2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and
2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)

Bug Severity Classification Based on Class-Membership Information

Sayeda Shamma Alia
and Md. Nazmul Haque
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bit0427@iit.du.ac.bd
bsse0635@iit.du.ac.bd

Sadia Sharmin
Dept. of Computer Science and Engineering
East West University
Dhaka, Bangladesh
sadia.sharmin@ewubd.edu

Shah Mostafa Khaled
and Mohammad Shoyaib
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
khaled@du.ac.bd
shoyaib@du.ac.bd

***Abstract*—Now-a-days bug classification along with its severity prediction has become an important issue for better maintenance and cost reduction of software. Different approaches have already been introduced in this regard including topic modeling, concept profile etc., however, most of them require a lot of parameter tuning, different types of computations for identifying bug severity. Furthermore, for reduced computational cost many of the state-of-the-art methods use standard Naïve Bayes for classification. However, priors used in this case may degrade the results due to the unavailability of training data. Moreover, cross project performance has rarely been discussed. To address these issues, we introduce a method namely Class-Membership Information of a Term (CMT) that does not use any priors, is computationally simple, free from parameter tuning and performs better compared to the existing methods. Rigorous experiments based on three benchmark datasets demonstrate that CMT on an average performs at most 5% and 12.5% better than other state-of-the-art methods considering within and cross project classification respectively.**

***Keywords*—component; CMT; bug severity; classification;**

I. Introduction

A bug represents an error or an unexpected behavior of a software and a bug report contains all the necessary information about that bug. There are some bug reporting/tracking systems (e.g., Bugzilla[1]) which is used to manage the bug reports. It provides several information, such as, bug ID, summary and description of that bug, product and component name, reporter name, priority, etc.

Recently the increasing demand of software development is being observed due to its various applicability in scientific, business and daily-life activities. The development, specially the maintenance of these software, involves more than 85% of the total cost [1] and manual effort. Usually the developers report a lot of bugs, most of which are not real. Therefore, a manual judgment is required to identify whether a reported bug is really a bug or not. If it is really a bug, the severity level should also be judged to assign that bug to an eligible bug fixer.

To reduce the cost of software development and the burden of manual judgment, researchers have been working for several years. The early works mainly focus on automatic decision on whether a given bug is severe or not [2], [3], [4], and later, the researchers pay attention to find the level of severity of these bugs [5], [6], [7], [8], [9], [10]. For both cases, similar algorithms are proposed as discussed in the following parts of this section.

A. Severe Bug Identification

One of the earliest works [11] that divide bug reports into severe and non-severe, uses the summary of the bug reports and Naïve Bayes (NB) as a classifier under the assumption that there exists potentially significant terms in the bug reports with good discrimination ability of severity which is also assumed commonly by other researchers. Their classification has been done per component basis and obtained reasonable accuracies. They also compare the performances of summary and descriptions separately and find that summary performs better than description. Later the authors [2], have compared four well known classification algorithms and concluded that Multinomial Naïve Bayes (MNB) performs the best for severity prediction. This is even better than Support Vector Machine (SVM).

Instead of using only summary, the authors in [12] suggest to incorporate more source of information and use MNB as a classifier to improve the performance. They first group the bug reports based on their Product followed by grouping them using Component and Reporter. Then information from summary and description are extracted. Similar observation is also found in [13] where the authors have used five different information namely summary, two types of structural information (extracted from long description) and two other information such as attachment and report length. They also use MNB as classifier and calculate the posterior ($P(\psi_k \mid t_1 \cdots t_n)$) using Eq. 1 and conclude that combining all these five information improves the overall performances.

$$P(\psi_k) \prod_{i=1}^{n} P(t_i \mid \psi_k) \tag{1}$$

here, ψ_k represents the severity levels and $t_1 \cdots t_n$ are different terms of the bug report.

