Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 167 (2020) 1345–1353

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2019)

Does bug report summarization help in enhancing the accuracy of bug severity classification?

"Ashima Kukkar[a], Rajni Mohana[b], Yugal Kumar [C]
Department of Computer Science
Jaypee University of Information Technology, Waknaghat
{[a]ashi.chd92, [b]rajnivimalpaul, [c]yugalkumar.14 }@gmail.com"

Abstract

The programmer cannot write a program without any bug. A large numbers of bugs are deposited into the bug tracking system through bug reports. To find the root cause of a bug, a meaningful and huge conversation happens between the developer and reporter. The developer (triager) reads the whole bug report and then classified according to severity. The previous researchers observed that the bug report summaries provide the more resourcefully investigate information in the bug repository to the developer as part of the severity classification task. To further investigate the relationship between bug report summary and bug severity classification. A novel approach is proposed by using swarm intelligence and machine learning approaches. Firstly the n-gram technique is used to extract the semantic features score. These features are fed into the Summary Subset Selection Phase to select the optimal summary subset. The selected subset features are fed into the feature scoring phase to provide a relative score to each feature. These optimized features are used to train the proposed model. At last Naive Bayes approach is used to classify the multiclass severity classification. The results are analyzed by using 10-fold cross-validation on three benchmark datasets showed better performance in terms of Precision, Recall and F-measure. It is observed that the performance depend on the bug report contents. If the bug report has larger data for summarization than the summarization increase the classification accuracy otherwise decrease.

Keywords: Natural Language Processing; Bug Report Summarization; Unsupervised; Supervised ; Particle Swarm Optimization; Ant colony Optimization; Bug Severity Classification.

10.1016/j.procs.2020.03.345

1. Introduction

The bug resolution is the important challenge of the software maintenance process. A vast number of bugs are deposited into the bug tracking system in the form of bug reports. Numerous software companies fork out a substantial amount in finding and fixing the bugs [1]. The National Institute of Standards and Technology (NIST) spend $60 billion on bug detection, and the investigators found that the finding and fixing the bugs save more than $22 billion per year [2]. So, the gap between saving and expenditure related to bugs identification and reporting that enhances the reliability and quality of software. The whole bug resolution process is shown in Figure 1. In standard practice, the end user experiences a bug, while dealing with the system and reports the bug into the bug tracking system (BTS) by entering the severity, description, product, platform and component fields. To find the root cause of a bug, a meaningful and huge conversation happens between the developer and reporter by bug reports. So the bug reports have reported conversation in the form of messages from multiple people, and these messages might contain few lines or multiple passages. To resolve the bug, the developer (triager) has to analyze all the comments in bug reports [3]. After that, the developer summarized the bug report, and other developers use this summary for duplicate bug report detection and bug triaging process. The bug triaging process includes bug severity classification and bud assignment process. In the bug triaging process firstly the bugs are classified according to bug severity and according to its severity these bugs are assigned to the appropriate developer. In literature, it is proven that the manual method of bug resolution is complicated to use in practice because it demands excessive human effort. Therefore, there is need for automatic bug resolution process [4, 5].

Nevertheless, previous researchers [6,7] described that the automatic generated bug report summaries provides the more resourcefully investigate information in the bug repository to the developer as part of the severity classification task. Perhaps optimally, after the bug is resolved, the summary of a bug report is written by its author save the other developer time who later accesses the bug report.

In this paper, the work is focused on the relationship between the bug report summary and bug severity classification. Further, whether the bug report summary helps in finding the appropriate severity of bug or not is investigated. The bug reports are varying in length. Some are lengthy that include user and many developer conversations. Others are short and consist of only a few words. The severity classification depends on the text of the bug reports. The manual process of summarization and classification is very time consuming, tedious and needs a lot of human efforts. So there is a need for automatic bug report summarization and multiclass severity classification system. The main objective of this work is to implement the unsupervised bug report summarization system and supervised severity classification system. A novel approach is proposed by using swarm intelligence and machine learning approaches. Firstly the text of the bug report is pre-processed. Further, the n-gram technique is used to extract the semantic features score. These features are fed into the Summary Subset Selection Phase to select the optimal summary subset by using Particle Swarm optimization (PSO). The selected subset features are fed into the feature scoring phase to provide a relative score to each feature by using Ant colony optimization (ACO). These optimized features are used to train the proposed model. At last, the Naive Bayes machine learning algorithm is used to classify the severity of the bug report. The results are analyzed by using 10-fold cross-validation on three benchmark datasets and showed better performance in terms of Precision, Recall and F-measure.

