Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 132 (2018) 352–361

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2018)

# A Supervised Bug Report Classification with Incorporate and Textual field Knowledge

*"Ashima Kukkar[a], Rajni Mohana**
*Department of Computer Science*
Jaypee University of Information Technology, Waknaghat

**Abstract**

Performance of the bug prediction model is directly depends on the misclassification of bug reports. Misclassification issue surely scarifies the accuracy of the system. To resolve this issue the manual examination of bug reports are required, but it is very time consuming and tedious job for a developer and tester. In this paper the hybrid approach of merging text mining, natural language processing and machine learning techniques is used to identify bug report as bug or non-bug. The four incorporates fields with textual fields are added to bug reports to improve the performance of classifier. TF-IDF and Bigram feature extraction methods are used with feature selection and K-nearest neighbor (K-NN) classifier. The performance of the proposed system is evaluated by using Precision, Recall and F-measure by using five datasets. It is observed that the performance of K-NN classifier is changed according to the dataset and addition of bigram method improve the performance of classifier.



*Keywords:* Natural Language Processing, K-nearest neighbor, Incorporate, and textual fields based prediction, Software maintenance and development, Bug Triaging System

## 1. Introduction

In recent years, the software bug prediction is booming in software development and maintenance phase. It predicts various aspects like severity, priority, location, or number- based on the repository data [1] [2] [3] [4]. The basic aim of bug identification is reliability, reducing the maintenance cost, developer time and effort. To achieve this aim many statistical models have been developed in the literature relies on historical data for prediction [5]. The bug tracking system (BST) has the rich bug information and many applications in practice, among all information sources [6]. The bugs occurred normally in testing phase after development phase. Theses bugs are reported in the BTS as bug reports by user or developer in large number. The bugs may be having a different impact from user inconvenience. Some may have a major effect like system crash. Some may have minor like change in the

**Corresponding Author: ashi.chd92@gmail.com*

1877-0509 
10.1016/j.procs.2018.05.194

functionality of the software. Therefore, it is important to deal with critical bugs on priority basis. The necessity of resolving the critical bug requires a parameter. This parameter needs to be added during the bug reporting process which is specified by bug severity [7]. Bug severity is defined as the degree of bug impact on the software. This includes the impact on development, various operations and components of the system. But many researchers [8][9][10][11] noticed that many submitted reports were marked as bug but in actual it is not. This misclassification leads to bias and has great effect on the prediction performance. Through this the bug prediction results become more doubtful and misleading. To reduce this effect as well as to provide validity to prediction efforts a close manual examination is required. The manual classification can reduce the noise, but in the case of large bug reports. It goes to an impractical option due to the additional cost, associated time and efforts. Therefore, there is a need of automated bug severity classification mechanism to check whether the given bug report is actually a bug or not. It should be much acceptable rather than argument productivity. In [11] the researcher investigated the automatic bug report classification by using text mining techniques to determine the feasibility. However the work only focused on the description part (free text) of the bug reports. Generally the bug reports contain semi-structured documents with fixed set of attributes and informal narrations. This semi-structured text written in machine generated stack traces natural language as long comments (including discussions) and summary of bug. On the other side the structural documents consist of well defined and finite values like keywords, priority, component, severity etc. therefore the bug reports resides with mixed characteristics as shown in Figure 1.

