An Approach for bug fixing using Naïve Bayes classification

Neha Jund[1]
Department of Computer Science and Engineering,
Chitkara University, Himachal Pradesh, India
[1]neha.jund27@gmail.com

Dr.Shaily Jain[2]
Department of Computer Science and Engineering,
Chitkara University, Himachal Pradesh, India
[2]shaily.jain@chitkarauniversity.edu.in

***Abstract*—Open source projects for example Firefox and Eclipse have open source bug warehouses. User reports bugs to these repositories. People uses these repositories are usually non-technical and are not able assign correct class to these bugs. The change history of a software project contains a huge collection of code changes that record previous development experience. In the warehouse that records a software project's change history, there are various changes where developers classify bugs as opposed to adding new features or re-factoring source code. Developers are usually skilled in particular areas. For example, few developers are expert in java functionality and others are in GUI. Assigning a specific bug to suitable developer could save time and would help to continue the concern level of developers by assigning bugs according to their choice. However, assigning right bug to right developer is actually difficult for tri-ager without knowing the actual class, the bug belongs to.**

In this research, we have categorized the bugs in variant labels on the basis of summary of the bug. Naïve Byes text classifier is used for categorization purpose. For feature selection of bugs, TFIDF algorithm was used. Naive Bayes classifier is the best known classifier for text mining as it does not use iterative steps and hence is fast and less time consuming. Naive Bayes classifier is simple to understand and implement yet powerful. Automatically fixing the recurrent bugs will save a lot of time in debugging the software and also will save time which is put to fix the same type of bugs again which are already fixed in previous versions. Afterwards the performance of approach is checked by using ROC (receiver optimization curve) considering the false positives and true positives and also Area, Confident interval, Standard deviation. Using Naïve Bayes and, we get average of 73% accuracy.

Keywords-Text mining, classification, software repositories, open source software projects, accuracy.

I. INTRODUCTION

The presence of bugs is a relentless quality of human created software. This is not intentional. The creation of bug free software has been a aim for engineer and researcher alike. One useful starting point and whether the frequency of bug kinds is common across multiple systems. Once this information is known, it is possible to grade the kinds of bugs from most to least common, and then focus research concern on diminishing the most common types of bug.

Source code repositories hold a treasure of information that is not only useful for maintaining and building source code, but also as a detailed log of how the source code has emerged during development. If a portion of the source code is re-factored, evidence of this will be in the repository. The code representing how to use the software pre and post re-factoring will exist in the warehouse. As bugs are fixed, the changes made to correct the problem are documented. The challenge, then, is to develop tools and techniques to automatically select and use this information.

It is easy for programmers to think about types of bugs that might occur, and then formulate a tool to look for these bugs. However, the space of possible tools to build is massive. Instead of inventing solutions and looking for bugs .Program maintenance and repair is one of the most time exhausting and familiar jobs for software projects. Catching and repairing bugs in software is essential for the software to be permanent, and repairing the bugs usually requires only small changes to the code base. However, finding the bugs and noticing the correct solution is not always an obvious or simple task, even for the most insignificant of software bugs. Mining Software Repositories- To understand constantly emerging software systems is a very daunting task. Software systems have background of how they come to be and this history is maintained in software warehouses. Software warehouses are the artifacts that document the evolution of software systems. Software repositories often contain data from years of progress of a software project.

Any software repository can be mined not necessarily the code, bug or documented communication repositories. In this paper we present an automated bug classification system. Proposed approach uses Naïve Bayes text classifier to classify bugs from open bug repository. Data from eclipse is used and 73% of precision accuracy is obtained.

II. PROBLEM FORMULATION

Prioritizing of bugs to developer to fix them is a tedious and

time consuming task. Developers are usually expert in some specific area. For example few developers are skilled in GUI and others are in pure java functionality. Assigning a particular bug to suitable developer could save time as well as would help to maintain the concern level of developers by assigning those bugs according to their choice. However assigning right bug to right developer is quite difficult for sort without knowing the actual class a bug belongs to. This research proposes a technique for classification of open source software bugs using the summary arranged by bug reporters. The paper discusses the issues and problems associated with manual bugs fixing. We collect the data to extract recurring bugs. We will study different types of bugs for finding recurrent bug fix pattern and then apply normalization, stemming and labeling on bug fixes.

III. Literature Review

Mircea Lungu et al. [1] mine the change history of 717 open source projects to extract bug-fix patterns and also manually inspect many of the bugs found to get insights into the contexts and reasons behind those bugs. Missing null checks and missing initializations are very repetitive. They can be automatically detected and fixed.

