International Journal of Research in Engineering Technology -– Volume 1 Issue 3, Mar – Apr 2016

RESEARCH ARTICLE OPEN ACCESS

An Effective Usage of Bug Triage in Software Industry using BM25F algorithm and Naïve Bayes Text Classification

AnushaSrinivas[1] , Siddaraju[2]

Computer Science, Dr.Ambedkar Institute of Technology, Bangalore

Abstract:

Software firms devote over 45 percent of cost in trade with software bugs.A foreseeable step of setting bugs is known as bug triage, which intentions to correctly assign a developer to a new bug. To drop the time cost in labor-intensive work, to conduct automatic bug triage different text classification techniques are applied. In this project, we address the problem of data reduction for bug triage, i.e., how to improve the quality and reduce the scale and of bug data. We associate (chain) instance selection with feature selection to simultaneously reduce data scale on the bug dimension and the word dimension. To apply instance selection and feature selection, previously order of applying must be determined; the attributes are extracted from historical bug data sets and predictive model for a new bug data set is built.The results will illustrate that our data reduction can effectually reduce the data scale and bug triage accuracy can be improved. This work provides one of the leveraging techniques to form reduced and high quality bug data on data processing which can be used in software development and maintenance.

Keywords **—Tokens, IDF (Inverse Document Frequency), BM25F algorithm, Naïve Bayes Text Classification.**

I. INTRODUCTION

One of the interdisciplinary domains is mining. By this data mining features, problems in software industry can be dealt. Software repositories, is the name given for combined documents such as source code, bugs, emails, and specifications in the modern software development process. Traditional approach is not suitable for modern day software development process. Since data is large scale and complex.

Some of the real world scenarios and to cover unseen patterns, interesting information around the globe can be gained from data mining approach in their respective fields.

In software industry, bugs are managed by using bug repository (i.e. cart which stores bug details).Software bugs are unavoidable and fixing bugs is expensive process in software development. Over 45 per cent of cost is spent by the company to fix the bugs.

Bug tracking systems are deployed by the software industries to support information gain i.e. collection and to assist developers to handle bugs. Textual description of bug and bug details i.e. status of the bug is reported known as bug report in the bug repository. Some of the tasks performed by the usage of bug reports are bug localization, fault prediction, prevention and some of the other tasks. Collection of bug reports is known as bug data. Some of the major concerned issue with respect to bug data is scalability and quality attributes. Due to the daily-reported bugs; a large number of new bugs are warehoused in bug repositories.

II. RELATED WORK

In the paper titled “Who should fix this bug” [1] the authors propose that Open source development projects typically support an open bug repository to which both developers and users can report bugs. The reports that appear in this repository must be

triaged to determine if the report is one which requires attention and if it is, which developer will be assigned the responsibility of resolving the report.

In the paper titled "Information needs in bug reports: Improving cooperation between developers and users" [2]the authors propose that for many software projects, bug tracking systems play a central role in supporting collaboration between the developers and the users of the software. In our Project bugs are classified and directly given to respective developers.

In the paper titled "Reducing the effort of bug report triage: Recommenders for development-oriented decisions"[3] the authors propose that A key collaborative hub for many software development projects is the bug report repository. Although its use can improve the software development process in a number of ways, reports added to the repository need to be triaged. A triager determines if a report is meaningful. Meaningful reports are then organized for integration into the project's development process. To assist triagers with their work, this Bug's presents a machine learning approach to create recommenders that assist with a variety of decisions aimed at streamlining the development process.

III.EASE OF USE

In the current approach first a list of bug reports across streams namely Eclipse, Mozilla and Open office are collected and then they are cleaned using a stream of stop words. Once the clean data is obtained the tokenization is performed on the bug reports and text frequency is computed. After performing IDF computation, Feature Vector is computed and list of categorical similarities are measured.

IV.METHODOLOGY

A. ***Data collection***

It is a process in which bug reports of various products are collected. Some of the products list are Mozilla, Firefox, eclipse, open office etc.

All the bugs will be collected as a set {Bug Id, Component, Priority, Type, Version, Status, and Description}.

B. ***Noise Reduction***

It involves eliminating unnecessary words in the bug report. This is done by the process known as bug Pre-processing.

1) ***Bug Pre-processing:***This module is used in order to remove stop words from the bug description. The stop words used in this project are standard words given in the web mining forums. Some of the stop words are of, the, anything, almost, alone etc.