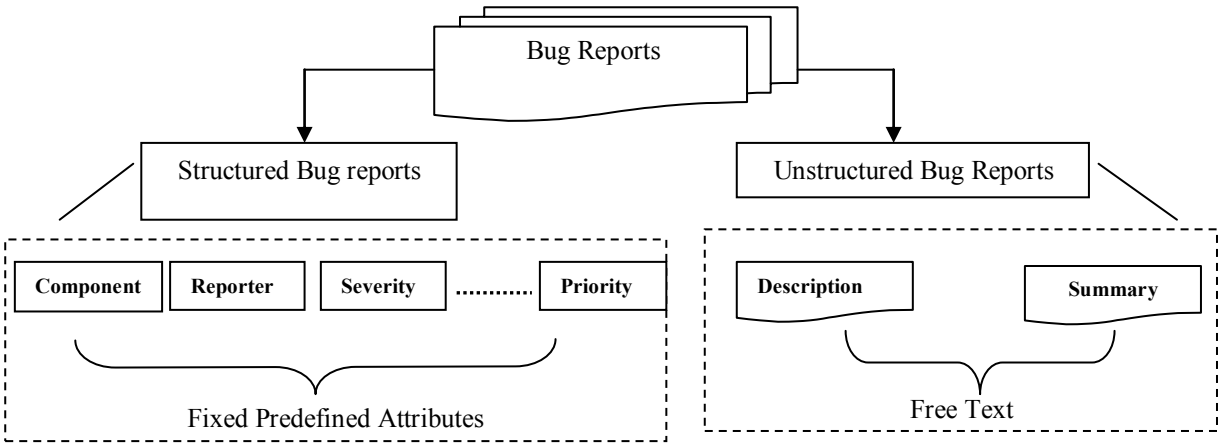



Figure 1: Fields of Bug Report

Only a few researchers [3][12][13][14][15][16][17] added the incorporate multiple fields in the classification results. These works have also focused on the association between unstructured text and selected fields. On the other hand, few researchers [18][19] argued that the structured text fields are important factors for bug prediction.
In this paper, we want to answer the question of whether or not a given bug report is actual bug one through a hybrid approach by merging text mining (TM), natural language processing (NLP) and machine learning techniques (ML). To improve the bug severity prediction, we introduced a novel method which added description and fixed selected fields (Reported fields reported by user).
Below is the contribution of this paper

1. We introduced the novel approach to automate the binary bug severity prediction process by using benchmark datasetes [16] of five open bug repertories that are Mozilla, Eclipse, JBoss, Firefox and OpenFOAM.
2. We added the four incorporate fields (reporter, severity, priority and component) with textual information like comments, description, summary etc. whether bug reports are real bug or not.
3. We also observed the effect of using bigram and TF-IDF approach with feature selection and K-NN classifier on different datasets. In order to calculate the accuracy of the proposed model, Precision, Recall and F-value are used.

*Organization*: This paper is divided into four sections. Section II presents the Related Work of bug report classification. Section II introduces the Proposed Approach. Section IV explains the experiments and results of the proposed approach and the conclusion and future work is discussed in Section V.

## 2. Related Work

Bug triaging helps in predicting whether the bug report is an actual bug or non-bug. The severity classification process is tedious and time consuming because of the accommodation of a substantial number of bug reports in BTS. From the long time, the researchers have been trying to automate the bug triage process. Decent achievement has likewise been accomplished. A list of such works is altogether examined in this section as follows:
In previous research, the researchers have classified the binary bug severity into Sever and Non-Sever classes /Bug and Non-Bug classes.

### *2.1. Bug class and Non-Bug class*

The work of Antoniol et al. [11], in 2008 developed an automatic bug severity approach to detect the bug report whether it is a real bug or request. The features are extracted by Active Directory tree (ADT) and vector space technique. The training is given by Naïve Bayes (NB) and Linear Logistic regression (LLR) algorithms. The precision of Mozilla, Eclipse and JBoss projects interfered with 77% to 82%. In the same field, Pingclasai et al. [20] in 2013 proposed a classification approach to identify bugs and non-bugs. They used Latent Dirichlet allocation (LDA) method with NB and LLR classifier. The precision of HTTPClient, Jackrabbit and Lucene projects varied from 66% to 76%, 65% to 77% and 71% to 82% respectively. The other work by Zhou et al. [16] in 2014, introduced the automatic bug prediction technique by combing text and data mining algorithms. The summary of bug reports are extracted by text mining algorithm and fed into the machine learner model with selected structured fields. The data drafting algorithm is used to combine the stages. The execution showed better results by increasing the f-measure. The range of f-measure of OpenFOAM, JBoss, Mozilla, Eclipse and Firefox projects are from 8% to 85.9%, 84.6% to 93.7%, 78.3% to 81.7%, 76.7% to 80.2% and 72.8% to 79.5% respectively. Another author, Chawla et al. [21] proposed a severity classification model by using latent semantic indexing (LSI) and tf-idf feature extracted techniques. The fuzzy logic algorithm is used to train the classifier for better results.

