IJSTE - International Journal of Science Technology & Engineering | Volume 2 | Issue 04 | October 2015
ISSN (online): 2349-784X

Implementing Bug Severity Prediction through Information Mining using KNN Classifier

Pankaj Rana
M. Tech. Student
Department of Computer Science & Engineering
Sri Sai University, Palampur

Saurabh Sharma
Assistant Professor
Department of Computer Science & Engineering
Sri Sai University, Palampur

Abstract

The severity of a according bug may be crucial think about deciding however presently it has to be fastened. Correct severity assessment is crucial for acceptable resource allocation and coming up with for fixing activities and extra testing. Severity assessment is powerfully influenced by the expertise of the take a look at engineers and by the time they pay on every issue. Code bug coverage is an integral a part of code development method. Once the bug is according on bug following system, their attributes analyzed and later on allocated to numerous fixers for his or her resolution. Bug severity, an attribute of a code bug report is that the degree of impact that a defect has on the event or operation of a element or system. Bug severity is classified into totally different levels supported their impact on the system. Prioritization of bugs decides the bug fix sequence. Incorrect prioritization of bugs leads to delay of break down the necessary bugs, that leads delay in unharness of the code. Prediction of bug priority wants historical knowledge on that we will train the classifiers. With the increasing dependence on code systems, importance of code quality is turning into additional crucial. There are other ways to make sure quality in code like code reviews and accurate testing in order that bugs is removed as early as attainable to stop the loss it should cause. "Every code program is rarely excellent, there's forever a minimum of one bug in it which may be encountered at any time." Software bug is usually wont to describe the prevalence of a fault in an exceedingly package which ends up it to act otherwise from its specification. Its encountered whereas operative the products either below take a look at or whereas in use.

Keywords: Bug Severity, Bug Repositories, Bug Priority, Severity Prediction, Natural Language Processing, Machine Learning

I. INTRODUCTION

This chapter discusses the effect of bugs on software quality and classification of it based on severity. Further bug triaging and Bug tracking system are discussed.

Then life cycle of bug in bug tracking system and benefits of bug tracking are described. With the increasing dependence on software systems, importance of software quality is becoming more critical. There are different ways to ensure quality in software such as code reviews and rigorous testing so that bugs can be removed as early as possible to prevent the loss it may cause. There is an old saying, "Every software program is never perfect, there is always at least one bug in it which can be encountered at any time. "Software bug is commonly used to describe the occurrence of a fault in a software system which results it to act differently from its specification [1]. It is encountered while operating the product either under test or while in use.

Bugs are mostly mistakes which originate due to human participation. 57% bug originates from error made by human, which could be either due to carelessness or absent mindedness [2]. When they lead to software failure these bugs can cost companies a big amount of money and in some case loss of human lives e.g. software bug in the a Royal Air Force Chinook aircraft's engine control computer caused it to crash in the year 1994 and 29 people were killed[3]. So early detection of bugs and their resolution is very critical.

A. The Bug Life Cycle in Bug Tracking System:

A cycle in which bug goes through during its lifetime is called Bug life cycle. The cycle starts when bug is reported and ends when that bug is closed and never reproduced. There are various stages of bug in its life cycle. However these stages are different in different bug tracking system.