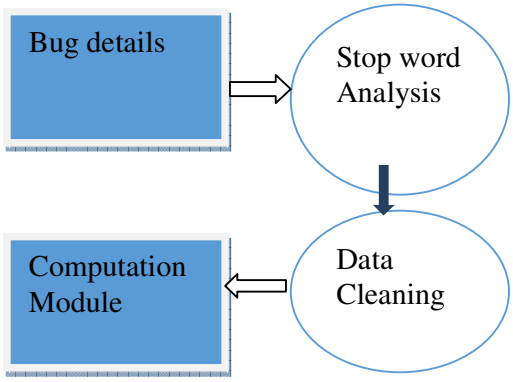

Fig. 1Collection and Cleaning process

C. ***Computation***

1) ***Tokenization:*** Tokenization is a process of translating the clean bug set into a set of sequenced tokens. Each token is associated with a bug in the form { TokenId, TokenName, BugId, ProductId }

2) ***Frequency Computation:***It is defined as the number of times a token appears in the review. The frequency will remove if any redundancy exists. The frequency is stored in the format {FreqId, TokenName, Freq, BugId, ProductId}

3) ***Score Computation:***The Score computation is performed on per token and is computed across the bugs by using the formula.

{ ScoreId ,Frequency, IDF, Score, BugId, ProductId).

4) ***Textual similarity:***The textual similarity is computed by comparing 2 bugs with respect to title and description

.

D. ***Classification***

1) ***Duplicate Bug Detection:***The algorithm is used to detect the whether the 2 bugs are similar or not. The algorithm finds the intersection sum and union sum and then the bugs are found in terms of grouping.

2) ***Naïve Bayes Text Classification:***This process is used to find the text classification of the bugs. The algorithm computes the probability that a bug belongs to a class {c1,c2,c3,c4}.Each class represents the unique feature then the algorithm computed contingency and enhanced contingency and classifies the bugs.

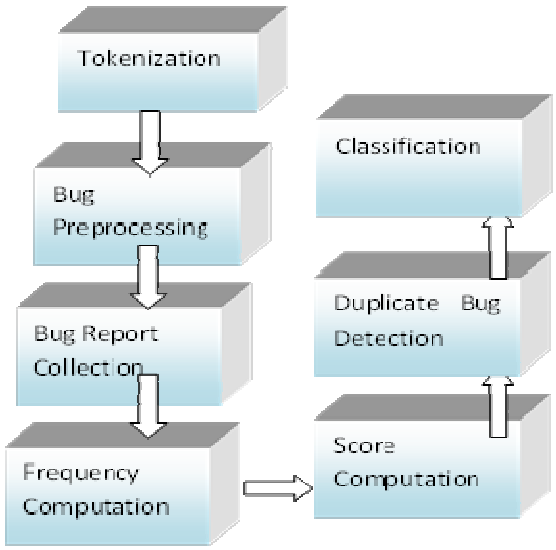

Fig. 2 Methodology Overview

v. SYSTEM DESIGN

A. BM25F algorithm.

BM25 (BM stands for Best Matching) is a ranking function used in search engines, to rank matching documents based on their relevance to a given search query. It is worked (based) on the probabilistic retrieval framework.

BM25 is retrieval function that gives ranking to set of documents based on query items which appears on each document, the interrelationship between query items doesn't have much significance in this algorithm.BM25 is a combination function i.e. it is family of scoring function with slight variation in components and parameters.

The following are the steps for the algorithm

- ✓ Collect the Bugs
- ✓ Perform the Data Cleaning on the Bugs
- ✓ Perform the tokenization and convert the clean bugs into set of tokens
- ✓ Measure the Frequency
- ✓ Compute the Score using the following equations for the bugs

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

f = *frequency*

IDF = *Inverse Document Frequency*

D = *length of document*

$avgdl$ = average document length in the text collection

k1 = 1.2

b = 0.75 $IDF(qi)$

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

$N = number\ of\ documents$

$n(q_i) = number\ of\ documents\ containing\ q_i$

- ✓ The textual similarity is computed by comparing 2 bugs with respect to title and description

Textual $_{title}(d_1, d_2)$=BM25F(d_1,d_2)

Where,

d_1=document1

d_2=document2

- ✓ The union set between the bugs is measured
- ✓ The intersection set between the bugs is measured

- ✓ The similarity score is also computed.

B. Naïve Bayes classifier.

It is simple technique used for constructing class of classifiers.

Naive Bayes classifiers can perform operation on an arbitrary number of independent variables whether they are continuous or categorical.

The advantage of this technique is only finite amount of training data is required to perform classification operation.