### 2.2. Sever class and Non-Sever class

The work by **Lamkanfi et al.** [22], in 2010 introduced the NB classifier with tf-idf approach to mark the bugs as non-sever and sever. The overall accuracy of Mozilla, GNOME and Eclipse came in the range of 65 to 85%. The similar work has been through by **Siahaan et al.** [23] in 2011. They have reduced the data by in-foGain method and trained the model with support vector (SVM). They have attained 99.83% accuracy. Another bug severity classifier is proposed by **Nagwaniet al.** [25], in 2013 to mark the bug reports as bug and non-bug. They have used the Simple Random Sampling (SRS) algorithm to generate the taxonomic terms. The accuracy of Mozilla, JBoss-Seam, Andriod and Mysql is evaluated. The chi-square feature selection method is used with bigram by **Kantiet al.** [24], in 2014. They noticed that the performance of classifier is enhanced by using n-gram application. Further **Gujral et al**. [26], in 2015 utilized the TF-IDF feature extraction technique and Naïve Bayes Multinomial (NBM) classifier. The accuracy as 69% and precision as 70% were calculated of Eclipse bug reports. In 2015, **Sharma et al.** [7] used two feature selection (In-foGain and Chi-square) techniques. The dictionaries of critical terms are created by selected terms and NBM or k-nearest neighbor (K-NN) classifier is used to train the model. They observed that the accuracy of K-NN classier is better with range of 69% to 75%. In 2017, **Pandey et al.** [27] noticed that the performance of the proposed classifier is switched by using different dataset. They implemented the classifier by using K-NN, NB, Random Forest (RF), SVM and LDA machine learning algorithms to classify the bugs as sever class and non-sever class and gained the accuracy from 75% to 83%. In 2010, **Gegick*et al.*** [28] proposed the bug severity classification approach by using SAS text miner with SVD on Cisco dataset and found that the 77% bug's reports are misclassified. They also compared many feature selection techniques like In-foGain, Chi-square and Correlation Coefficient with NBM classifier. They noticed feature selection methods helped in increasing the precision of the dataset. **Jin et al.** [17], in 2016 proposed an approach for normal bug severity reports which showed the better

results on Lamkanfi dataset [22]. The NB classifier is designed with addition of text and Meta –fields.
From the above literature study we conclude that there is a need of bug fixing due to its harmful impact on software. But the classification task of bug is a manual. We require a master to look at announced bug reports and identify whether the reported bug is bug or non-bug. Numerous endeavors are taken by various researchers to automate this procedure by utilizing various machine learning and text mining techniques as examined in our literature study. Yet at the same time it is a long way from giving reliable exactness of automatic severity prediction model.

## 2 Proposed Summarization System

The entire procedure of severity classification which is proposed in this paper can be condensed as follows: The detailed methodology of the proposed model is described in Fig.2 and steps of whole process are described below:

### *2.1. Data Acquisition*

The experiment considers the benchmark dataset [16] of five open bug repertories that are Mozilla, Eclipse, JBoss, Firefox and OpenFOAM. The researcher collected the 3200 bug reports from these projects and labeled them manually as bug and non-bug shown in Table 1.

Table 1: The Information of Bug Reports

| Projects | No. of Bug Reports | Bug | Non-Bug |
|---|---|---|---|
| Mozilla(core) | 539 | 343 | 196 |
| Eclipse | 693 | 473 | 220 |
| JBoss | 573 | 365 | 208 |
| Firefox | 620 | 412 | 208 |
| OpenFOAM | 795 | 535 | 260 |

### *2.2. Preprocessing*

The XML file of bug reports are taken as input with the summary and addition attribute like reporter, severity, priority and component. On the textual summary the standard preprocessing steps are performed like tokenization, removal of strop words and stemming.

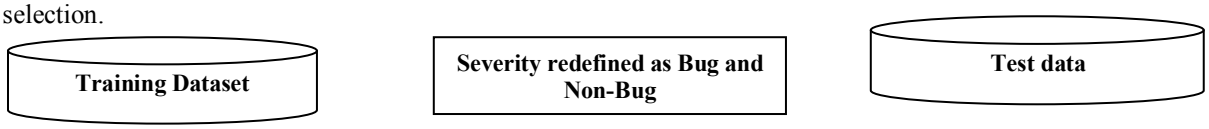
a) Tokenization: It is the task of tokenization is to remove periods, commas, punctuation, other characters like hyphen and brackets.
b) Stop Word Removal: It is the task of removing unwanted words to reduce the textual data. So that, the performance of the system is improved.
c) Stemming: It is the task of cutting the word from their root node to reduce the textual data.

