Case Study on Incorporating Clustering on Reduced Training Bug Data for Effective Bug Triaging

Dhanya S
M.tech, Computer & Information Science
SIST, Thiruvananthapuram

Ms. Resmy V R
Assistant Professor
SIST, Thiruvananthapuram

***Abstract*— Many open source projects for example Eclipse, Aspectj and Firefox deploy open source bug repositories to handle or manage the bugs. Users, developers and testers can report bugs to these repositories. In a bug repository, a bug is maintained as a bug report, which records title and description as textual content, and various attributes like status, product, component, version, etc. One typical bug repository is Bugzilla. Software companies spend over 45 percent of cost in dealing with software bugs. An inevitable step of fixing bugs is bug triage, which aims to correctly assign a developer to a new bug. In traditional software development, bug triaging was done manually, which is expensive in terms of time and cost due to the large number of daily bugs and the lack of expertise for all the 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. Two typical characteristics of low-quality bugs are noise and redundancy. Noisy bugs may mislead related developers while redundant bugs waste the limited time of bug handling. The data set reduction can be achieved by applying feature selection and instance selection in bug repositories. Here instances are considered as bug report and features as word in the report. To decrease the time cost in manual work, text classification techniques are applied to conduct automatic bug triage. Clustering the reduced set of training bug data, would group similar bug reports. The clustering of similar bugs will enhance the bug fixing step once a bug has been identified as meaningful.**

Index Terms—Bug Triage, Bug Repository, Naïve Bayes.

I. INTRODUCTION

Data mining for software engineering consists of collecting software engineering data, extracting some knowledge from it. It facilitates the usage of this knowledge to improve the quality of software. In modern software development, software repositories can be considered as large scale databases for storing the output of software development like source code, bugs, emails and other specifications. Data mining techniques like frequent pattern matching, classification, clustering etc can be used to mine the software repositories so as to uncover the hidden and interesting information and can be applied for solving real world problems. A typical repository, for storing details of bug is called bug repository. It plays an important role in handling software bugs. Large software projects deploy bug repositories (also called bug tracking systems) to store the bugs and to assist developers to handle bugs. In a bug repository, a bug is maintained as a bug report, which records title and description as textual content, and various attributes like status, product, component, version, etc. If we take an open source project like eclipse, an average of 333,371 bugs are reported to bug repository from 2001 to 2010 from over 34,917 developers and users. It is a fact that, software companies spend over a 45 percent of cost in dealing with software bugs. Dealing with bugs implies fixing those bugs.

An inevitable step of bug fixing is bug triage, which is a task of assigning correct developer to a new bug. One problem while handling the repositories is the large variability in the formats and tools needed, standards, etc. that make the data gathering process a very labor intensive one. One example is the mining of textual data to deal with bugs for classification, clustering, etc.

Some of the repositories such as the BTS, are composed of a large number of attributes, however, many of those attributes are missing values that need to be discarded in order to apply machine learning algorithms.

The existence of irrelevant and redundant features in the datasets has a negative impact in most data mining algorithms, which assume a certain level of balance between the classes attributes. Also, when dealing with classification, there could be a problem of overlapping between classes, in which a region of the data space contains samples from different values for the class, thereby making the induction of good predictive models difficult. Understanding the task of bug triage requires knowledge about open bug repositories, Bugzilla report and lifecycle of a bug**.**

The remaining section of the paper is organized as follows: Section II describes related works and the summary. Section III discusses the problem statement. Section IV describes objective of the proposed work. Section V discusses the working of proposed system. Section VI tells about the methodology including algorithms to be implemented. Section VII describes the general architectural design of the system.

II. RELATED WORK

*Davor Čubranić, Gail C. Murphy, "Automatic bug triage using text categorization" [2004].*This paper proposed a method of applying machine learning techniques to assist in bug triage by using text categorization to predict the developer for the bug reported based on the bug's description. The data set was divided into a test set and training set by randomly selecting a percentage of bug reports from the data

set for placing into the training set, with the rest going to the test set.