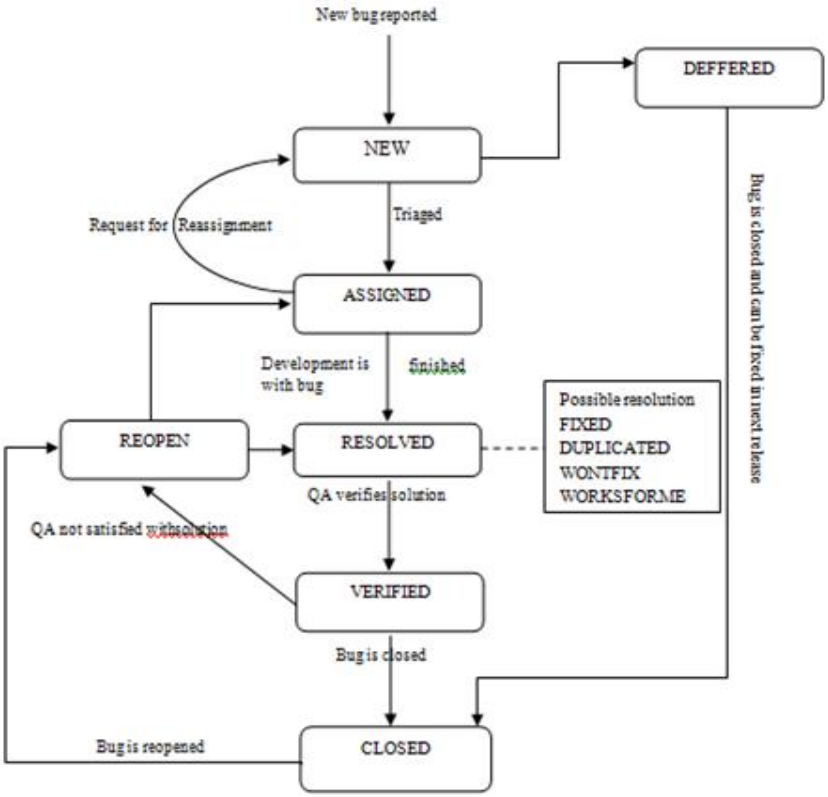

Fig. 1: General Bug life cycle

B. Motivation:

A large number of bug reports are reported in bug tracking system by users in open software system that is encountered while operating the software. These bugs are analyzed to find there validity, correctness, importance, severity and also to verify its duplicity. These reported bugs are generally higher in number so it becomes difficult to resolve them on time. Therefore developer has to make choice among all reported bugs. Bug severity is one of the important factors which can be used to identify bugs that need immediate attention to resolve. It helps bug triager to triage the bugs. Triager is person who uses his knowledge and experience to analyze and refine the bug that is reported. But it is very tedious job to manually assign severity and time consuming and also accuracy of classification depends on knowledge and experience of triager who analyze the bug. Moreover such manual effort is costly in labour expenditure. There has been a growing need of automating the whole process of severity prediction of bugs which will surely help in making bug triaging process more efficient and less time consuming. Therefore, we need to automate the task of classifying bugs based on severity. Bug reports come with textual description so to classify these reports text mining algorithms provide support. In past text mining techniques and machine learning algorithms have been applied by many researchers on the descriptions of bug reports for automation of the bug triaging process [10] and for detection of duplicate bug reports [11]. Attributes of bug reports (summary, severity, priority etc) are used to predict the bug severity automatically in literature. Combined approach of text mining and machine learning are used for the classification task on bug reports. But till now it is still far from reliable accuracy and there is still room for perfection. Thus an attempt is made in to automatically predict the bug severity classification with increased reliability and accuracy.

II. LITERATURE SURVEY

Bug triaging helps in deciding what to do with the reported bugs. These bugs may be reported during testing phase by tester or during operating the software by user. Due to extremely large number of bugs being reported in BTS, their classification during triaging is a tedious and time consuming process. Researchers have been for long time trying to automate the bug triaging task. Decent success has also been achieved. A list of such work is thoroughly discussed in this chapter as literature survey and followed by conclusion of it.

Davor Cubanic, Gail C. Murphy [12] made first attempt in 2004 and proposed to apply supervised machine learning algorithms to assists in bug triaging task. The approach predicts the developer to which the reported bug should be assigned. The proposed work was applied on 15,859 bug reports of Eclipse project. Text mining algorithms and Naïve bayes was used in the approach. The accuracy level 30 % was achieved.

Gaeul Jeong et al. [15] coined a new term called “tossed” (reassignment). Bug tossing signifies the reassignment of bug report to new developer. Authors introduced a tossed graph model using Markov chains. The experiment was performed on 450,000

bug reports of Eclipse and Mozilla. The result indicated that bug tossing activity was reduced to 72 %. Thus prediction accuracy was increased up to 23 % as compared to existing approach.

Israel Herraiz et al. [16] analyzed the bug reports of Eclipse. It was concluded that these bug report has too many options for severity and priority field. It was hard from user's point of view to distinguish them. The authors recommended making the bug report simpler than existing format. Severity levels could be reduced to three levels as important, non important and request for enhancement based on time taken to close the bug. Similarly priority field in bug reports were grouped according to mean time taken to close the bug. It was found that this field could also be classified into three levels i.e. high, medium, low.