### *2.3. Extraction of features to determine actual bug.*

After the preprocessing step the features are extracted as summary, reporter, severity, priority and component. For extraction task term frequency and inverse document frequency (TF-IDF) and bigram method are used to calculate the scores. These extracted features help in categorizing the bug reports as bug or non-bug. The use TF-IDF feature extraction method helps in finding the occurrence feature in the document. The use of bigram feature extraction method adds the semantics to bug reports and helps in reducing the sparsity of dataset [29]. These both scores matrix are added to generate the single matrix.

### *2.4. Feature Selection*

After feature extraction, the feature selection methods are applied to recover the most important features from feature matrix corpus. In this work, the info gain algorithm is utilized. This algorithm is used to reduce the features by selection.



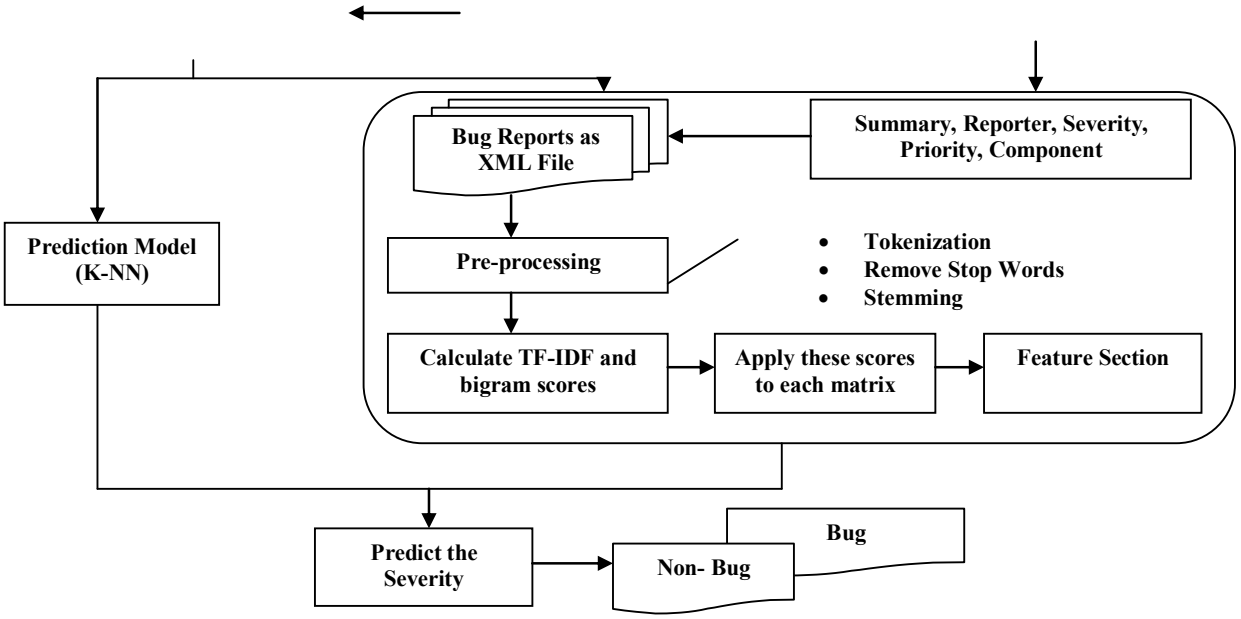



Figure 2: The Process of Proposed Model

### *2.5. Classification using Bug and Non-bug*

The best characterized selected features are fed into naïve bayes as machine learner [30]. The model is trained by using training data and the performance is evaluated using test data. The training and test data of 70-30% of available reports are chosen for the experimental purpose.