Kim et al. [2] created a tool named BugMem that extracts bug fix rules from the history of a project and applies bug detection. This approach is smart and new but the rules are not "patterned" and they are instead saved in a concrete form. This directs to the saved fix rules being suitable only to code clones within the same project. Code clone tracking tools would perform surely better by following the changes of a clone and applying it on all other clones.

Anvik et al. [5] presented a semi-automated technique to assign issue reports to developers. A machine-learning algorithm is utilized on bug reports to learn the kinds of reports each developer resolves.

Chen et al. [8] created a tool (CVS Search) that searches for fragments of source code by using CVS comments. CVS Search allows one to better search the most recent form of the code by looking at previous reports to better understand the current version.

Ostrand et al. [10] describe a tool that automatically looks at the attributes of a software project and, utilizing historical data, anticipates which files are likely to contain a larger number of faults.

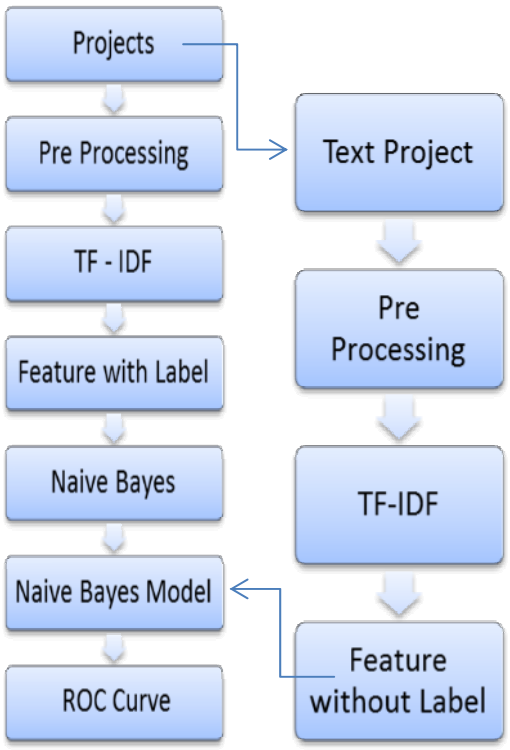
Graves et al. [11] use change histories to understand how code ages. Code is to be aged if its structure makes it unnecessarily difficult to understand or maintain. Data based on change history is more beneficial in predicting fault rates than metrics supported on the code, such as size.

Antoniolet al. [13] describes the different types of classifier and discussed the drawback the Naïve Bayes classifier in detail. A feature selection technique applicable to categorization-based bug prediction is proposed. Technique is applied to foresee bugs in software changes, and execution of Naive Bayes and Support Vector Machine (SVM) classifiers is characterized The Naive Bayes classifier strongly simplify learning by assuming that features are independent given class.

R. Koschke et al. [11] proves a modal for understand the data characteristics which affect the performance of naive Bayes. Their approach uses Monte Carlo simulations that allow a systematic study of classification accuracy for several classes of randomly caused problems. They analyze the impact of the distribution entropy on the categorization error, showing that low-entropy characteristic distributions yield good performance of naive Bayes. They also prove that naive Bayes works well for certain closely functional feature dependencies, thus reaching its best execution in two opposite cases: fully independent features (as expected) and functionally dependent features (which is startling).

IV. Problem Solution

This section describes the proposed approach for bug classification, data used for classification task and results obtained in different experiments.



1. Input Data

Eclipse data is obtained from Git Hub-an open bug repository. Data set of almost 717 open source projects is

mined to extract bug-fix patterns. This data is divided into training and testing groups and experiments are performed on different set of data from these groups.

2. Pre-processing

Pre-processing of data is the most important step of data mining. Data taken from bug repositories is in raw form and this data cannot be directly used for training the classification algorithm. First of all the data is pre-processed to make it useful for training purpose. Data pre-processing is the most important step of data mining and time consuming as well. Stop-words dictionary and regular expression rules are used to filter useless words and filter the punctuation respectively.

3. There are a number of feature selection techniques such as Term Frequency Inverse Document Frequency (TFIDF), and Document Frequency (DF). In this research, and TFIDF algorithms are used for feature selection.

4. Feature Selection

Most feature selection techniques perform either exhaustive or heuristic search for an optimal set of features. They typically only consider the labeled training set to get TTDWX the most suitable features. When the distribution of instances in the labeled training set is varied from the unlabeled test set, this may result in large generalization error. In this paper, a fusion of heuristic measures and exhaustive search based on both the labeled data set and the unlabeled data set is proposed. Information on both the labeled data set and the unlabeled dataset is applied to choose a better fusion of features.

5. Model for prediction

When the bug is first detected to repository, it is submitted to our proposed system. System extracts all the terms in these records using bag of words approach. These features are used for training of classification algorithm which is then used for categorization of bug reports. The classification algorithm used in proposed system is Naïve Bayes.