The following paper is categorized into six sections. Section 2 presents the related work; Section 3 the detailed part of the proposed methodology; the experiments and results show in section 4; Section V discusses the conclusion and future work.

2. Related Work

Mostly researchers have worked on either bug report summarization or bug severity classification process. Very less work is done on the combination of both processes. Firstly, Murphy et.al [6] proposed a supervised summarization model by using three classifiers in 2010. The classifier is trained with Email dataset, Email and meeting dataset and bug report dataset. The generated summaries are validated by using human-generated summaries of 36 bug reports from various open sources projects. The classifier which is trained with bug reports, showed more than 62% precision rather than two classifiers. In the extended version [7], Murphy et al. checked

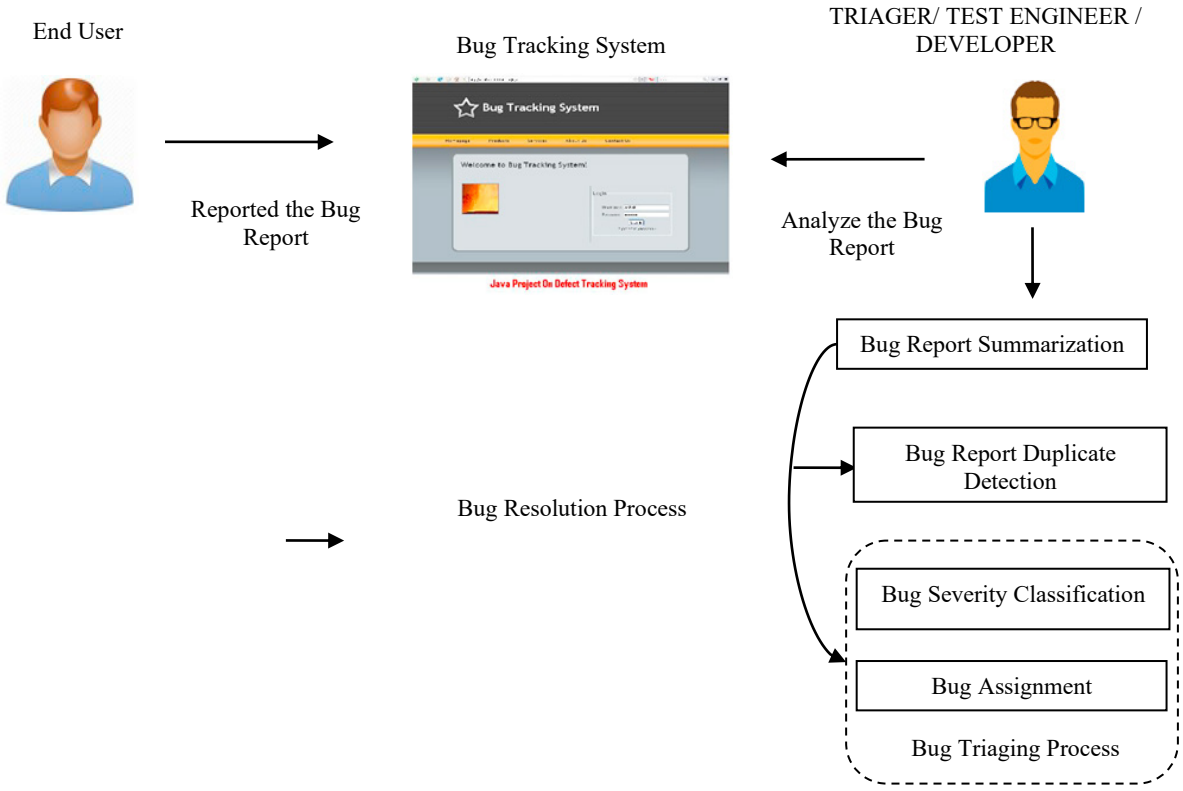

Figure 1: The Bug Resolution Process

whether the system generated summaries assist in identifying the duplicate bug reports. The task-based validation is performed, and the results showed that the generated summaries save the developer time in the duplicates bug without decreasing the accuracy. In 2018, Kukkar et al. [8] proposed a summarization model by using ant colony optimization algorithm (ACO). The feature extraction techniques (TF-IDF, N-gram) are used to extract the features after the pre-processing. Based on the user summary percentage, all subsets are generated. Later the optimal subset is selected by using ACO. The proposed model showed better results as compared to state art approaches [6,7]. In 2016 Zhou et al. [9] proposed a hybrid approach to automate the prediction process by combining both data and text mining techniques. Initially, the bug report's summary was extracted by using text mining techniques and divided them into three levels. Then in the second stage, the other structured and features were extracted and provided to the machine learner. The various data grafting techniques were utilized to combine two stages. The performance showed the best result by increasing the f-measure from 79.8% to 85.9% for OpenFOAM, 84.6% to 93.7% for JBoss, 78.3% to 81.7% for Mozilla, 76.7% to 80.2% for Eclipse 72.8% to 79.5% for Firefox. In 2017, Pandey et al. [10] applied the different machine learning algorithms (NB, LDA, SVM, Random Forest (RF) and K-NN) to classify the bugs into sever and non- sever category. The author observed that the performance of the classifier depends on the dataset. The proposed approach achieved accuracy from 75% to 83%. In 2018, Katerina et al. [11] implemented a supervised and unsupervised approaches to classify security and non-security bugs of NASA dataset. The author used TF-IDF, Term Frequency (TF) and Bag of word frequency feature vector methods for both approaches, further multiple classifiers like NBM, SVM, K-NN, NB and BN for supervised approach and anomaly detection method for the unsupervised approach. The results showed that the supervised approach performed better than the unsupervised approach. Zhang et al. [12] modified the REP and K-NN methods to find the bug report, which was similar to the historical bug report in

