AUTOMATED CATEGORIZATION OF BUGS

Hariprasad B P[1], H M Girish[2], Rakesh Y[3], Susheela N[4], Pradeep Kumar H S[5]

[1,2,3,4]*Student, Dept. of Information Science and Engineering, National Institute of Engineering, Mysuru, India.*
[5]*Assisstant Professor, M.Tech from VTU & BE from University of Mysuru, India.*

Abstract

About 40 percent of cost spend by software companies in dealing with software bugs. A bug triage is step of fixing bugs that assigns a new bug to developer. An automatic bug triage is conducted to decrease the cost in manual work and the text classification techniques are applied. In this paper, address the problem of data reduction for bug data which are collected from various products like, Mozilla, open office and eclipse, to improve the quality of bug and to reduce the scale of bug data. To reduce the word dimension and the bug dimension we combine feature selection with instance selection which done simultaneously. To determine the feature selection and instance selection, we can a build a predictive model for a bug data set by extracting attribute from previous bug data set.

Keywords—*Software Bugs, feature selection, instance selection, attributes*

I. INTRODUCTION

The author propose that open source project supports an open bug, both users and developers can report bugs. The report which appears that must be triage to determine attention to which it is needed. And also the developer will be assigned of resolving the report. The large open source project are loaded by the bug rate repository where new reports appears in that repository of bugs. In this paper we provide to one part of process by a semi-automated approach to developers. This approach is based on machine learning to open bug repository for learning the kinds of reports which each developer resolves the report. To resolve the report, classifier produced by the machine learning which suggest a small number of developers. Our approach is applied to the open source with less result. Under this approachthe condition are defined which is applied to machine learning repository in open source project.
This model reduces the scale of given bug data by removing stop words which are not required for the classification of the bug.
Tokenization is then applied to the clean data to calculate frequency. After the frequency is computed, the frequency is given and source computation is done. The algorithm detects whether the two bugs are similar or not, for duplication by calculating the union and intersection sum so that bugs can be found in grouping.

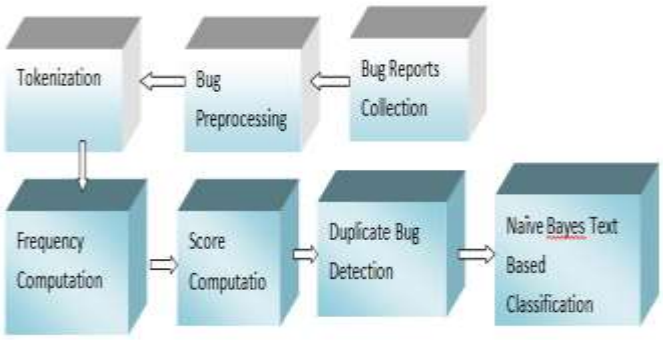
By using Naïve bayes algorithm which finds the text classification of bugs, which then computes the bugs probability to the related classes like performance, usability, user interface and design. Each class represent their unique feature, the algorithm then computes the enhance contingency, that classify the bug. The classify bug is then posted to the developer belonging to that class. This methodology can be applied to the software development stages which reduces the time and cost in bug processing.

II. EXISTING SYSTEM

To detect the duplicate bug reports the previous approach is used by the bug reports to specific contextual material, software engineering terms are listed as follows as Nonfunctional requirements and architecture keywords. In the world list contexts, the bug reports contains a word and this report is considered as to be associated with that context. This information helps to improve bug-duplication methods.

III. PROPOSED SYSTEM

Methodology



Collection of Bug Reports: All the products of bugs are in the form of Software Engineering products such as open source, Mozilla and eclipse. A set of bugs are collected in the form of bug-id, component, priority, type, version, status and description.

Bug Preprocessing:
Stopwords can be removed from bug description and this module helps in removing Stopwords. The stopwords are given in the web mining are in standard words in the project. Some of the stopwords are used in this project are and, across, above, almost, again, ahead, against, apart, all, allow, alone, already, also, anyway, amidst, an, according, actually, alongside, ago, about, anyhow, anybody, aside, are, aren't, appreciate, anything, appear.

Tokenization: A clean bug which is converted into a sequence of tokens is called a Tokenization process. Each of the token is associated with a bug{**TokenId, TokenName, BugId, and ProductId**}

Frequency Computation: The number of times a token appears in the review is known as frequency computation. If any redundancy exists the frequency will be removed and it will be stored in the format **{FreqId, Token Name, Freq, BugId and ProductId}**

Score Computation: It is a computation which is performed per Token and also computed across the bugs by using the below given formula and is stored in the standard format**{ScoreId, Frequency, IDFT, Score, BugId, ProductId}**

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D).(k_1+1)}{f(q_i,D)+k_1(1-b+b).\frac{|D|}{avgdl}}$$

f = *frequency*
IDF = *Inverse Document Frequency*
D=*length of document*
avgdl = average document length in the text collection
k1 = 1.2
$b = 0.75$ *IDF*(*qi*)

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

N = *number of documents*
$n(q_i)$ = *number of documents containing* q_i

By comparing 2 bugs with respect to Title & description the textual similarity can be computed.

$$textual_{description}(d_1,d_2) = BM25F(d_1,d_2)$$

Where,
d_1 = *document*1
d_2 = *document*2

Duplicate Bug Detection: To detect or find whether the 2 bugs are similar or not we use this algorithm, and also to find the Intersection sum & Union sums then the bugs can be treated as in terms of grouping.

Naïve Bayes Text Classification: The text classification of the bugs can be found by using naïve bayes text classification algorithm, this algorithm also computes the probability of a bug that belongs to the class **{c1, c2, c3, c4}**. Each class represented as unique in feature then the algorithm can be computed as contingency and enhanced contingency and also classifies the given bugs.