[1]https://www.bugzilla.org/

978-1-5386-5163-6/18/$31.00 ©2018 IEEE

Use of Topic modeling is introduced in [14] where, Latent Dirichlet Allocation (LDA) is used to extract the topics and the corresponding topic terms. They use KL divergence based on smoothed Unigram Model to find out the similarity of a new bug report with the historical bug reports and take these similarity measures into k-Nearest Neighbor (k-NN) to find the top k-similar training bugs having same topic and multi-features (such as product and component) to identify the severity of the bug reports. In [15], the authors have prepared a dataset having 59 features collected from 163 bug reports and suggested to use the existing classifiers such as NB and random forest along with Adaboost classifier. It is found that boosting improves the performances of the existing classifiers.

Instead of using all the terms (features) of bug reports directly, several researchers suggest to use feature selection to reduce the dimension of the features [3], [10], [16]. More importantly such feature selection helps to reduce noisy features and improve the overall performances [17]. In [16], the authors use a subset of bi-grams and χ^2 feature selection using NB for classification. They conclude that using bi-grams along with uni-gram improves the results and feature selection has a great impact on the performance. In [3], the authors compare two different feature selection strategies namely info-gain [18] and χ^2 [19] based feature selection which select features from the Term-Document Matrix filled by TF-IDF (term frequency and inverted document frequency) score and consider only 125 features for classification. They also compare two different classifiers namely MNB and k-NN and conclude that k-NN performs best.

B. Fine-grained Bug Severity Identification

Other than dividing the bug reports into two severity levels, there are several recent work that identify more fine grained levels of severity of the reported bug. The method described in [20] divide the severity into three levels and has two stages. In the first stage, the authors use only summary of the bug reports (as description may degrade the performance) and classify them using MNB. In the second stage, they use the output of the first stage along with some structured information such as priority, severity and component information. They use Bayesian Network classifier in the second stage and data grafting to process the output of the first stage and prepare the input of the second stage.

In [21], the authors use a similarity function namely REP that use four types of features and weights followed by the use of k-NN to extract top k nearest neighbors and predict the severity levels. They also propose an extension of BM25F (which is a function of finding similarity between two documents and mainly use TF and IDF for its calculation) to calculate the similarity of two longer documents and use that to calculate REP. However, they use summary, description, product and component information for severity calculation. This REP requires 16 parameters and finally they classify the bug into five different severity levels. Similar approach is also followed in [7] where they perform topic modeling and propose $\text{REP}_{\text{topic}}$, which is the extension of the REP used in the previous paper and have 17 parameters. They show that their method performs best when the value of k is 5 during the use of k-NN classifier.

Apart from topic modeling, concept profile (CP) is proposed for the prediction of severity, which depends on four tuples namely severity status, a threshold value, a set of concept terms and bug reports [5]. Further, as done in topic modeling [14], this paper also use smoothed UM for probability vector transformation for CP and KL-divergence for finding the similarity and decide the severity of the new bug report that has the highest similarity. In [22], the authors also use TF-IDF as feature taken from the summary part of bug reports as it contains most useful information and compared five different classifiers. They also suggest to use information gain for feature selection and found that use of top 125 features stabilizes all the classifiers except NB.

Unlike the aforementioned works, there are other few interesting works that mainly address feature selection [10], give emphasis on emotion terms [9] or deal with data imbalance [8]. EWD-multinomial method uses different types of source of information, emotion word-based dictionary and MNB [9]. Mutual information based feature selection is investigated in [10] where the performance of SVM and decision tree is also compared for within project and cross project.

Two major findings can easily be observed by analyzing the methods discussed so far. First, feature selection is necessary to improve performance. However, in this paper, instead of feature selection, we mainly focus to improve the performance by using only the textual information found in the bug reports. Second, a significant number of work focuses on a lightweight solution that mainly uses frequency based approaches for the calculation of feature and NB/KNN as a classifiers.

One of the major drawbacks of NB based approaches in this regard is- its prior calculation. Because, the data we have in this case are very imprecise, i.e., in many cases data of different severity levels are not found for the earlier version of a software and also the number of bugs and the containing terms are not adequate for proper training and thus priors mislead the classifiers in many cases. Beside this, to improve the performances, different types of calculation (such as topic modeling) and parameters (such as weights in CP and $\text{REP}_{\text{topic}}$) are introduced. However the values of these parameters are not easy to determine.