2016. Next, the features are extracted to classify the bug reports into Blocker; Trivial, Critical, Minor and Major classes of Eclipse, GCC and Mozilla datasets. The author observed that the similarity measure improved the severity prediction and fixer recommendation process. Further, Sharmin et al. [13] drafted a bug feature selection (BFS) technique by using Pareto optimality to classify the bugs into Blocker, Trivial, Critical, Minor and Major classes by searing informative features in 2017. The performance was carried on three open source projects, i.e. Eclipse, GCC and Mozilla and results showed that the BFS technique performed better than existing technique [9]. In 2019, Kukkar et al. [14] proposed a deep learning severity classification model. The Convolutional Neural Network was used to extract the features and Random forest with Boosting was used as classifier. The proposed model showed better results as compared to state art approach [9].

From the related work, it can be seen that only Murphy et al. observed the effect of system generated summaries on another process of bug resolution. The rest researches do not observe whether the generated summaries are helpful in bug resolution process or not. Therefore in this work, we checked whether the generated summaries are helpful in bug severity classification task.

3. Proposed Summarization and Severity Classification System

The overall proposed approach is elaborated in this section. This model automatically generates the summary of bug reports and after that the bug reports classifies according to its severity. The proposed model is described in the Figure 2.

3.1. Proposed Model

The problem of automatic bug report summarization is described as unsupervised algorithm whereas severity classification is illustrated as the supervised classification algorithm. The essential steps are explained as follows:

3.2. Pre-processing

The bug reports have various attributes like title, description, product, component, code snippets, stacktraces, priority, severity. But in this work severity, description and title attributes are taken as an input. After that text goes into the pre-processing phase. The pre-processing step aims to remove the unnecessary words from the content of the bug report [15]. It includes three necessary steps first is tokenization, second is stop-word removal and then stemming. Initially, the stream of text is broken into words, numbers, punctuations, and so forth called tokens. Further, all the punctuations are replaced with the blank spaces, non-printable escape characters are removed, and the words are converted into lowercase. After that, the stop words like verbs, nouns, articles, pronouns, adverbs, prepositions, and so forth, which are provided in the Natural Language Toolkit, are expelled. Finally, the most essential step is performed called stemming. In this, the common stem of the words is replaced and saved as a selected feature. For example, the word "move", "moves", "moved", and "moving" are replaced with "move" — the words which are gained after pre-processing called features.

3.3. Sentence Scoring

The output features of pre-processing phase are fed into the sentence scoring phase. The n-gram technique [16] is used to represent a vector of feature (word) counts. The unigram, bi-gram and trigram features are extracted to preserve the semantics of bug summary which reduced the sparsity of the text and calculates the probability (score) of each feature order. After that, these feature scores are combined to gain the sentence score.