1) Naïve Bayes Classifier

- ✓ Obtains the bugs from the collection
- ✓ For each of the bugs the probability is computed using the following formula

$$P(b \mid C_i) = \frac{Number\ \ of\ \ words\ \ of\ \ category\ \ C_i}{Total\ \ Number\ \ of\ \ Words}$$

$$1 \le C_i \le 4$$

- ✓ Also the negative probability is also computed for each of the bug
- ✓ The probability computation is computed and constructed as below
  {Probability, bugID, CatName, Negative Probability, Count, Total Words }

Where,

BugID -ID of the Bug
Probability- Positive Probability
CatName - c1,c2,c3 and c4
Negative Probability - Finding the negative probability
Count- Number of words for the category
Total Words- Number of words

- ✓ The contingency is measure using the following

$$Total\ \ Positive\ Other_{c1} = p(C2) + p(c3) + p(c4)$$

$$Total\ Negative\ Other_{c1} = p^1(c2) + p^1(c3) + p^1(c4)$$

- ✓ The enhanced contingency is measured using the following equation

$$Positive\ Category\ Ratio_{c1} = p(c1) + Total\ Negative\ Other_{c1}$$

$$Other\ Category\ Ratio_{c1} = p^1(c1) + Total\ \ Positive\ Other_{c1}$$

- ✓ The bugs are then classified by order by positive category ratio maximum and other category ratio minimum
- ✓ The count for each category bugs is then made.

VI. RESULTS

Using the methods described in the previous section the results obtained are discussed in this section.

Bugs from several standard products like Mozilla, Firefox are collected in this project.

Below figure shows the details about the number of bugs obtained from product Firefox (fig. 3).

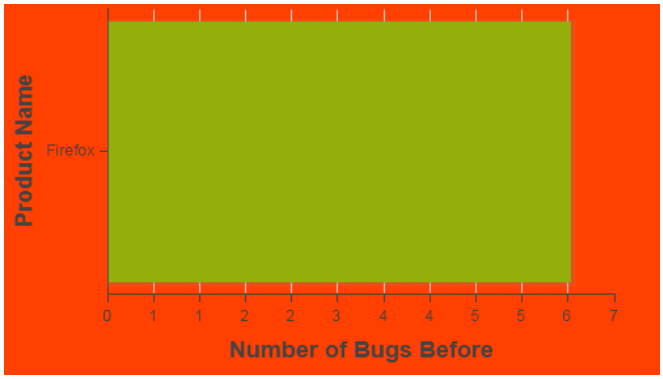

Fig.3 Number of raw bugs before processing

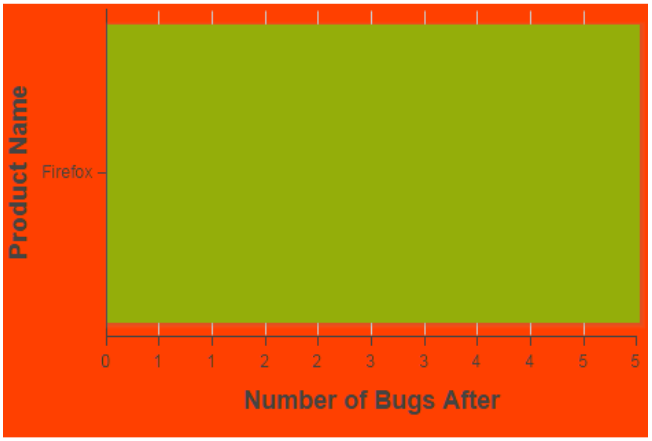

Fig. 4 Number of bugs after all methodology processes

After performing various methodologies described in this projects the number of projects are decreased. Here scalability and quality of the bugs are considered as a criterion (fig. 4).

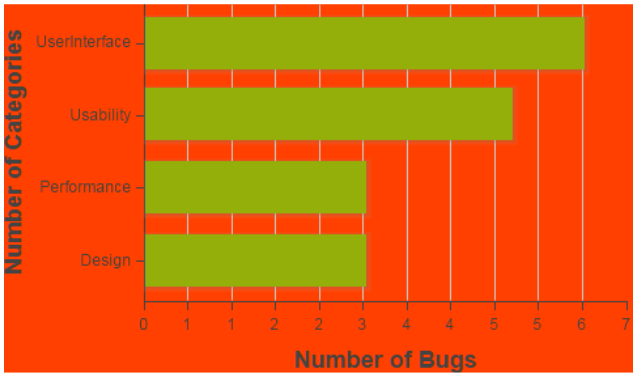

Fig. 5 After processing bugs is assigned to their respective categories.

In this project bugs are assigned to the respective category (fig. 5).so that bugs are given to the intended developer to fix the bug.

CONCLUSION

In this project we have first taken bugs from standard products like Mozilla, Firefox etc. and applied series of algorithms like data cleaning, tokenization, frequency computation, Score computation and duplicate bug detection algorithm.

The bugs are also grouped into various layers by computing the probability, contingency and enhanced contingency and finally applying the classifier.

After the classification process, bugs are given to their respective developer. Hence time and cost factor are considered here.

ACKNOWLEDGMENT

Every project is successful due to the effort of a number of people who have always given their valuable advice. We sincerely appreciate their valuable support and guidance of all those who have been there in making this project a success.

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] 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.

[3] 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, Bugs 10, Aug. 2011