Moreover, the cross project performances are rarely addressed in the existing literature whereas it is very much necessary for the identification of the severity levels especially for the newly released software. To address the aforementioned issues, our contributions in this paper are summarized as follows:

- We propose to use Class-Membership Information of a Term (CMT) for predicting the bug severity
- CMT is parameter free and has linear complexity
- A CMT based weight generation mechanism is introduced, which can be used in general
- Cross project performances are examined

II. Proposed Method

We propose class-membership of a term (CMT) as a general text classification method and describe it for bug severity prediction in this paper. It comprises of three tuples (ψ, μ, β), where ψ represents a severity level (e.g.,"Blocker") of a bug, μ is the degree of membership of a term to a particular severity level (defined in Eq. 2) and β is a set of given bug reports with different severity levels (ψ).

A bug report is generally comprised of several words. To classify the bug reports based on these words, we preprocess them applying standard Natural Language Processing (NLP) based techniques. In this paper, a bug report is thus tokenized at first by splitting a sentence into pieces of words, stop words (e.g., a, the) are then removed as they do not contribute in classifications. Finally, stemming [23] is applied to convert the words into their root form. A term (word) of a bug report may belong to different severity levels. It is generally expected that there exists a set of terms from where the appearance of a term is more likely for a particular (or a set of) severity level of a bug under the assumption that terms of a bug report has some discrimination ability. We further assume that each of these terms is independent and the dataset that contains the terms is imprecise. This is a valid assumption because it is natural that the earlier version of a software bug report may not contain different types of bug reports and enough terms to represent all types of bug severities. Again, all the terms from a set of bug reports do not have the same capability to describe a severity level, or in other words, the capability of a term to be a member of a different severity level is different. To calculate this capability we propose to use CMT, which is defined below:

Definition 1. CMT In Eq. 2, $\mu_{i,j}$ defines CMT of a term t_i for a given severity level ψ_j,

$$\mu_{i,j} = \frac{P(t_i \mid \psi_j)}{P(t_i \mid \neg\psi_j)} \tag{2}$$

here, $P(t_i \mid \psi_j)$ is the probability of a term $t_i (i = 1 \cdots n)$ for a given severity level $\psi_j (j = 1 \cdots m)$ and $P(t_i \mid \neg\psi_j)$ is the probability of that term given all severity levels other than ψ_j.

Properties of CMT: The major properties of CMT are:

- CMT can determine the degree of membership [0, ∞) of a term for a particular ψ. The higher the value of CMT, the higher the chance of the term to be a member of that ψ.
- Thus through CMT, a term can be a member of multiple ψ but in different/same degree. This means CMT of a term emphasizes or de-emphasizes a bug report to be a member of a particular severity level.

To understand these properties, let us consider Fig. 1, where ψ_1 and ψ_2 are two severity levels and t_1, t_2, t_3, t_4 are four terms associated with these severity levels. The weight of each edge represents the degree of membership of a term for a particular μ_j. For example, t_1 has higher degrees of membership to ψ_1 than t_1 has with ψ_2 and t_3 has same degrees of membership for both ψ_1 and ψ_2.

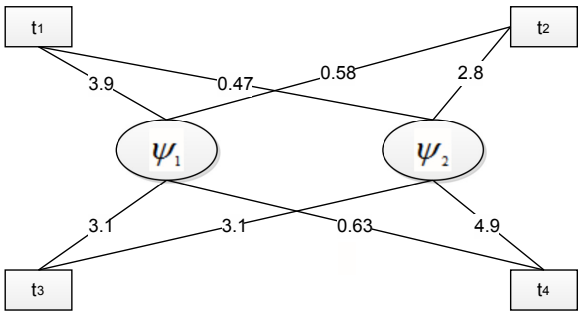

Fig. 1: An example of CMT using Bug reports β

Use of CMT for severity prediction: During training we calculate $\mu_{i,j}$ for all the terms in the training data. For predicting the severity level of a new bug, we first calculate the product of all $\mu_{i,j}$'s for a given bug report and take decision in favor of that ψ_j for which μ_j is maximum following Eq. 3 and Eq. 4.