3.4. Summary Percentage and Summary Subsets

According to the user input as a summary percentage, the subsets of the summary are generated by combining the sentences. The relative subset score is calculated for each summary subset.

3.5. Summary Subset Selection by PSO

After the formation of all possible subsets, these subsets are acted as the particles and their score as the initial position for the PSO algorithm [17] in summary subset selection phase. Assume that initially the particle position is

set to the rest and the particle velocity is revised at every iteration until the condition is not met by using Equation 1 and 2.

$$PV_d^{(t+1)} = s(t)PV_d^t + r_{p1}(t)\left(pbp_d - pcp_d(t)\right) + r_{p2}(t)\left(gbp_d - pcp_d(t)\right) \quad (1)$$

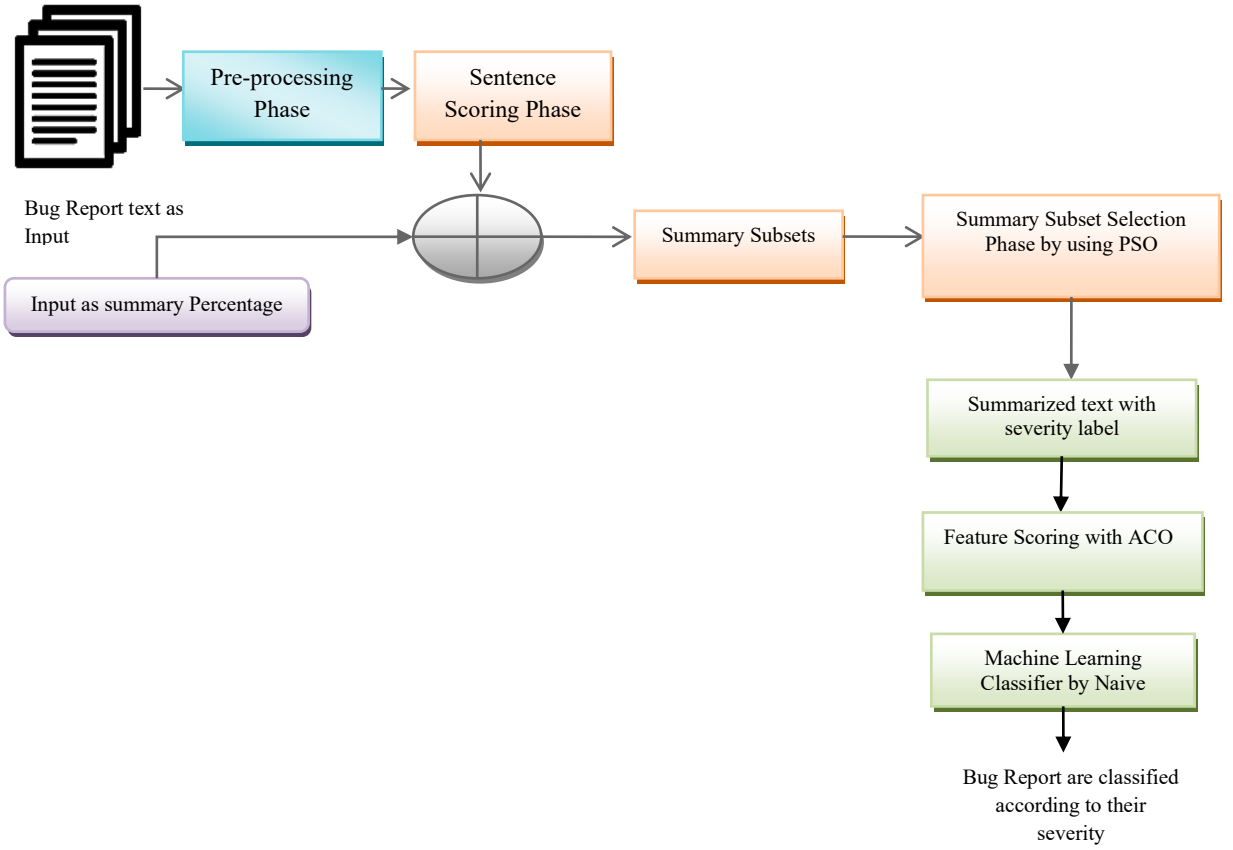

Figure 2 : The Proposed Model for Bug Report summarization and Bug Severity Classification Process