IV. SYSTEM ANALYSIS AND DESIGN

A Design is the most important aspect in a software development area. System organization is established by a process called design that satisfies the non-functional and functional system requirements. Large Systems are decomposed into sub-systems to provide some set of services. Software architecture is the output of the design process.

A. Data Flow Diagrams

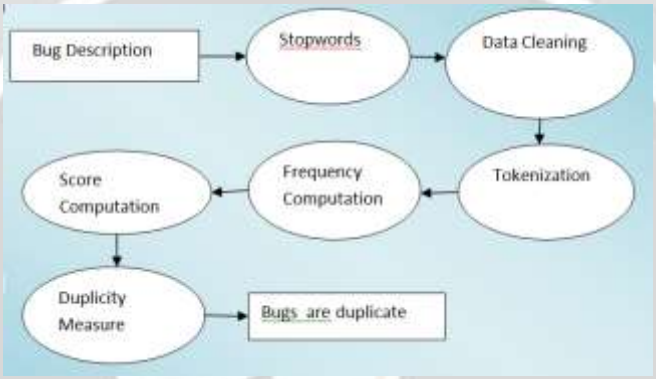


Figure shows the DFD Level 2 the input to the DFD Level 1 are Bug details which will contain the bug id, bug title, bug description, priority and type acts as input. The stop word analysis module is responsible for retrieving the stop words, adding and removing the stop words, Data cleaning is used to clean the bug description and bug title. Tokenization converts the clean title and clean description into a sequence of words. Frequency computation computed the repetition of the tokens. Similarity measure is computed based on score computation. The output is to indicate whether the bugs are duplicate or not

B. Naïve Bayes Classifier

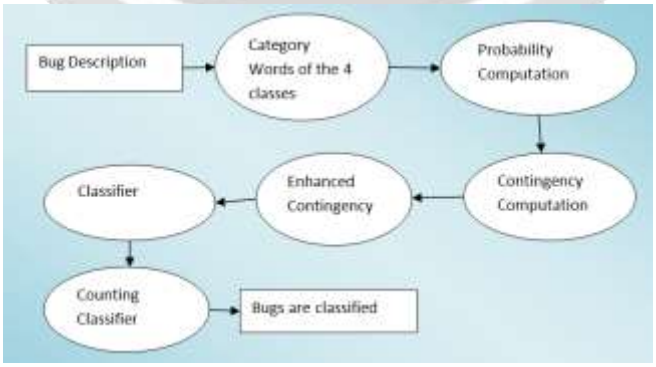
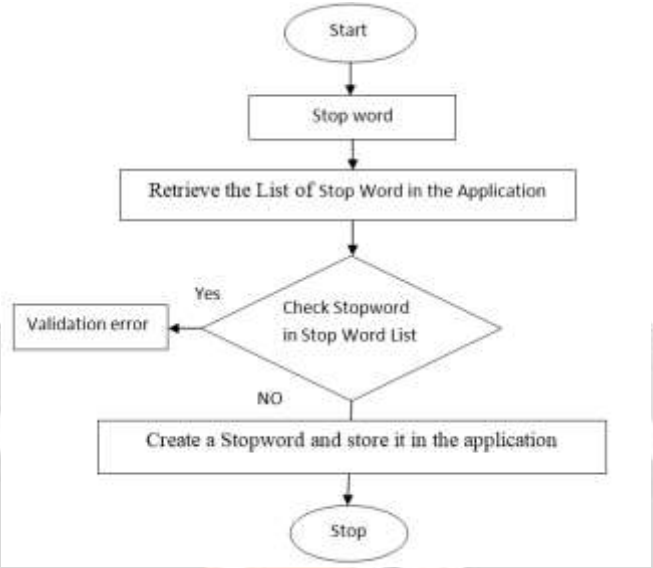


Figure shows the DFD Level 2 the input to the DFD Level 2 are Bug details which will contain the bug id, bug title, bug description acts as input. The category module has keywords belonging to each of the categories. The Bug collection matrix is scanned and then the bug description is scanned and probability can be computed for each and every category the contingency is computed which is a probability, negative probability, and positive other and negative other. Once these are computed then enhanced contingency is computed which has positive

cat ratio and other cat ratio. Classifier is a process which assigns the bug to a category. The counting of the bugs is done per category and finally bugs are classified and obtained.

C. Detailed Design

Stopword Creation Module is responsible for creating the stopwordif the stopword exist then validation error is shown otherwise stopword is created.



Stopword Creation Module

V. FUTURE SCOPE

Several different adaptations of the application and various other planned experiments and possible improvements have been set aside for the future due to the shortage of time. Some of our planned future endeavors pertaining to the project are explained in this section. We aim to further improve the accuracy of the classification algorithm. We aim to look into to the other bug data combination with computer generated samples that can help extend the framework to generate better classification results.

VI. CONCLUSION

Now that we have a detailed understanding of our project and the outcomes of our tests, we can finally draw some conclusions from these outcomes. This project revolves around building classifiers using imbalance bug datasets and posting it to developer automatically. The results show that the Naive Bayes algorithm demonstrated a superior performance in classification of bug type, small dataset gave 89% accuracy but when large dataset are used the Naive Bayes gave an accuracy of 82% on increasing the size of the dataset the accuracy gradually decreased. The project contributes to the field by developing a new workflow in bug processing by reducing the manual effort of the developer. The major strength of the workflow is that it reduces the duplication of bug report which again reduces the scale of the bug datasets. The classified bug is then posted to the developer belonging to that class and with further improvement it can reduce the process of bug triaging.

REFERENCES

[1] 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.

[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] C. C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.
[4] 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.

[5] 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.

[6] 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.