Tim Menzies et al. [17] in their paper proposed a new automated method called SEVERITY ISSUE assessment (SEVERIS) to assign severity level. It was based on text mining approach and machine learning techniques. Authors conducted a case study using data sets of bug report of NASA's Project and Issue Tracking System (PITS) and applied SEVERIS on it. The result of case study indicated that SEVERIS was a good method of predicting issue severity levels.

Giuliano Antoniol et al. [18] presented an approach to create an automatic routing system that routed the real bug to maintenance team and request for enhancement to the team leader automatically. This approach considered 1800 issues from BTS of Eclipse, Mozilla and JBoss. Text mining technique was applied on description of report. The classifier was build using three supervised ML techniques like naïve bayes, decision trees and logistic regression. The performance of this approach was evaluated. It indicated that recall and precision level of Eclipse, Mozilla and JBoss was obtained between 33 % to 97 % and 64 % to 98% respectively.

Syed Nadeem et al. [19] suggested an approach to automate bug triage system that predicts the developer to bug reports. Bug reports sample of 1983 of Mozilla were used and feature selection and feature reduction method was applied on them. Then seven different machine learning methods were used for classification. The best result was obtained using latent semantic and SVM and 44.4 % accuracy level was achieved. The value of recall and precision of approach was 28 % and 30 % respectively.

Ahmed Lamkanfi and co-authors [20] proposed a new method for classifying bugs based on severity. Bug reports of Eclipse, GNOME and Mozilla were preprocessed using text mining algorithms (tokenization, stop word removal, stemming). Then machine learning classifier naïve bayes was applied. The average precision and recall of Eclipse and Mozilla was 0.65-0.75 respectively and 0.70-0.85 in case of GNOME.

Thomas Zimmerman et al. [21] conducted a survey among developers and user of APACHE, Mozilla and Eclipse. The result of survey indicated that there was mismatch between the information required by developer to the information provided by user. Further to overcome this mismatch authors proposed a new tool CUEZILLA that assess the quality of new bug report. This tool also provided the recommendation about the elements that should be added in bug report to improve the quality of it. The tool was trained on 289 bug reports samples and it calculated the quality of bug report 31- 48 % accurately.

John Anvik et al. [22] further extended previous work [13] [14] to create developer oriented recommender. This developer oriented decision recommender was used by four industrial triager and they found that recommendations generated were accurate. The approach was applied on historical datasets of five projects and 75 % accuracy level was achieved. high priority while bugs that fall in isolated region fall have low priority. An empirical study was performed considering crash reports of beta releases of Firefox. It was found that performance of proposed approach was better than existing technique of triaging these reports.

III. PROBLEM FORMULATION

From the extensive literature survey, it is concluded that early detection and classification of bugs is very critical for maintaining the quality of software. Bug tracking system helps developer and user to report the encountered bugs in open source software. These reported bugs need to be classified based on severity level as it helps to decide which bug need immediate fixation among all reported bugs.

However it is manual task and depends on expert's knowledge. In case of larger number of bug reports, it takes lot of time to classify bug reports. Therefore, there is need for designing a system which would automatically classify the bug reports as severe or non-severe based on their textual description.

Although, many attempts has been made at the problem and several machine learning and text mining algorithms have been applied for the same, a novel idea of using dictionary of critical terms (terms specifying severity level) for the prediction of severity level of reports is proposed in this thesis. KNN and Naïve Bayes are being exploited as classifiers with information gain and Chi square. Thus, the problem statement for the thesis is "Bug Severity Classification using Machine learning".

IV. METHODOLOGY

Bug reports are extracted from respective bug repository. Then preprocessing on textual information of bug reports is applied to obtain more reliable information. Term-document matrix (TDM) is created and by using feature selection method dictionary of critical terms is created. Then reduced TDM obtained by using critical dictionary terms is fed to classifier for classification of severe and non-severe bug report.

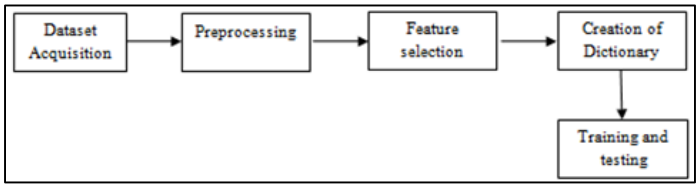

Fig. 2: Process

A. Dataset Acquisition:

The experiment considers bug report instances of Eclipse to perform classification of bugs based on severity. Eclipse is an open source integrated development environment (IDE). It is used worldwide by different developers to develop software. It is expected that bug reports of Eclipse should be of good quality [21] as users of Eclipse are developer itself and they will use technical terms for defining bug report. This will help in our study to create more accurate dictionary of terms for specifying bug severity level. Bug report instances of Eclipse are downloaded from Bugzilla [59] bug repository.