$$pcp_d^{(t+1)} = pcp_d^{(t)} + PV_d^{(t+1)} \quad (d = 1,2.....N) \quad (2)$$

Where, PV_d^t is represent as the old velocity, $PV_d^{(t+1)}$ is defined as the new velocity, $pcp_d^{(t)}$ is the current position , $pcp_d^{(t+1)}$ described as updated particle position , gbp_d is the global best position , $s(t)$ is the inertia weight , pbp_d is the personal best position, $r_{p1}(t), r_{p2}(t)$ are random numbers between [0,1]. The particle's current position (pcp) is compared with its personal best position (pbp), after every iteration. If the current position of the particle is better then, the personal best position is updated. Further, the pbp of all particles is compared with the global best position (gbp). If any particles pbp is better than gbp than, the gpb is updated. This process is continued until the termination condition is not satisfied with the fixed number of iterations. The condition in which the particle becomes homogeneous and best score does not vary for the specified number of iteration. At the end of every iteration, the numbers of subsets are reduced. The subset whose pbp is equal to gbp is the required optimal summary subset. If more than one summary subset met the termination condition, then the summary subset is chosen randomly. These summary subsets of features of each bug report are fed into the severity classification phase.

3.6. Feature Scoring with ACO

These features and their severity label are the input for the ACO [18] based feature scoring phase. The ACO is used to optimize the feature scores of each bug report. Initially, the random scores (weights) are assigned to each feature and number of ants are initialized. These ants are associated with each features column data so that every ant starts with random feature score. After that, the different combinations of scored features are created to trace the pheromone by using transition probability function, as shown in Equation 3.

$$Pro_A(T) = \frac{(\tau_A(T))^{\alpha}.\eta_A^{\beta}}{\sum_A(\tau_A(T))^{\alpha}.\eta_A^{\beta}} \quad (3)$$

In formula where η_β describe the background information of features to improve the results, $\tau_A(T)$ is pheromone amount for the Ath feature in time T, α and β are the control parameters that provide the pheromone and background information, and $Pro_A(T)$ is the Transition probability.

The feature which has high pheromone amount gets the higher score. The ants travel the features score probabilistically from these initial values until the stopping evaluation criteria of transverse is not satisfied. The scores of features are used to extract correlation information by updating the pheromones using Equation 4.

$$\tau_A(T+1) = \rho\tau_A(T) + \Delta\tau_A(T) \quad (4)$$

In this formula where, ρ defines as evaporation rate of pheromone trail and lies between [0, 1]. $\Delta\tau_A(T)$ is the pheromone trail amount added to Ath feature between time Δ and ΔT . In this work the, formula of updating pheromone trail was reformulated to calculate the $\Delta\tau_A$; where $\mu(\tau_A)$ is the mean of all pheromones trail amount added to Ath feature, $\frac{1}{\sigma(E_A)}$ is the standard deviation of accuracy.

$$\Delta\tau_A = \mu(\tau_A) + \frac{1}{\sigma(E_A)} \quad (5)$$

The resulting features scores are evaluated, and if features scores are not converged, and no changes are possible, then process leads to the termination. Afterwards, every optimized feature weight is used for learning. If weights are not unique, then the standard deviation for base prediction is calculated for relative weights of each feature and later fed to the machine learning classifier for training the model.

3.7. Machine Learning Classifier

The Naive Bayes [19] is used to classify the severity of the bug reports. The probability for each severity label is computed by using Equation 6. The probability distribution over the set of features is calculated by using Equation 7. At last the bug reports are classified, and the performance is evaluated.

$$P(x_n^d) = \frac{P(y_i)P(y_j)}{\sum_{i=1}^{c} P(y_i)P(y_j)}, \; j=1,2\ldots\ldots c \quad (6)$$

Where, P(yi)is the yi prior probability and P(yj)is the conditional class probability density function

$$P(x) = \prod_{i=1}^{k} P(c_i)P(x_n^d/c_i) \quad (7)$$

Where k is the number of classes, c_i is the i^{th} class, and $\mathbf{x_n^d}$ is the observed attribute values to certain class label c.