The model was learned using a *Naive Bayes* classifier to automatically assign bug reports to developers. **Gaeul Jeong , Sunghun Kim, Thomas Zimmermann," Improving Bug Triage with Bug Tossing Graphs", 2009 ACM** , This paper proposed graph model based on Markov chains, which captures bug tossing history. The authors generated tossing graphs from training set of bug reports and utilize the graph for prediction. Using the tossing history in their model revealed developer networks which can be used to discover team structures and to find suitable experts for a new task and it helped to better assign developers to bug reports. **D.Matter, A.Kuhn, and O.Nierstrasz,"Assigning Bug Reports using a Vocabulary based Exprtise Model of Developers" (2009).** This paper modeled a developer's expertise using the vocabulary found in the developer's source code. The system recommended potential developers by extracting information from new bug reports and looking it up in the vocabulary. **Weiqin Zou, Jifeng Xuan, HeJiang "Toward Training Set Reduction for Bug Triage", Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual Conference.** In this paper the authors has proposed training set reduction with both feature selection and instance selection techniques for bug triage and evaluated the training set reduction on bug data of Eclipse.

a) SUMMARY OF LITERATURE REVIEW

The Machine learning techniques for prediction or recommendation purposes has found that prediction accuracy depends on the choice of classifier, i.e., a certain classifier outperforms other classifiers for a specific kind of a problem. While classifiers and tossing graphs are effective in improving the prediction accuracy for assignment and reducing tossing path lengths, their accuracy is threatened by several issues: outdated training sets, inactive developers, and imprecise, single-attribute tossing graphs. The classifier recommends a set of potential developers, and for each potential developer, a tossing graph – whose edges contain tossing probabilities among developers is used to predict possible re-assignees. However, the tossing probability alone is insufficient for recommending the most competent active developer. In particular, in open source projects it is difficult to keep track of active developers and their expertise. To address this, in addition to tossing probabilities, it is possible to label tossing graph edges with developer expertise and tossing graph nodes with developer activity, which help reduce tossing path lengths significantly. However, large scale and noisyredundant bug data block the techniques of automatic bug triage.

III. PROBLEM STATEMENT

From the above survey it is noticed that there are two challenges that may affect the bug repositories in software development task, namely large scale and the low quality. Two typical characteristics of low quality bugs are noise and redundancy. Noisy bugs may mislead related developers and redundant bugs waste the limited time of bug handling. Data reduction can be done along the bug dimension and the word dimension. A binary classifier is built to predict the reduction order, which will increase the speed and accuracy of classification. Here the attributes selected for prediction are mostly statistical values, e.g., the number of words or length of bug reports etc and should be extracted before the bug triage process. No representative words in the bug data sets are extracted as attributes

IV. OBJECTIVE

In this paper, it is intended to improve the prediction accuracy by selecting more relevant attributes from the bug dataset. Along with labeling the document with the developer name, i.e. applying text categorization, it is possible to group the new bug data based on the similarity to the nearest training set cluster. The clustering will reveal the structure of characteristics bugs so that it would be more helpful in fixing of the bug.

V. PROPOSED SYSTEM

The paper focuses mainly on accurate and speedy bug triage process by correctly assigning an expert developer when a new bug arrives. The system uses a collection of bug reported over a period of one month from an open bug repository like Eclipse or Aspectj. Each of the reported bugs has several attributes value pairs. The main task is to identify the relevant attributes only. Since bug reports are usually submitted as free form text along with the predefined fields, it is a fact that the most relevant information about a particular bug can be obtained from its title and description. Bug reports are collected with STATUS field having a value either open or verified and the resolution field with value as FIXED only. Bug reports that has FIXED as resolution can be used as the training data set. The remaining reports are referred to as test set. Thus the entire collection of bug data set is divided in to training and test sets.

The system assumes bug reports as instances and words in the description and summary as features. Since the training set consists of redundant and noisy bug data, it is important to do feature selection as well as instance selection for selecting relevant features and bug reports respectively. For that, the main content bearing attributes like summary and description are extracted and are represented as a text matrix (Bug X Word). Then the words and the frequency of occurrence of those words in the textual contents are identified. The words are then compared against a stop list to find the irrelevant words. A predefined set of stop words are available in SMART database. The identified stop words are thus removed reduced the number of words in the summary and description by almost 30%, facilitating a fast and accurate triaging. Still, the report might contain irrelevant words. So, to remove such words rank or index the words according to relevance by using a feature selection algorithm named CHI square. The system also performs a clustering on the entire set of training bug data, in order to group similar bug reports .The clustering of similar bugs will enhance the bug fixing step once a bug has been identified as meaningful. The clustering also reveals a structure within in each cluster based on the similarity matrix. My thesis work mainly focus on automatic

bug triage using machine learning approach. Here the system adopts a probabilistic text classification algorithm called Naive Byes algorithm to classify a new bug to an already defined label, particularly a developer name. When a new bug arrives, the system generates a score to each developer based on the text similarity. And assign the newly reported bug to a developer having highest score.