The instances have severity of type blocker, critical, Enhancement, major, minor, normal, trivial. The report instances that have normal and enhancement severity type are not considered in experiment. Enhancement bug reports do not correspond to real bug as these are requests for new features. Normal bug reports require manual inspection to assess the severity as these bug reports represents grey zone to classify bug report into severe or non severe category [20]. Also, in Bugzilla normal severity level is the default option and many reporters do not bother to assign severity level to bug report.

B. Pre-processing:

Pre-processing steps on textual summary of bug reports are performed. The reason for including summary as prediction of severity level is that in [24], authors concluded that summary of bug report gives better result than detail description of bug report. It includes tokenization, stop word removal and stemming.

1) Tokenization:

The purpose of tokenization is to remove all the punctuation marks like commas, full stop, hyphen and brackets. It divides the whole text into separate tokens to explore the words in document.

2) Stop word Removal:

The purpose of this process is used to eliminate conjunction, prepositions, articles and other frequent words such as adverbs, verbs and adjectives from textual data. The reason to remove these words is that these words are very common and rarely contain any important information. Thus it reduces textual data and system performance is increased. For example if a search engine is searching query “what is Text Mining” then search engine will find a lot of irrelevant page containing terms “what” and “is” , however “Text ” and “Mining” terms would help to find relevant web pages of text mining.

3) Stemming:

Stemming is used to reduce the words to their root words e.g. words like “computing”, “computed” and “computerize” has it root word “compute”. There are different algorithms to perform stemming such as Lovins Stemmer, Porters Stemmer, Paice/Husk Stemmer, Dawson Stemmer, N-Gram Stemmer, YASS Stemmerand HMM Stemmer. Porters stemming algorithm is considered as best stemming algorithm till now.

4) Term -Document matrix:

The Term- Document Matrix (TDM) is created after performing preprocessing steps. Each column in matrix represents the terms occurring in documents and row represents each bug report. The cells of matrix are filled with TF-IDF score. If term is not present in the particular bug reports then cell is filled with zero. Term Frequency- Inverse Document Frequency(TF-IDF) [47] score is generally is used to give weight to each term.

C. Feature Selection:

Feature selection methods are used to retrieve the most informative terms from corpus of matrix. In our research, we have used two feature selection methods info gain and CHI square methods. The detail descriptions of these methods are given in chapter 3 (section 3.2). These methods are applied on TDM matrix to reduce matrix.

1) Performance Metrics

Performance of selected classifier for our work is compared from the given metric described below:

- Precision: Precision is calculated as percentage of examples predicted as belong to class x that is actually correctly predicted.
- Accuracy: Accuracy is calculated as fraction of sum of correct classification to total number of classification.

V. RESULT AND COMPARISON

In this research, component specific dictionaries are created of four component of Eclipse. These dictionaries are created using top 125 terms using two feature selection methods; namely-info-gain and Chi square. The set of dictionary terms are then fed to

two widely used ML algorithms named Naïve Bayes and KNN for classification task and performance is analyzed in terms of precision and accuracy.

A. Probability based Classification:

The most well-known classification techniques that were used in early text classification systems were probabilistic ones. These techniques are based on assumption of conditional independence of terms occurrence. For probability based classification naïve bayes Multinomial classifier is used in this approach.

The accuracy varies form 69 % to 75 %, while precision of severe and non-severe category is found to be varying from 65-69 % and 77-84% respectively. The reason of less precision of the severe category than the non-severe ones could be attributed to the fact that the terms extracted by info gain has more bits of information than the severe class level in the datasets used in our experiments. In this approach of creating dictionary using info- gain and NBM classifier, the performance of UI component has been found to be the best.