$$\mu_j = \prod_{i=1\cdots n} \frac{P(t_i \mid \psi_j)}{P(t_i \mid \neg\psi_j)} \tag{3}$$

$$d = \underset{j}{\operatorname{argmax}}\, \mu_j, \forall j = 1 \cdots m \tag{4}$$

Usually bug reports found in Bugzilla are associated with several types of information, such as summary and description of a bug report, their product and component name. In this paper, we suggest to incorporate product and component information as a weight along with summary and description information for better prediction accuracies. We also use Eq. 3 for generating these weights (for product/component) and the final decision taken using Eq. 5 and Eq. 6

$$D_j = (\mu_j^P + \mu_j^C) \times (\mu_j^S + \mu_j^D) \tag{5}$$

$$D = \underset{j}{\operatorname{argmax}}\, D_j, \forall j = 1 \cdots m \tag{6}$$

where, μ_j^P, μ_j^C, μ_j^S and μ_j^D are the CMT of product, component, summary and description respectively for a particular ψ_j and D is the final decision which is the maximum among all D_j. It is noteworthy to mention here that as long as the name of a product or a component is a single word term, the product term in Eq. 3 for generating these weights is not required.

Based on the aforementioned process, we identify the severity levels of a bug and answer the following research questions:

RQ1: What is the performance of CMT in comparison to other methods?

RQ2: Does the product and component information help to improve the performance of severity prediction? If yes, is it possible to build up a generalized way to do this? If no, why?

RQ3: What is the generalization ability of the existing data for the prediction of new bug in different projects?

RQ4: Which criteria is more suitable between summary and description for severity prediction?

RQ5: Is it possible to build up a lightweight solution with improved performance compared to the existing methods?

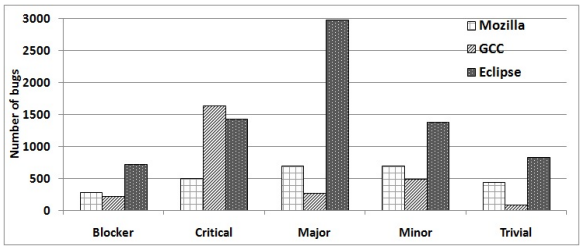

Fig. 2: Total number of bugs for different severity levels of three different projects

III. Experimental Results and Discussion

In this section, we first briefly describe the datasets and the implementation protocol that have been followed in our experiment. The experimental results along with relevant discussions and the answers to the research questions are then discussed.

A. Dataset Description and Implementation Details

We have used three open source projects namely Eclipse (E), Mozilla (M) and GCC (G) which are also employed in [7] and they make their source code and dataset[2] public for ease of comparison. There are seven severity levels of bug reports for each of projects. However, according to the experimentation protocol followed in [7], we have used five levels: Blocker, Critical, Major, Minor, Trivial. Because, the rest two namely Normal and Enhancement are not really bugs. Fig. 2 represents the total number of bug reports of E, M and G for each severity level. For training and testing, we first sort the bug reports in chronological order and then divide the reports into 11 non-overlapping segments of equal sizes. A special variant [7] of 10-fold cross validation (10-CV) is then performed. Here, at first, segment 1 is used for training and segment 2 for testing. Then, training is conducted using segment 1 and 2, and testing with segment 3. Following this way, the final training is performed using segment 1 to 10 and testing with segment 11. The average F-measures (defined in Eq. 7) of these 10-CV are reported for all the methods.

$$F - measure = 2 \times \frac{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} \quad (7)$$

here, TP, FP and FN represents true positive, false positive and false negative respectively.

B. Results and Discussion

In Table I and Table II, we compare the proposed CMT with two other state-of-the-art approaches namely standard NB based method and REP_{topic} [7]. Here, C_S and C_D represents CMT using summary and description respectively. **Answer to RQ1:** In Table I, we have shown the average F-measure of 10-CV for each severity level using all the datasets where NB, REP_{topic} and CMT win for 3, 4, and 8 cases respectively. It demonstrates the superiority of the proposed method.