### *2.6. Performance measurement*

The performance is evaluated by using Precision, Recall, and F-measure. Recall is the ratio of true positive divided by sum of true positive and false negative. Precision is the ratio of true positive divided by sum of true positive and false positive. F-measure is the harmonic mean of recall and precision [31].

| **Algorithm 1: The Algorithm of Proposed Model** |
|---|
| Input: XML file of bug reports<br>Output: Classification of bug report as bug or non-bug. |
| Step 1: The input text as XML file of bug reports with summary, reporter, severity, priority and component for pre-processing<br>Tokenization→ Stop word removal→ Stemming<br>Step 2: Extract TF-IDF and Bigarm features. Collect features that have occurred frequently more than once in the document. We called this collection of features as vocabulary (V).<br>$V_1^n = v^1, v^2, v^3 \ldots\ldots v^n$<br>Step 3: Select the features by using Info gain method.<br>Info-gain method<br>$H_{(P_j \ldots\ldots P_k)} = \sum_{j=1}^{k} P_j \log_2 Pj$……(5)<br>$P_j$ = Probability of class value<br>n= No. of possible classes<br>$IG = H_P - \sum_{j=1}^{k} P_{cj} H_{cj}$……(6) |

$H_P$ = Entropy of dataset
$n$ = No. of values of target class
$P_{cj=}$ Probability of feature j
$H_{cj}$ = Entropy of feature i
Step 4: Calculate the distance D from the training features($y_1, y_2 \ldots y_n$) to testing features($y'_1, y'_2 .. y'_n$) using Euclidean distances function.

$$D(y, y') = \sqrt{(y_1 - y'_1)^2 + (y_2 - y'_2)^2 + \cdots + (y_n - y'_n)^2} \quad \ldots\ldots\ldots(7)$$

Step 5: Sort the training features in ascending order according to the distance.
Step 6: Select the top K training features as nearest neighbor.
Step 7: The k training features of common class C is assigned to test features.
Step 8: Bugs are categorized as bug or non-bug and performance is evaluated.

## 3. Experimental Result

Bug reports includes a status field which demonstrates the present status of bug, a resolution field which presents data identified with the bug, an assignee field that shows who is incharge of settling the bug and a Reporter field which displays name who submit the bug. These fields contain single attribute text value while some fields contain multiple attribute values for example, banners CC rundown and catchphrases and banners. Bug report fields, for example, Priority, Component, Reporter, Severity and Summary were utilized as a part of the experiment. These five fields of the bug report were chosen for test with a specific end goal to enhance precision for bug severity. The criteria may be varying according to the value of Component and reporter fields. But the Severity, Priority and Summary fields are fundamental part for deciding whether the bug is bug or not. The proposed approach works in this way: at first the proposed approach combined the bug reports by Component and Reporter fields and after that the approaches extract the information from summary fields. At last the naïve bayes is applied on extracted features. The proposed approaches have exploited the combined effect of bigram and TF-IDF in comparison to bigram. The K-nearest neighbour (K-NN) approach is considered to support the features representation that has been utilized with accomplishment in past research. In that capacity, the two fundamental research questions we reply in the paper are:

RQ1. Does bi-gram approach deliver an improvement over TF-IDF approach with feature selection in bug and non-bug classification ?

RQ2. What is the influence of bigram when added to TF-IDF approach with feature selection for bug and non-bug classification?

The series of experiments are performed based on the above approach to observe the effectiveness of proposed system in determining the bug or non- bug by using Precision, Recall and F-measure.

### *3.1. Experiment 1*

In this experiment, the proposed approach with both feature extraction method TF-IDF and bigram is applied on five open source project as shown in Table 1. The results of bug and non- bug classification are shown in Table 2. The f-measure of bug and non- bug classes varies from 68% to 94% and 86% to 97% respectively.

Table 2 : Prediction analysis of Bug Class and Non- Bug using bigram and TF-IDF with Info gain and K-NN

| Projects | PREDICTION ANALYSIS of BUGS | | | RESULT ANALYSIS of NON-BUGS | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-measure | Precision | Recall | F-measure |
| Mozilla(core) | 0.92 | 0.54 | 0.68 | 0.92 | 0.84 | 0.87 |
| Eclipse | 0.94 | 0.89 | 0.91 | 0.94 | 0.88 | 0.90 |
| JBoss | 0.95 | 0.94 | 0.94 | 0.98 | 0.96 | 0.97 |
| Firefox | 0.93 | 0.96 | 0.94 | 0.94 | 0.86 | 0.89 |
| OpenFOAM | 0.93 | 0.83 | 0.87 | 0.94 | 0.80 | 0.86 |

### *3.2. Experiment 2*

In this experiment, the proposed approach with feature extraction method bigram is applied on five open source project as shown in Table 1. The results of bug and non- bug classification are shown in Table 3. The f-measure of bug and non- bug classes varies from 62% to 81% and 59% to 86% respectively.