Table – 1
Results of info-gain and Naïve Bayes multinomial

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	69.67%	66.77%	78.03%
Debug	71.66 %	69.18%	76.40%
Core	72.33%	68.69%	78.03%
UI	**75.38%**	**71.21%**	**84.52%**

The performance of four datasets after using Chi square method and naïve bayes multinomial classifier is shown below in table 5.2. The accuracy of this experiment lie between 68 % and 74 %. While precision of severe class ranges from 65 % to 70 %, the precision of the non-severe class is found to be between 77 % to 84 %.The similar pattern of performance is achieved in these datasets using Chi square as achieved in information gain. SWT component has least accuracy and UI component achieve more accuracy.

Table – 2
Results of Chi square and Naïve bayes multinomial

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	68.04%	65.19%	77.33%
Debug	70.73%	68.00%	76.37%
Core	71.23%	68.26%	75.60%
UI	**74.32%**	**69.92%**	**84.81%**

B. Similarity Based Classification:

The other classification technique used in text classification is based on similarity between the two documents. If the two documents have high similarity than these two have high possibility to be of one class. For similarity based classification KNN is used to retrieve its nearest neighbor for classification. It assigns majority class of K nearest neighbor to unlabeled document. The performance of this classifier using two approaches of feature selection is explained below:

1) Preparation of Dictionary Using Information Gain:

The result is obtained using dictionary formed by Chi square test and KNN classifier and shown below in table 5.3. The accuracy ranges from 87 % to 91 %. The precision of severe class and non-severe class is obtained as lying from 91 % to 96 % and 79 % to 90 % respectively. UI achieves maximum accuracy of 91 % using nearest neighbor classifier.

Table – 3
Results of info gain and KNN

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	87.51%	96.97%	79.62%
Core	85.13%	84.81%	85.47%
Debug	88.32%	91.07%	85.45%
UI	**91.16%**	**91.83%**	**90.39%**

2) Preparation of Dictionary using Chi Square Test:

The performance of classification task after using Chi square method and KNN classifier is shown below in table 5.4.

Table – 4
Results of Chi square & KNN

Component	Accuracy	Precision (Severe)	Precision (Non-Severe)
SWT	87.92%	97.39%	80.01%
Core	86.97%	86.39%	87.58%
Debug	88.91%	**92.32%**	85.49%
UI	**91.59%**	88.86%	**95.60%**

The accuracy of this experiment lie between 87 % and 91 %, precision of severe class ranges from 86 % to 92 % whereas precision of non-severe class has range from 85% to 95 %. The similar pattern of performance is achieved in these datasets using Chi square as achieved in information gain. SWT component has least accuracy and UI component achieve more accuracy.

Results show that four combinations of feature selection methods and classifier gave same pattern of accuracy. SWT achieves least and UI component achieves maximum performance in terms of accuracy and precision. However the performance of core and debug component show different pattern in both approaches. In probability based classification the debug component has higher accuracy than core component whereas core component has higher accuracy than debug component in similarity based approach. Also precision value of severe class in case of NBM classifier is less than non-severe class, but it was not the case with KNN classifier. The reason could be because of assumption of independence terms of NBM and there could be possibility of more dependence of severe terms as result in less precision of severe class.

C. Comparison of Probability and Similarity based Classification:

The comparison of accuracy level obtained using NBM and KNN using info- gain as feature selection method. The maximum accuracy obtained with KNN and NBM is 91 % and 75 % respectively.

The comparison of accuracy of NBM and KNN using Chi square for dictionary formation of different component is also shown in Figure 6.2.It is again found that KNN performs better than NBM in terms of accuracy. The minimum accuracy of NBM is 68 % and maximum accuracy is 74 % whereas minimum accuracy of KNN classifier is 84 % and maximum is 91 % which are much greater than those achieved using NBM classifier.

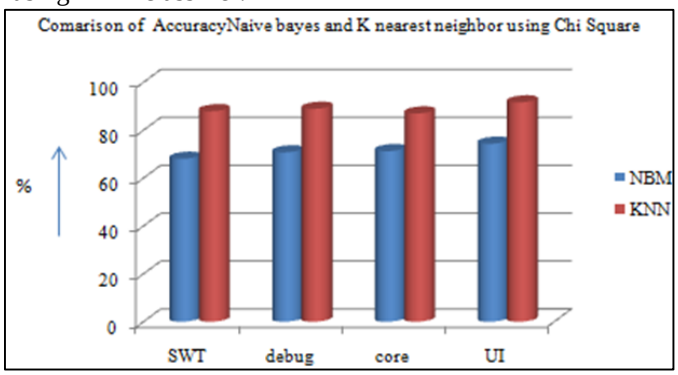

Fig. 3: Showing comparisons of Naive bayes multinomial and KNN in terms of accuracy of four components.