[2]https://github.com/ProgrammerCJC/SPFR

TABLE I: Performance Comparison of CMT with other methods

	C_S	C_D	C_S + C_D	CMT_W	NB	REP_{topic}	CMT
	Blocker						
E	20.67	21.33	22.85	23.76	16.74	5.67	**23.86**
M	29.75	24.44	29.36	24.75	22.27	1.38	**30.03**
G	26.16	23.41	28.09	28.52	**28.18**	27.66	27.83
	Critical						
E	26.95	24.09	25.85	26.35	27.16	**28.42**	26.07
M	43.92	38.98	43.25	42.58	34.47	37.09	**42.73**
G	62.09	69.32	65.17	65.9	71.48	**76.49**	65.96
	Major						
E	25.59	25.08	24.22	25.64	31.14	**51.39**	25.69
M	36.85	34.87	35.44	34.84	38.20	**39.30**	35.22
G	11.89	8.82	10.22	10.99	6.59	6.85	**11.77**
	Minor						
E	33.84	36.42	37.59	38.58	33.37	36.35	**38.51**
M	30.67	36.78	34.68	32.25	**38.82**	37.77	36.01
G	40.59	31.77	35.16	35.68	31.12	10.57	**36.10**
	Trivial						
E	41.54	39.45	41.83	43.10	37.05	25.55	**43.26**
M	46.43	36.29	39.59	38.89	**41.97**	32.71	40.01
G	9.91	7.87	9.12	9.12	5.99	2.87	**8.17**

However, in few cases REP_{topic} performs much better than CMT. For example, in case of the severity level "Major", its performance is around twenty five percent better than CMT. This is because this level contains most of the bug reports and this method is biased to that level, whereas for most of the other severity levels (where data is small) its performances are not as good as CMT. Beside this, the differences in performances for different classes are much small for CMT compared to other methods. Table II represents the average of F-measure of these five severity levels given in Table I. From this table, it is evident that on an average CMT performs better compared to other state-of-the-art methods.

Answer to RQ2: Every bug report of a software has its own product and component name and it is expected that these two possess information about the severity of bug. For example, if the product name is "Linux" and the component is "Kernel" then the bug may be severe. From Table I, it is also observed that the incorporation of the product and component information help in improving the performance of severity prediction (CMT vs C_S + C_D in Table I). In Table I, CMT_W represents the use of fixed weights for product and component along with CMT (instead of the proposed weight) which are empirically found for different software and used in [7]. Comparing CMT_W and CMT it is observed that both of these two generate similar results which indicate that similar weight can be produced by a generalized weight generation mechanism that uses CMT and validates the proposed method.

TABLE II: Comparison of different methods based on their average F-measure(%)

	NB	REP_{topic}	CMT
Eclipse	29.09	29.48	**31.48**
Mozilla	35.67	29.65	**36.80**
GCC	26.29	24.89	**29.97**

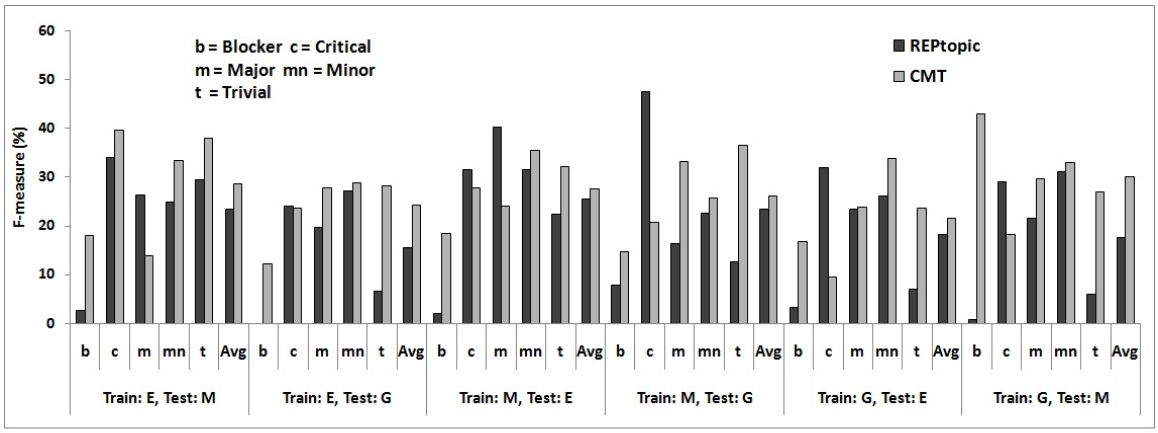

Fig. 3: Cross project performance comparison

It is noteworthy to mention here that the use of the generalized weight generation method proposed in this paper is much more acceptable than any empirically found constant values specific to the particular software.