Table 3: Prediction analysis of Bug Class and Non- Bug using bigram with Info gain and K-NN

| Projects | PREDICTION ANALYSIS of BUGS | | | RESULT ANALYSIS of NON-BUGS | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-measure | Precision | Recall | F-measure |
| Mozilla(core) | 0.49 | 0.87 | 0.62 | 0.80 | 0.93 | 0.86 |
| Eclipse | 0.88 | 0.84 | 0.85 | 0.66 | 0.71 | 0.68 |
| JBoss | 0.52 | 0.87 | 0.65 | 0.43 | 0.94 | 0.59 |
| Firefox | 0.78 | 0.88 | 0.82 | 0.78 | 0.85 | 0.81 |
| OpenFOAM | 0.74 | 0.90 | 0.81 | 0.84 | 0.85 | 0.84 |

### *3.3. Experiment 3*

In this experiment, the proposed approach with feature extraction method TF-IDF is applied on five open source project as shown in Table 1. The results of bug and non- bug classification are shown in Table 4. The f-measure of bug and non- bug classes varies from 58% to 76% and 55% to 81% respectively.

Table 4: Prediction analysis of Bug Class and Non- Bug using TF-IDF with Info gain and K-NN

| Projects | PREDICTION ANALYSIS of BUGS | | | RESULT ANALYSIS of NON-BUGS | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-measure | Precision | Recall | F-measure |
| Mozilla(core) | 0.45 | 0.83 | 0.58 | 0.76 | 0.87 | 0.81 |
| Eclipse | 0.79 | 0.75 | 0.76 | 0.60 | 0.68 | 0.63 |
| JBoss | 0.47 | 0.82 | 0.59 | 0.40 | 0.89 | 0.55 |
| Firefox | 0.73 | 0.78 | 0.75 | 0.78 | 0.80 | 0.78 |
| OpenFOAM | 0.69 | 0.81 | 0.74 | 0.79 | 0.80 | 0.79 |

### *3.4. Experiment 4*

In this experiment, proposed approach with both feature extraction method TF-IDF and bigram is compared with existing approach with all relevant features [16]. Both methods are applied on five open source project as shown in Table 1. The comparison results are shown in Table 5 and Figure 3. It can be seen that overall proposed approach performed better than the existing approach.

Table 5: Comparison with Proposed Work using Best Existing Prediction Model based on Bug class and Non-Bug Class.

| Types | Classes | Mozilla(core) F-measure | Eclipse F-measure | JBoss F-measure | Firefox F-measure | OpenFOAM F-measure |
|---|---|---|---|---|---|---|
| Proposed Approach (bigram + TF-IDF + Info gain + K-NN) | Bugs | 0.68 | 0.91 | 0.94 | 0.94 | 0.87 |
| | Non-Bugs | 0.87 | 0.90 | 0.97 | 0.89 | 0.86 |
| Existing Approach | Bugs | 0.87 | 0.87 | 0.95 | 0.86 | 0.90 |
| | Non-Bugs | 0.72 | 0.66 | 0.91 | 0.66 | 0.76 |