The comparison of accuracy of NBM and KNN using Chi square for dictionary formation of different component is also shown in Figure 6.2.It is again found that KNN performs better than NBM in terms of accuracy. The minimum accuracy of NBM is 68 % and maximum accuracy is 74 % whereas minimum accuracy of KNN classifier is 84 % and maximum is 91 % which are much greater than those achieved using NBM classifier.

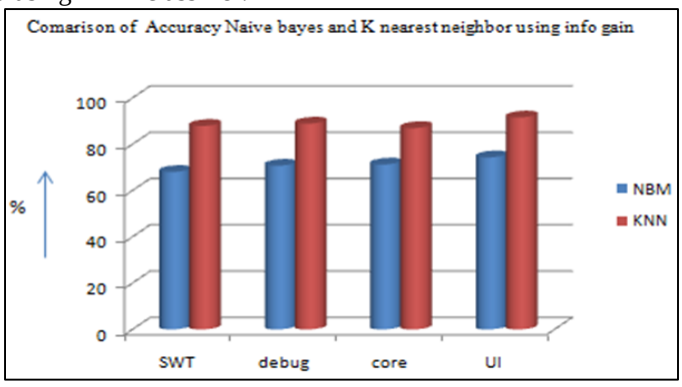

Fig. 4: Showing comparisons of Naive bayes multinomial and KNN in terms of accuracy of four components.

Hence, it can be concluded from the results that KNN performs better for bug classification severity under the given experimental set up. The probable reason behind better performance of KNN can be attributed to the fact that useful preprocessing steps were applied to raw dataset extracted from, Bugzilla repository. These preprocessing steps and the process of dictionary formation further refined the datasets. This enable KNN to efficiently classify to bug to their respective classes, as it is well known fact that KNN performs best when noise free data is fed to it. Therefore, it has been concluded that under used experimental conditions KNN performs better for bug severity classification. The underperformance of NBM with respect to the KNN algorithm can be understood in terms of their working principle. The KNN algorithm works on the principle of Euclidean distance while the NBM works on the principle of conditional initial probabilities. As the probabilities are calculated during training in NBM, the classifier classifies each data points on the basis of the same initial probabilities while the KNN classifies each data point on the basis of its current distance from its neighbors. Since, this problem requires scores calculated by text mining, each data point may have common values of attributes which is actually not utilized in the NBM.Hence, it can be concluded from the results that KNN performs better for bug classification severity under the given experimental set up. The probable reason behind better performance of KNN can be attributed to the fact that useful preprocessing steps were applied to raw dataset extracted from, Bugzilla repository.

VI. CONCLUSION

The software bugs that are detected after the deployment of software affect the reliability and quality of software. Bug tracking systems allow users to report these bugs of many open source software. However predicting the severity level of these bug report is emerging issue. Many attempts have been made to address the problem of severity prediction, but no attempt was made for creating dictionary of critical terms of severity indicator. The work presented in this thesis proposed a feature selection and classification approach for categorizing the bug reports into severe and non-severe class. Feature selection methods filter out most informative terms from datasets after preprocessing steps. Top 125 terms are selected and used as dictionary terms to train classifier.

Bug reports of four components of Eclipse are chosen for this research, four main sub processes performed in experiment are: dataset acquisition, preprocessing, Feature selection and classification. In this research work, after the preprocessing task, matrix of terms and documents are created. Then a feature selection technique is applied over them to get dictionary terms. Two feature selection methods named info – gain and Chi square. Then classification is done by two ML algorithms named as NBM and KNN and on basis of performance matrices their performance is compared. It is found that accuracy, precision is in the range of 64 % to 75 % and 66 % to 74 % using Naïve bayes classifier for bug reports of different components whereas using KNN classifier accuracy and precision is in the range of 87 % to 91 and 79 % to 95 % respectively.