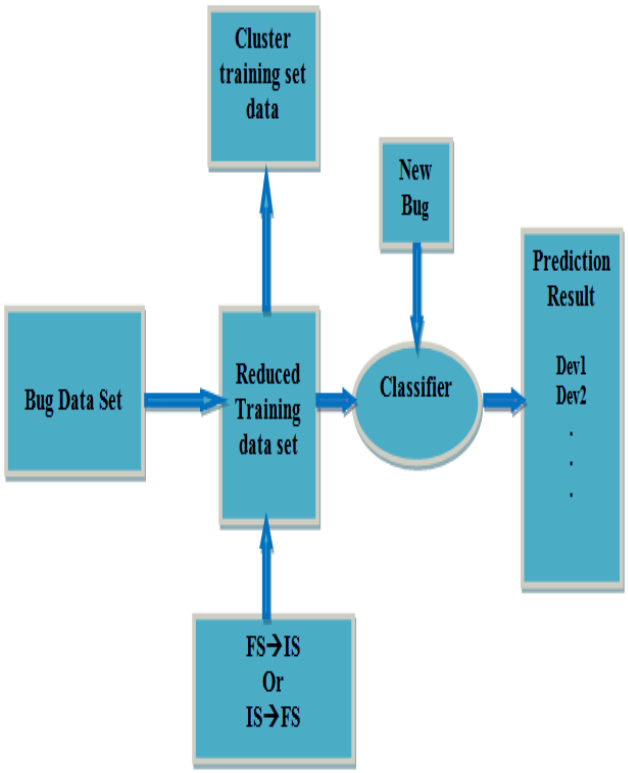


Fig 2.1 Block diagram specifying working of the system

VI. METHODOLOGY

Bug Triaging is an inevitable step in bug fixing which involves the task of assigning correct developer to a new bug there by facilitating a speedy bug triaging. The data preparation requires a continuous set of bugs from an open bug repository of large open source project like Eclipse or Mozilla over a period of time. From the entire dataset select those bug reports which have their status either FIXED or DUPLICATE, since bug triage aims to predict the developers who can fix the bugs. The data set reduction can be achieved by applying feature selection and instance selection in bug repositories. Here instances are considered as bug report and features as word in the report. Feature selection can be implemented using $\chi 2$ statistics (CHI).

- Generate developer list & word list
- Converting to a 2-D text matrix
- Generate word set for each developer
- Training set Reduction
- Clustering
- Assigning a developer

i. Data Collection and Preparation

Collect continuous bug reports from an open bug repository (Bugzilla, JIRA) etc, for any open source project like Eclipse, Mozilla, Aspectj etc over a period of say one or two months. For each bug report extract details of bug report. Choose bug reports with status either FIXED or DUPLICATE as Training set. The remaining bug reports are considered as test data set. The bug reports could be in XML, CSV or ARFF format.

ii. Generate Developer List

From the training set generate corresponding label, i.e, developer names. Thus for each bug reported we have a developer also.

iii. Pre-Processing

Extract the most relevant attribute like summary and description of the bug report. Usually these attributes are textual contents bearing relevant information about the bug. Thus the textual contents needed some sort of pre processing which includes tokenization and stop word removal.

Tokenization is a process in which it simply segregates all the words, numbers, and their characters from a given document. These identified words, numbers, and other characters are termed as tokens. Along with token generation this process also computes the frequency of occurrence of all these tokens present in the input documents. Stop words are those words of no relevance. A stored list of stop words is available in a data base called SMART. The tokens thus generated are compared against the stop list and identified similar words are removed from the document. T his process reduces the size of document by 30%.

Consider an example: *Please add org.aspectj:aspectjrt alias to your package. Currently all packages which require org.aspectj:aspectjrt need to be patched.*

a) Tokenize+stopword removal

Please:1 add:1 org:2 aspectj:2 aspectjrt:2 alias:1 package:2 currently:1 all:1 require:1 need:1 patched:1

iv. Converting To 2-D Text Matrix (Bug X Word)

After preprocessing, we have a reduced set of words corresponding to each bug report. The next step is to build a two-dimensional text matrix

Words ↓

w1	w2	w3		
.	.	.		
.	.	.		
.	.	.		