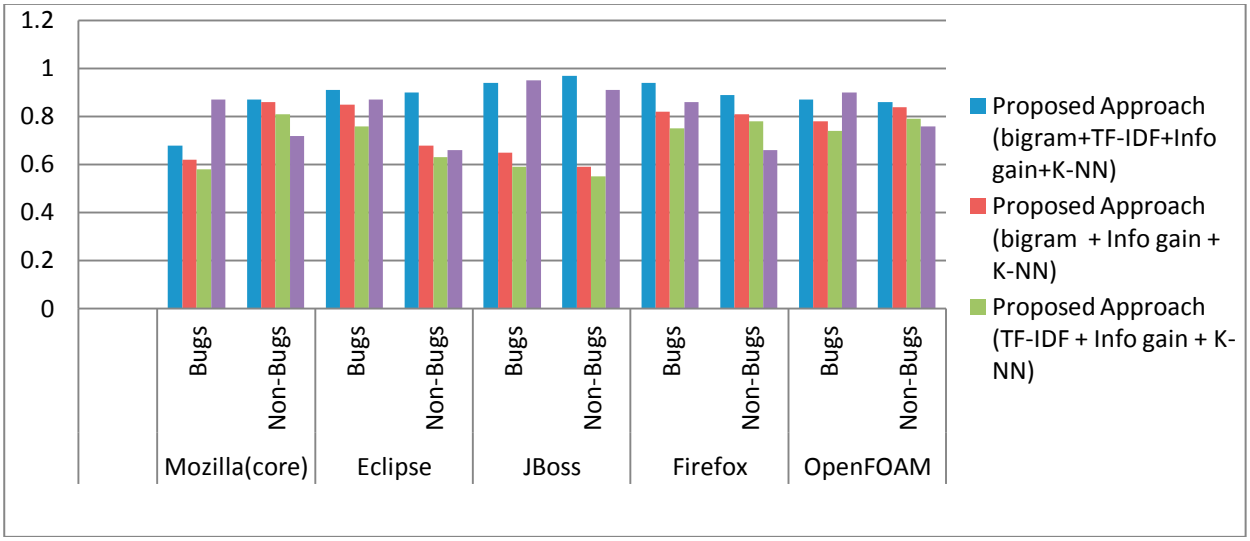



Figure 3: Comparison with Proposed Work using Best Existing Prediction Model based on F-measure.

The Figure 3 shows the comparison of precision, recall and F-value using K-NN classifier. It was observed that the recall, precision and F-measure is changed according to bug reports dataset. That means the performance of K-NN classifier is changed by using different datasets. The same approach showed different results bug using different datasets. The main findings of this paper are summarized below. These points also give the answer of the RQ1 and RQ2.

- Application of bigram in addition to TF-IDF for feature selection delivers the better result for bug and non-bug classification in some cases;
- Feature selection conveys more extensive advantages to the execution of the classifiers, with the additional advantage of terms that can be utilized for stakeholders. We proposed to introduce TF-IDF and bi-grams with the guide of feature selection, with the goal that exclusive the most informativeness and phraseness terms are added. On account of feature selection, obviously, the improvements are for every dataset considered freely of the features considered;

Because the application of bigram features extraction method adds the semantics to bug reports and helps in reducing the sparsity of dataset. Whereas the application of TF-IDF features extraction method only helps in finding the occurrence feature in the document.

## 4. Conclusion

In this paper the hybrid approach of merging text mining, natural language processing and machine learning techniques are used to identify which bug report as bug or non-bug. Due to this the noises of misclassification is reduced by filtering the bug reports and enhance the performance of automatic bug prediction. In this work the four incorporate fields (reporter, severity, priority and component) with textual information like comments, description, summary etc are added to testing and training dataset. The bigram and TF-IDF approach is used with Info gain for the bug severity prediction. The bigram approach helped in reducing the sparisity of dataset. In order to calculate the accuracy of proposed model, Precision, Recall and F-measure are used. It is observed that, the performance of K-NN classifier is changed according to the dataset. In future the optimization algorithms are applied with bigram and the text mining techniques to enhance the performance of automatic bug severity prediction.

**Acknowledgment**

We are very thankful to Yu Zhou from Nanjing University of Aeronautics and Astronautics, Yanxiang Tong from State Key Lab. of Novel Software Technology , Ruihang Gu from Nanjing University , Harald Gall from University of Zurich, Switzerland for providing the dataset [16].