A. Future Scope:

The proposed technique is used on few components of Eclipse for performing severity classification. Therefore in future other components of Eclipse may be used and cross component approach could be applied by creating global dictionary of all components of Eclipse. Also, domain specific projects could be taken into consideration for the study to create global dictionary of domain specific projects. The study could help to make domain specific tool for prediction of severity The study may be extended using data sources of other open source projects for the validation of proposed approach and findings. The approach used in this study takes NBM and KNN into account for classification. A comprehensive study could be conducted of other ML algorithms. Also, other feature selection methods could be used for creating dictionary of terms. The work in thesis only proposes to implement the proposed approach on offline database downloaded from Bugzilla repository and is not designing any such automated system for online classification. Therefore a study could be conducted to make the system online for real time classification of bug reports.

REFERENCES

[1] R. Patton, Software Testing (2nd Edition). Sams, 2005.
[2] Kiyoh Nakamura, Mamoru Sugawara and Masahiro Toyama “Defect Prevention Activities And Tools”IEEE 1991.
[3] Rogerson, S. (2002). The Chinook helicopter disaster. IMIS Journal,12(2).
[4] “DefectPriority”,http://softwaretestingfundamentals.com/defect-priority/, August 25, 2011.
[5] “Defect severity classification in software testing (with an example)”, http://www.zyxware.com/articles/3559/defect-severity-classification-in-software-testing-with-an-example, May 24, 2013.
[6] Iftekhar Ahmed, Nitin Mohan and Carlos Jensen “The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla”ACM 2014.
[7] A. E. Hassan, "The Road Ahead for Mining Software Repositories,"IEEE Computer society, pp. 48-57, 2008.
[8] E. S. Raymond. The cathedral and the bazaar. FirstMonday, 3(3), 1998.
[9] “15 Most Popular Bug Tracking Software to Ease Your Defect Management Process”, http://www.softwaretestinghelp.com/popular-bug-tracking-software/”, Feb 12, 2015.
[10] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic Bug Triage Using Semi-Supervised Text Classification," Proc. 22th Intl. Conf. Software Engineering & Knowledge Engineering (SEKE '10), Jul. 2010, pp. 209-214.
[11] P. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in Proceedings of the 29th international conference on Software Engineering, 2007.

[12] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in Proc Sixteenth International Conference on Software Engineering, Citeseer, 2004, pp.92–97.
[13] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in Proc 28th International Conference on Software Engineering. ACM, 2006, pp. 361–370.
[14] Anvik, J. 2007. Assisting Bug Report Triage through Recommendation. Ph.D. Dissertation, University of British Columbia
[15] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Tossing Graphs," Proc. 17th ACM SIGSOFT Symp. Foundations of Software Engineering (FSE '09), Aug. 2009, pp. 111-120.
[16] I. Herraiz, D. German, J. Gonzalez-Barahona, and G. Robles, "Towards a Simplification of the Bug ReportForm in Eclipse," in 5th International Working Conference on Mining Software Repositories, May 2008.
[17] T.Menzies and A. Marcus, "Automated severity assessment of software defect reports," in IEEE International Conference on Software Maintenance, 28 2008-Oct. 4 2008, pp. 346–355.
[18] G.Antoniol, K. Ayari, M. Di Penta, F. Khomh, andY.-G. Gu´eh´eneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in CASCON '08: Proceedings of the conference of the center for advanced studies on collaborative research.ACM, 2008, pp. 304–318
[19] Syed Nadeem Ahsan , Javed Ferzund , Franz Wotawa, "Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine", Proceedings of the 2009 Fourth International Conference on Software Engineering Advances, p.216-221, September 20-25, 2009
[20] Lamkanfi, Ahmed, et al. "Predicting the severity of a reported bug." Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.
[21] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What Makes a Good Bug Report?," IEEE Trans. Software Engineering, vol. 36, no.5, Oct. 2010, pp. 618-643.
[22] J. Anvik and G. Murphy, "Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions," ACM Trans. Software Eng. and Methodology, to appear.
[23] A. Tamrawi, T.T. Nguyen, J.M. Al-Kofahi, and T.N. Nguyen, "Fuzzy-Set and Cache-Based Approach for Bug Triaging," Proc. 19th ACM SIGSOFT Symp. Foundations of Software Engineering (FSE '11), Sept. 2011, pp. 365-375.
[24] Lamkanfi, Ahmed, et al. "Comparing mining algorithms for predicting the severity of a reported bug." Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. IEEE, 2011.
[25] Guo, Philip J., et al. "Not my bug! and other reasons for software bug report reassignments." Proceedings of the ACM 2011 conference on Computer supported cooperative work. ACM, 2011.