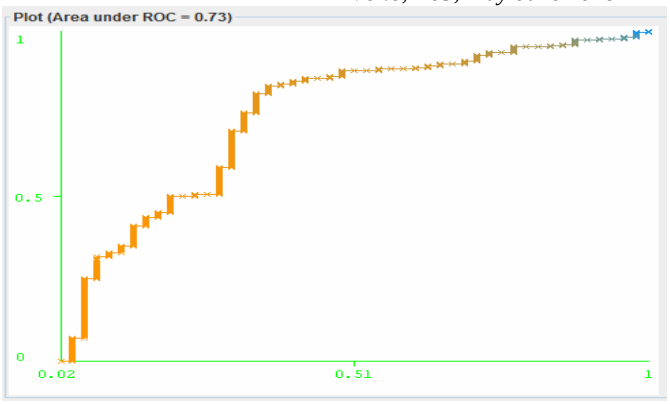
6. Classifier Modeling

Text classification is an automated technique of finding some metadata about a document. Text classification is used in many areas like document indexing by suggesting its categories in a content management system, spam filtering, automatically arranging help desk requests etc. Naïve Bayes text classifier is used in this research for bug classification. Naïve Bayes classifier is based on Bayes' theorem with independent prediction and is a probabilistic classifier. INDEPENDENCE means the classifier predicts that any feature of a class is unrelated to the presence or absence of any other feature.

7. ROC

The performance of approach is checked by using ROC (receiver optimization curve) considering the false positives and true positives and also Area, Confident interval, Standard deviation. Using Naïve Bayes.data set and the unlabeled dataset is applied to choose a better fusion of features.

V. EXPERIMENTAL RESULTS



Results of automated bug classification are obtained on the basis of prediction accuracy. Prediction Accuracy is defined as the Ratio of the bug reported with correct class to the total number of bug reported. In this research an automated system for classifying software bugs is formulated, using Naïve Bayes text classifier. The area under ROC is 0.73. It visibly shows that prediction accuracy increases as training to testing ratio increases. Highest accuracy is obtained when this ratio is 73%.

VI. FUTURE SCOPE

The system can be further ungraded by applying feature selection techniques other than Naive Bayes and TFIDF. It is not possible to design a system that fulfills all the requirements of the user, user requirements keep changing as the system is being used. In this paper, we basically reviewed various bug classification techniques and architecture also discussed the benefits of the research as well as their importance in such buildings. In future work, we will implement a robust and improved bug fixing technique which will outperform all the existing techniques.We will extract some more features and apply some more classification algorithm. Using the new technique and specific features we will try to improve the accuracy of classification.

REFERENCES:

[1] Osman, Haidar, Mircea Lungu, and Oscar Nierstrasz. "Mining frequent bug-fix code changes." Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014.

[2] S.Kim, K. Pan and E. E. J Whitehead, Jr., "Memories of bug fixes," in Proceedings of 14th ACM SIGSOFT International symposium on Foundations of software engineering, SIGSOFT ' 06/FSE-14,(New York, NY USA).

[3]https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Naive_Bayes_classifier.html

[4]http://in.mathworks.com/help/stats/naivebayesclass.html?nocookie=true

[5] Anvik, J., Hiew, L., and Murphy, G. C., "Who Should Fix This Bug?" in Proceedings of 28th International Conference

on Software Engineering (ICSE'06), Shanghai, China May 20-28 2006, pp. 361-370.
[6] http://scikit-learn.org/stable/modules/naive_bayes.html
[7]http://en.wikipedia.org/wiki/Naive_Bayes_classifier#Training.
[8] Daniel M. German" Mining CVS repositories, the softChange experience" Proceedings of the International Workshop on Software Clones (IWSC). 2009.
[9] http://www.statsoft.com/textbook/naive-bayes-classifier
[10] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, “Where the Bugs Are,” Proc. 2004 ACM SIGSOFT Int’l Symp. Software Testing and Analysis (ISSTA ’04), July 2004.
[11] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, “Predicting Fault Incidence Using Software Change History,” IEEE Trans. Software Eng., vol. 26, no. 7, pp. 653-661, July 2000.
[12] http://en.wikipedia.org/wiki/Naive_Bayes_classifier.
[13] Mockus, A., Fielding, T., and Herbsleb, D., "Two Case Studies of Open Source Software Development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology vol. 11, no. 3, July 2002, pp. 309-346.
[14]http://www.thearling.com/text/dmtechniques/dmtechniques.htm.
[15] http://en.wikipedia.org/wiki/GitHub.
[16] https://github.com/git/git.