4. Experiment and Result

The evaluation metrics like Recall (R), Precision (P) and F-measure (F) [20] are used on the benchmark dataset of three open source projects [9] that are Mozilla, Firefox and Eclipse. The Mozilla and Firefox dataset has seven classes and Eclipse has five classes as shown in Table 2-5. The dataset is cut across into the testing and training dataset based on the 10-cross-validation mechanism. The different values are tested for PSO and ACO parameters. The result demonstrates that the best execution is accomplished by setting the parameters to values appeared in (Table 1).

Table 1: ACO and PSO parameters setting

Parameter	Value
Ants	100
Iterations	500
Initial Background Information	1
Initial Pheromone	1
$\boldsymbol{\rho}$	0.5
$\boldsymbol{\alpha}$	0.7-1
$\boldsymbol{\beta}$	2-5
Particles	100
Iterations	500
$\boldsymbol{s(t)}$	0.4-0.9
$\boldsymbol{r_{p1}(t)}$	1.9
$\boldsymbol{r_{p2}(t)}$	1.9

The performance of the proposed model is compared with the severity classification performance on the same dataset. For severity classification performance, the bug report text and its severity labels are the input, and then the text of bug report is pre-processed after that n-gram is used to extract the unigram and bigram. These n-grams are fed into the ACO feature weighting phase, and the output of this phase is fed into Naive Bayes classier to classify the severity of bug reports. The summary subset selected by PSO was more effective because PSO worked on semantic relation by using n-grams. Further the relative features weights are assigned to represent the relative feature learning. Take an example, the weight of a feature is 0.25, therefore, this feature learns only $^1/_4$times as compared to other feature and this feature has only $^1/_4$ role in decision making for any severity class.

Table 2-4 represent the comparison of results by using generated summaries for the severity classification process and bug report severity classification without using the summaries of bug reports. It can be seen that the performance of bug severity classification process with bug report summarization process is decreased in some cases and increased in some cases. The performance of the classifier is totally depended on the data of the bug reports. The bug reports which has a larger amount of comments and description of the bug, in that case, the performance of the classifier is increased. This is because the bug report summary can hold the vital information of the bug reports and also reduce the words (features) which can produce the noise for severity classification learning by the classifier. On the other hand, the bug reports which has less amount of description and comments, in that case, the important words (information) are lost, which are useful in the classification process.

Table 2 : Result analysis on Mozilla open source Project

Mozilla Classes	Bug severity classification with the summarization of Bug Reports			Bug severity classification without summarization of Bug Reports		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Blocker	76.18	75.98	76.57	76.50	79.28	78.37
Critical	79.65	78.10	79.37	79.25	77.16	78.69
Enhancement	79.74	78.79	79.76	76.50	75.22	76.36
Major	78.26	76.33	77.78	79.50	78.22	79.36
Normal	78.59	76.31	77.94	79.50	79.22	79.86
Minor	78.77	75.47	77.59	79.50	75.22	77.81
Trivial	81.74	79.45.	81.07	80.50	78.22	79.84

Table 3: : Result analysis on Firefox open source Project

Firefox Classes	Bug severity classification with the summarization of Bug Reports			Bug severity classification without summarization of Bug Reports		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Blocker	76.67	75.025	76.34	78.50	75.28	77.36

Critical	76.17	76.495	76.84	77.25	77.17	77.71
Enhancement	75.80	77.52	77.15	77.50	78.22	78.36
Major	77.70	80.42	79.54	76.50	79.22	78.34
Normal	75.30	76.02	76.15	76.50	79.22	78.34
Minor	75.30	74.52	75.41	76.50	75.22	76.36
Trivial	75.30	75.02	75.66	76.50	76.22	76.86

Table 4: : Result analysis on Eclipse open source Project

Eclipse Classes	**Bug severity classification with the summarization of Bug Reports**			**Bug severity classification without summarization of Bug Reports**		
	P (%)	**R (%)**	**F (%)**	**P (%)**	**R (%)**	**F (%)**
Blocker	79.58	75.39	77.93	79.50	77.28	78.88
Critical	81.58	77.70	80.03	81.25	76.16	79.13
Enhancement	80.00	76.37	78.64	82.50	77.22	80.27
Major	79.50	76.72	78.58	79.50	78.22	79.35
Normal	83.00	78.72	81.31	82.50	78.22	80.81