Bugs →

Fig 3.1 Text Matrix

v. Generate Word Set for Each Developer

There could be a developer who had fixed similar bugs. So identify such bugs and extract or group the words of bugs for that developer. Like this generate word set for each developer in the training set.

vi. Training Set Reduction

*a)**Feature Selection***

Even after the stop word elimination, the document might contain some other non- relevant words. Feature selection uses a statistical method called Chi-Square (CHI) to rank the words in the document according to their relevance. Also, remove bug reports having Status DUPLICATE to reduce the size of training set.

a1) Feature Selection using CHI Square

CHI is a statistical method used to test the independence of two events, say occurrence of terms & occurrence of class. In order to rank the terms use the equation $\chi 2 = \sum_{e_t \in \{0,1\}, e_c \in \{0,1\}} \sum \frac{(N_{e_c e_t} - E_{e_c e_t})^2}{E_{e_c e_t}}$ where e_t = 1 (vocabulary contains term t) e_c= 1(word is in class c). If **N** is the observed frequency of words and **E** is the expected frequency of words then, E = N*P(t)*P(c). Also, $E_{11} = N * \frac{(N_{11}+N_{10})}{N} * \frac{(N_{11}+N_{01})}{N}$, which gives the expected frequency of ***t*** and ***c*** occuring together.

Arithmetically, we can write

$$\chi^2(V,t,c) = \frac{(N_{11}+N_{10}+N_{01}+N_{00}) * (N_{11}N_{00} - N_{10}{N_{01}}^2)}{(N_{11}+N_{01})(N_{11}+N_{10})(N_{10}+N_{00})(N_{01}+N_{00})}$$

b) Algorithm for Feature Selection

Chi-square (curr_class,training set,no: of desired features)
For i=1 to no: of words
For j=1 to no: of training data
If j_{th} bug belong to curr_class then
Observed freq (O_{ij})= O_{ij}+ freq of word i in j_{th} bug
End if
Create contingency table for i^{th} word
Compute expected freq (E_{ij})
$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{1} \frac{(O_{ij}-E_{ij})^2}{E_{ij}}$
End for
Sort χ^2 values
Best feature=feature with highest χ^2 value

vii. Text Categorization

In text classification, we are given a description $d \in$ **X** of a document, where **X** is the *document space*; and a fixed set of *classes* **C** = {*c* 1, *c*2, . . . , *cJ*}. Classes are also called *categories* or *labels*. Typically, the document space **X** is some type of high-dimensional space, and the classes are human defined for the needs of an application. We are given a *training set* **D** of labeled documents <*d*,*c*>, where <*d*, *c*> ∈ **X** × **C**. Using a *learning method* or *learning algorithm*, we then wish to learn a Classifier or *classification function g* that maps documents to classes: $g : \mathbf{X} \rightarrow \mathbf{C}$. This type of learning is called *supervised learning* because a supervisor (the human who defines the classes and labels training documents) serves as a teacher directing the learning process.

a) Naive Bayes Text Classification

The first supervised learning method introduced is multinomial *Naïve Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document *d* being in class *c* is computed as P(c|d)=$\frac{P(d|c)P(c)}{P(d)}$ or simply P(d|c)*P(c),since P(d) is negligible, it can be neglected. Here in this work, text categorization relies on Bag of Words representation of bug reports. To find the most likely class i.e. developer suitable for fixing the bug, use the equation, $C_{MAP} = argmax_{c \in C} P(d|c)P(c)$

Assuming the conditional independence between the terms $(t_1, \ldots, t_n)$ and developer *(c)*, we have $P(t_1, \ldots, t_n|c) = P(t_1|c)P(t_2|c) \ldots \ldots P(t_n|c)$. Also, to find the best developer using Naïve Bayes, C_{NB}, find the maximum prior probability for each of the posterior probability of terms in bug report using the equation,

C_{NB}=argmax$_{c \in C} P(c_j) \Pi_{t \in V} P(t_j|c)$.

a1) Learning the Multinomial Naïve Bayes Model

The Prior Probability can be estimated using $P(c) = \frac{N_c}{N}$, where N_c=number of documents labeled as c, N = total number of documents