References

[1] E. Giger, M. Pinzger, and H. Gall (2010), "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, pp. 52–56.
[2] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller (2007), "Predicting faults from cached history," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, pp. 489–498.
[3] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals (2010), "Predicting the severity of a reported bug," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, pp. 1–10.
[4] T. J. Ostrand, E. J. Weyuker, and R. M. Bell (2005), "Predicting the location and number of faults in large software systems," *Software Engineering, IEEE Transactions on*, vol. 31, no. 4, pp. 340–355.
[5] M. D'Ambros, M. Lanza, and R. Robbes (2010), "An extensive comparison of bug prediction approaches," in *Mining Software Repositories (MSR), Working Conference on*. IEEE, pp. 31–41.
[6] M. DAmbros, H. Gall, M. Lanza, and M. Pinzger (2008), *Analysing software repositories to understand software evolution*. Springer.
[7] Gujral, S., Sharma, G. and Sharma, S., (2015), February. Classifying bug severity using dictionary based approach. In *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), International Conference on* (pp. 599-602). IEEE.
[8] K. Herzig, S. Just, and A. Zeller (2013) , "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, pp. 392–401.S.
[9] Kim, H. Zhang, R. Wu, and L. Gong (2011), "Dealing with noise in defect prediction," in *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE , pp. 481–490.
[10] P. S. Kochhar, T.-D. B. Le, and D. Lo (2014), "It's not a bug, it's a feature: does misclassification affect bug localization?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, pp. 296–299.
[11] Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc YG (2008) Is it a bug or an enhancement? A text-based approach to classify change requests. In: Proceedings of the conference of the center for advanced studies on collaborative research: meeting of minds (CASCON'08), ACM, pp 23:304–23:318.
[12] T. Menzies and A. Marcus (2008), "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, pp. 346–355.
[13] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto (2010), "Predicting re-opened bugs: A case study on the eclipse project," in *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, pp. 249–258.
[14] Y. Tian, D. Lo, and C. Sun (2012), "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, pp. 215–224.
[15] J. Xuan, H. Jiang, Z. Ren, and W. Zou (2012), "Developer prioritization in bug repositories," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, pp. 25–35.
[16] Zhou Y, Tong Y, Gu R, Gall H (2016) Combining text mining and data mining for bug report classification. J Softw Evol Process 28(3):150–176.
[17] Jin, K., Dashbalbar, A., Yang, G., Lee, J.W. and Lee, B., (2016). Bug Severity Prediction by Classifying Normal Bugs with Text and Meta-Field Information. *Advanced Science and Technology Letters*, *129*, pp.19-24.
[18] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim (2008), "Extracting structural information from bug reports," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, pp. 27–30.
[19] Y. Tian, D. Lo, and C. Sun (2013), "Drone: Predicting priority of reported bugs by multi-factor analysis," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, pp. 200–209.
[20] Pingclasai N, Hata H,Matsumoto Ki (2013) Classifying bug reports to bugs and other requests using topic modeling. In: Proceedings of the 2013 20th Asia-Pacific software engineering conference (APSEC'13). IEEE, vol 2, pp 13–18.
[21] Chawla I, Singh SK (2015) An automated approach for bug categorization using fuzzy logic. In: Proceedings of the 8th India software engineering conference (ISEC'15). ACM, pp 90–99

[22] Lamkanfi A, Demeyer S, Giger E, Goethals B (2010) Predicting the severity of a reported bug. In: Proceedings of the 2010 7th IEEE working conference on mining software repositories (MSR'10). IEEE, pp 1–10.

[23] Sari, G.I.P. and Siahaan, D.O., (2011). An attribute selection for severity level determination according to the support vector machine classification result. In *Proceedings of the 1st international conference on information systems for business competitiveness (ICISBC)*.

[24] Roy, N.K.S. and Rossi, B., (2014), August. Towards an improvement of bug severity classification. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on* (pp. 269-276). IEEE.

[25] Nagwani, N.K., Verma, S. and Mehta, K.K., (2013), November. Generating taxonomic terms for software bug classification by utilizing topic models based on Latent Dirichlet Allocation. In *ICT and Knowledge Engineering (ICT&KE), 2013 11th International Conference on* (pp. 1-5). IEEE.

[26] Sharma, G., Sharma, S. and Gujral, S., (2015). A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms. *Procedia Computer Science*, *70*, pp.632-639.

[27] Pandey, N., Sanyal, D.K., Hudait, A. and Sen, A., (2017). Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, pp.1-19.

[28] Gegick, M., Rotella, P. and Xie, T., (2010), May. Identifying security bug reports via text mining: An industrial case study. In *Mining software repositories (MSR), 2010 7th IEEE working conference on* (pp. 11-20). IEEE.

[29] Verberne, S., Sappelli, M., Hiemstra, D. and Kraaij, W., (2016). Evaluation and analysis of term scoring methods for term extraction. *Information Retrieval Journal*, *19*(5), pp.510-545.

[30] Lee, C.H., Gutierrez, F. and Dou, D., (2011), December. Calculating feature weights in naïve bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on* (pp. 1146-1151). IEEE.

[31] Mohana, R. (2016). Anaphora resolution in Hindi: Issues and directions. Indian Journal of Science and Technology, 9(32).