Answer to RQ3: In the existing literature, it is usually found that bug severity prediction methods are trained and tested using a single project with different version (within project testing). However, it is necessary to test the bug severity during the release of the first version of a software and thus demands a bug severity prediction method that can be used in general, i.e, during the training phase bug severity of few software is used and can be applied to new software (cross project testing). Fig. 3 represents the cross project performances comparison of CMT and $\text{REP}_{\text{topic}}$. The F-measure of these two methods for each severity level during each cross-project experiment indicates that the proposed method performs better for most of the cases. Furthermore, CMT wins for all six cases when the average F-measure is considered. Among them, two wins are significant (by performing paired t-test). Even though the cross project performances are not yet satisfactory for any method, the results in Fig. 3 indicate that the CMT has better generalization capability than other methods.

Answer to RQ4: We compare the performances of summary (C_S), description (C_D) and their combination (C_S + C_D) using CMT separately. From Table I, it is observed that CMT on summary shows better results than description in most of the cases, which is also asserted by other researchers [11], [21], and the improvement is not significant when their combination is used. However, few existing literature asserts that description shows better performance than summary [10]. One of the main reason behind the failure of description is that, it contains noisy feature and feature selection is required. We use both summary and description as it does not degrade the result and to present a generalized formation of bug severity identification. In future, we plan to incorporate feature selection for obtaining better performance. Besides this, improved result is observed when we add product and component information (Table I).

Answer to RQ5: Yes, it is possible to build up a lightweight solution for bug severity prediction with improved performance compared to the existing methods. In CMT, we use ratio of two probabilities (likelihoods). Since the calculations of probability is linear, the complexity of CMT is also linear. Similar ratio based methods can also be found in other literature [24], however in this paper, we define CMT for bug severity classification. Similar complexity is also observed for NB. However, due to the misleading priors its performance degrades. To understand this issue let us consider Example 1.
Example 1 In Table III contains two severity levels (ψ_1 and ψ_2) and three terms (t_1, t_2 and t_3). Further, there are 100 and 5 bug reports (β) for ψ_1 and ψ_2 respectively. Now consider that we have a new bug report that have only term t_1 and t_2. If we take decision using Eq. 1, it gives decision in favor of ψ_1 whereas it is reasonable to take decision in favor of ψ_2. Because, even though their occurrences are same but the total terms for ψ_1 is 112 and for ψ_2 is only 62 and thus ψ_2 is more probable comparing to ψ_1. Such a desirable result can be obtained using the proposed method. In NB classifiers, the prior for ψ_2 will be estimated as 0.048, since there are only five bug reports. This small prior actually forces NB to give the decision in favor of ψ_1. Therefore, in case of unavailability of bug reports one should not make use of such small priors. In real life scenario, along with the dataset used here, have the same dataset.

Thus, it is reasonable to consider equi-likely prior and use CMT which is the main reason behind the success of the proposed method. It is observed that the proposed method also performs well even when there is small amount of data.

TABLE III: Example Data

Severity Level	t_1	t_2	t_3	Total Terms	β
ψ_1	2	10	100	112	100
ψ_2	2	10	50	62	5
Total	4	20	150	174	105

For example, during experiment with GCC dataset, we found that critical and trivial class have 1644 and 87 bug reports respectively. Although the trivial class has very small amount of data CMT performs well compared to the existing ones.

C. Threats to Validity

We use the experimental data which is also used in [7]. As long as they collect a large set of data from three large scale projects, it is believed that the proposed approach may be effective to other open source projects and thus, the threat is relatively small in this case. However, during cleaning process the data used in this paper and the data used in [7] might not remain exactly same and thus results may differ. Furthermore, the bug reports and their severity levels are determined by human and thus, we can not ignore the chance of human error. In this paper, we assume equi-likely prior and believe that our assumption is reasonable and obtain better results in this case. However, there may exist some cases where this assumption might not work. Beside this, we have also done cross project experiments to show the generalization ability of the proposed approach. However, such a generalization can not be guaranteed for other projects as we have only empirical evidence for six cross project experiments using three datasets.