Conditional probability can be defined as the fraction of times word appears among all words in bug reports of class developer(c). It can be computed using

$$P(t_k|c) = \frac{Count(t_k, c)}{\sum_{t \in V} count(w, c)}$$

The best class can be found using $C_{MAP} = argmax_{c \in C} P(t_1, t_2, \ldots.)|c).P(c)$

b) Algorithm for Classifying a New Bug

Input: A new vocabulary v (words in description of new bug)

A fixed set of classes C = {c_1, c_2 ...c_J}

A training set of labeled set (v_1, c_1)... (v_m,c_m)

Output: a learned classifier γ: v→c

Method:

Extract vocabulary from training bug dataset

For each Assigned-To attribute in the training set

i. Find prior probability (p) for each term in the vocabulary

ii. Find conditional probability cp=p + cp

Return maximum value of cp and assign class label

viii. Cluster the Data Set

Cluster the bug reports in the training dataset. For clustering I would like to use the k-means clustering algorithm

a) K means Algorithm

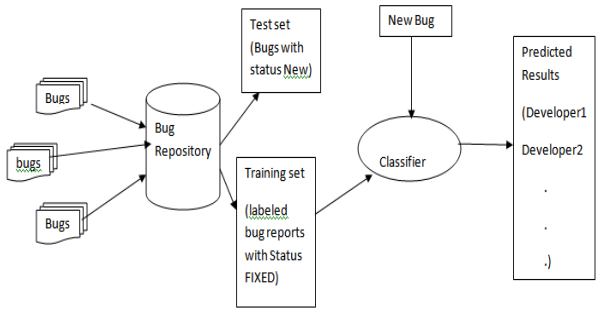
1. Specify the no: of clusters
2. Randomly select k reports and place one of them in each cluster
3. Place the remaining reports in the clusters based on similarity between other reports and the selected ones in each cluster
4. Compute centroid for each cluster
5. Again find similarity between centroids and the reports
6. Re-compute the centroid and replace the reports and reassign based on similarity
7. Repeat the process until similar reports falls in one cluster

ix. Prediction of New Bug

When a new bug arrives, using text categorization, label the report with a correct developer and find a closest group using the K-means clustering algorithm to put the new bug in that cluster.

VII. SYSTEM ARCHITECTURE

General architectural design focuses on the components or elements of structured system and unifies them in to a coherent and functional unit. In the proposed work, first a training bug data set is created from the available bug dataset, which contains predefined labels (developer name)for each existing bug in the repository suggesting the developers who are capable of fixing those bugs. Thus the remaining data set can be termed as test set. The bugs with status as unresolved or new in the test set could be assigned to a developer based on these predefined labels.



3.3 Working of the System

The thesis work is based on machine learning approach. The initial task is to get the data from an open bug repository like *Bugzilla* and divide the entire dataset into training and test set. The training set consists of all the bug data with the status resolution FIXED. All the remaining data constitute test data. Since the relevant information about the bug reports are maintained as textual contents (summary and description), apply the general pre-processing techniques like stop word removal, tokenization etc. Also, generate developer names from the training set and represent each developer with a set of word in the bug reports they had handled. The main problem addressed here is the reduction of noisy and redundant bug data. This could be achieved by selecting relevant features (words) and relevant instances (bug reports).

When a new bug arrives, perform all pre-processing on the summary and description and represent the bug as feature vectors. Then apply the text categorization method like Naive Bayesian in order to label the new bug with a correct developer. This could be made possible by learning the already labeled trained data using a classifier. The next step is to cluster the training bug data set based on their similarity. It is possible to group the bug reports based on reporter, priority developer who has handled similar bugs etc which will feature out the structure of each cluster.

VII. CONCLUSION

The paper is based on machine learning approach. The initial task is to get the data from an open bug repository like *Bugzilla* and divide the entire dataset into training and test set. The training set consists of all the bug data with the status resolution FIXED. All the remaining data constitute test data. Since the relevant information about the bug reports are maintained as textual contents (summary and description), apply the general pre-processing techniques like stop word removal, tokenization etc. Also, generate developer names from the training set and represent each developer with a set of word in the bug reports they had handled. The main problem addressed here is the reduction of noisy and redundant bug data. This could be achieved by selecting relevant features (words) and relevant instances (bug reports).

When a new bug arrives, perform all pre-processing on the summary and description and represent the bug as feature

vectors. Then apply the text categorization method like Naive Bayesian in order to label the new bug with a correct developer. This could be made possible by learning the already labeled trained data using a classifier. The next step is to cluster the training bug data set based on their similarity. It is possible to group the bug reports based on reporter, priority developer who has handled similar bugs etc which will feature out the structure of each cluster.

