird, "Understanding developer social network and its evolution," in Proc. 27th IEEE Int. Conf. Softw. Maintenance, pp.323 332, Sep. 2012.

13. 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, pp. 80–84, May 2004.

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

15. 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, Issue No. 3, article 10, Aug. 2011.

16. "Eclipse"Online (Available) Internet: http://eclipse.org/(Accessed On: 12th Feb 2016).

17. V. Bolon Canedo, N. Sanchez Marono, and A. Alonso Betanzos, "A Review of Feature Selection Methods on Synthetic Data," Knowl. Inform. Syst., Vol. 34, Issue No.3, pp. 483–519, 2013.