5. Conclusion

In this paper, to investigate the relationship between bug report summary and multiclass bug severity classification, a bug report summarization and bug severity classification model is proposed by using swarm intelligence and machine learning approaches. The proposed approach used the n-gram technique to extract the semantics features score. These features are fed into the Summary Subset Selection Phase to select the optimal summary subset by using Particle Swarm optimization. The selected subset features are fed into the feature scoring phase to provide a relative score to each feature by using Ant colony optimization. These optimized features are used to train the proposed model. At last, the Naive Bayes machine learning algorithm is used to classify the severity of bug report. The results are analyzed by using 10-fold cross-validation on three benchmark datasets and showed better performance in terms of Precision, Recall and F-measure. It was concluded from the results that summarization of bug reports some time help in the severity classification. . The performance is depended on the bug report contents. If, the bug report has more extensive data for summarization, than the summarization process increased the classification performance, else decreased. This is because relevant information is lost during the summarization process because the bug reports do not contain large data. In future, we investigate the relationship between summarization, severity classification and bug assignment.

References

[1] Canfora, G., & Cerulo, L. (2005). How software repositories can help in resolving a new change request. STEP 2005, 99.

[2] Ahsan, S. N., Ferzund, J., & Wotawa, F. (2009). Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on (pp. 216-221). IEEE.

[3] Sharma, Y., Dagur, A., & Chaturvedi, R. (2019). Automated Bug Reporting System with Keyword-Driven Framework. In Soft Computing and Signal Processing (pp. 271-277). Springer, Singapore.

[4] Angel, T. S., Kumar, G. S., Sehgal, V. M., & Nayak, G. (2018). Effective Bug Processing and Tracking System. Journal of Computational and Theoretical Nanoscience, 15(8), 2604-2606.

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

[6] Rastkar, S., Murphy, G. C., & Murray, G. (2010). Summarizing software artifacts: a case study of bug reports. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1 (pp. 505-514). ACM.

[7] Rastkar, S., Murphy, G. C., & Murray, G. (2014). Automatic summarization of bug reports. IEEE Transactions on Software Engineering, 40(4), 366-380.

[8] Kukkar, A. and Mohana, R., (2018). Feature Weighting with for Swarm Intelligence Optimization as a Tool for Bug Report Summarization. Journal of Advanced Research in Dynamical and Control Systems, pp. 2122-2133.

[9] Zhou, Y., Tong, Y., Gu, R., & Gall, H. (2016). Combining text mining and data mining for bug report classification. Journal of Software: Evolution and Process, 28(3), 150-176.

[10] Pandey, N., Sanyal, D. K., Hudait, A., & Sen, A. (2017). Automated classification of software issue reports using machine learning techniques: an empirical study. Innovations in Systems and Software Engineering, 13(4), 279-297.

[11] Goseva-Popstojanova, K., & Tyo, J. (2018). Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification. In 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 344-355). IEEE.

[12] Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software, 117, 166-184.

[13] Sharmin, S., Aktar, F., Ali, A. A., Khan, M. A. H., & Shoyaib, M. (2017). BFSp: A feature selection method for bug severity classification. In Humanitarian Technology Conference (R10-HTC), 2017 IEEE Region 10 (pp. 750-754). IEEE.

[14] Kukkar, A., Mohana, R., Nayyar, A., Kim, J., Kang, B. G., & Chilamkurti, N. (2019). A Novel Deep-Learning-Based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting. Sensors, 19(13), 2964.

[15] Liang, J., Koperski, K., Dhillon, N. S., Tusk, C., & Bhatti, S. (2013). NLP-based entity recognition and disambiguation U.S. Patent No. 8,594,996. Washington, DC: U.S. Patent and Trademark Office.

[16] Wang, R., Zhao, H., Lu, B. L., Utiyama, M., & Sumita, E. (2015). Bilingual continuous-space language model growing for statistical machine translation. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23(7), 1209-1220.

[17] Kennedy, J. (2011). Particle swarm optimization. In Encyclopedia of machine learning (pp. 760-766). Springer US.

[18] Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. IEEE computational intelligence magazine, 1(4), 28-39.

[19] Mitchell, T. (2002). Machine Learning. McCraw Hill, 1996. 93 D. Monière et D. Labbé. Essai de stylistique quantitative. In JADT (pp. 561-569).

[20] Huang, Y. J., Powers, R., & Montelione, G. T. (2005). Protein NMR recall, precision, and F-measure scores (RPF scores): structure quality assessment measures based on information retrieval statistics. Journal of the American Chemical Society, 127(6), 1665-1674.