REFERENCES

[1] Jifeng Xuan, He Jiang, Member, IEEE, Yan Hu, Zhilei Ren, Weiqin Zou,Zhongxuan Luo, and Xindong Wu,Fellow, IEEE, *"Towards Effective Bug Triage with Software Data Reduction Techniques"*, IEEE TRANSACTIONS ON KNOWLEDE AND ENGINEERING, Vol 27[2015].

[2] Anjali, Sandeep Kumar Singh ,Department of Computer Science and Engineering, Jaypee Institute of Information Technology, *"Bug Triaging: Profile Oriented Developer Recommendation"*, International Journal of Innovative Research in Advanced Engineering (IJIRAE) ,Volume 2 Issue 1 [2015].

[3] W. Zou, Y. Hu, J. Xuan, and H. Jiang, *"Towards training set reduction for bug triage,"* in Proc. 35th Annu. IEEE Int. Comput. Soft.Appl. Conf.[2011].

[4] P.Bhattacharya and I.Neamtiy, *"Fine Grained incremental learning and multi feature tossing graphs to improve bug triaging"*, Proc. IEEE Intl. Conf. Software Maintainence, [2010]

[5] D. Matter, A. Kuhn, and O. Nierstrasz, *"Assigning bug reports using a vocabulary-based expertise model of developers,"* Proc. IEEE Working Conf. Mining Software Repositories, IEEE Computer Society[2010].

[6] 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[2011]

[7] 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].

[8] Veena Jadhav et al, *"Prediction by Using Cos-triage Algorithm"*, (IJCSIT) International Journal of Computer Science and Information Technologies,[2014].

[9] J.W. Park, M.W.Lee, J.Kim, S.Won Hwang and S.Kim, *"Costriage:A cost aware triage algorithm for bug reporting systems"*, in AAAI[2011].

[10] 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[2011].

[11] G.Jeong, S.Kim and T.Zimmermann,"*Improving Bug Triag with Tossing Graph"*, Proc.Joint Meeting European Software Engineering Conf. & ACM SIGSOFT Symp [2009].

[12] A. E. Hassan, "*The road ahead for mining software repositories,"*in Proc. Front. Softw. Maintenance,[2008].

[13] Katja Kevic, Sebastian C. M¨uller, Thomas Fritz, and Harald C. Gall ,Department of Informatics University of Zurich, Switzerland, *"Collaborative Bug Triaging using Textual Similarities and Change Set Analysis"*.

[14] J.Anvik, "*Automating bug report assignment"*, Proc.Intl. Conf. Software Engineering (ICSE 06), ACM [2006].

[15] J. Anvik, L. Hiew, and G. C. Murphy, *"Who should fix this bug?"*, Proc. Intl. Conf. Software Engineering (ICSE 06), ACM [2006].

[16] D.Čubranić and G. C. Murphy, *"Automatic bug triage using text classification"*, In Proceedings of Software Engineering and Knowledge Engineering [2004].

[17] Pamela Bhattacharya, Iulian Neamtiu, Christian R. Shelton, "The Journal of Systems and Software 85, *"Automated, highly-accurate, bug assignment using machine learning and tossing graphs" [*2012]

[18] Y. Yang, "*An evaluation of statistical approaches to text categorization"*, Information Retrieval,[1999].

[19] Y. Yang and J. Pedersen, *"A comparative study on feature selection in text categorization,"* in Proc. Int. Conf. Mach. Learn.[1997].

[20] K P Soman,Shyam Diwakar,V Ajay, *"Insight into Data Mining Theory and Practice"*.

[21] Charu C Aggarwal, IBM T. J. Watson Research Centre Yorktown Heights, NY charu@us.ibm.com, ChengXiang Zhai University of Illinois at Urbana-Champaign Urbana , *"A Survey Of Text Classification & Clustering Algorithms*" , (chapter 4) .

[22] Data Ronen Feldman Bar-Ilan University, Israel James Sanger ABS Ventures,Waltham,Massachusetts *"TheTextMiningHandbook:Advanced Approaches in Analyzing Unstructured Data"*.

[23] Jiawei Han, Micheline Kamber, Jin Pei, *"Data Mining Concepts and Techniques"-3rd Edition*

[24] Bugzilla, http://www.bugzilla.org/