IV. Conclusion

In this paper, we propose CMT for classifying the software bugs based on their severity and it achieves better performance compared to other state-of-the-art methods. The complexity of CMT is linear and it is parameter free. Moreover, instead of using empirical weights, the same CMT can be used for generating a weight to emphasize (or de-emphasize) a particular severity level based on the information in hand. The proposed CMT thus can be used in general both for NLP based classification applications and for generating respective weights, especially when dataset is imprecise. Beside this, we believe the use of an appropriate feature selection mechanism along with CMT will increase the overall performance, which will be addressed in future. Furthermore, even though several efforts have already been taken for the identification of the severity levels automatically, adequate performances specially for cross project are still not achieved.

Acknowledgment

This research is supported by ICT Division, Ministry of Posts, Telecommunications and Information Technology, Bangladesh. 56.00.0000.028.33.079.17-223. This work is also supported by the University Grants Commission, Bangladesh under the Dhaka University Teachers Research Grant No Reg/Admin-3/54295.

References

[1] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT professional*, vol. 2, no. 3, pp. 17–23, 2000.

[2] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 249–258.

[3] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Computer Science*, vol. 70, pp. 632–639, 2015.

[4] P. Kaur and C. Singh, "A systematic approach for bug severity classification using machine learnings text mining techniques," *Intl. Journal of Comp. Science and Mobile Computing*, vol. 5, pp. 523–528, 2016.

[5] Z. Tao, Y. Geunseok, L. Byungjeong, and T. C. Alvin, "Predicting severity of bug report by mining bug repository with concept profile," *30th Annual ACM Symposium on Applied Computing(SAC '15), Proceedings of*, pp. 1553–1558, 2015.

[6] M. N. Pushpalatha and M. Mrunalini, "Predicting the severity of bug reports using classification algorithms," in *Circuits, Controls, Communications and Computing, Intl. Conference on*. IEEE, 2016, pp. 1–4.

[7] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[8] N. K. S. Roy and B. Rossi, "Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results," in *Software Engineering and Advanced Applications, 43rd Euromicro Conference on*. IEEE, 2017, pp. 426–429.

[9] Y. Geunseok, B. Seungsuk, L. Jung-Won, and L. Byungjeong, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects." ACM Digital Library, 2017, pp. 1280–1287.

[10] S. Sharmin, F. Aktar, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "Bfsp: A feature selection method for bug severity classification," in *Humanitarian Technology Conference (R10-HTC), 2017 IEEE Region 10*. IEEE, 2017, pp. 750–754.

[11] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 1–10.

[12] K. Jin, A. Dashbalbar, G. Yang, J.-W. Lee, and B. Lee, "Bug severity prediction by classifying normal bugs with text and meta-field information," *Advanced Science and Tech. Letters*, vol. 129, pp. 19–24, 2016.

[13] C.-Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, "Improving severity prediction on software bug reports using quality indicators," in *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*. IEEE, 2014, pp. 216–219.

[14] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Computer software and applications conference (COMPSAC), 2014 IEEE 38th annual*. IEEE, 2014, pp. 97–106.

[15] A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, "Severity prediction of software bugs," in *Information and Communication Systems, 2016 7th International Conference on*. IEEE, 2016, pp. 92–95.

[16] N. K. S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Software Engineering and Advanced Applications, 40th EUROMICRO Conference on*. IEEE, 2014, pp. 269–276.

[17] S. Sharmin, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "Feature selection and discretization based on mutual information," in *Imaging, Vision & Pattern Recognition, IEEE Intl. Conf. on*, 2017, pp. 1–6.

[18] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," *ICML*, vol. 97, pp. 412–420, 1997.

[19] W. J. Dixon and J. M. Frank, *Introduction to statistical analysis*. McGraw-Hill, 1969, vol. 344.

[20] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.

[21] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Reverse Engineering, 19th Working Conference on*. IEEE, 2012, pp. 215–224.

[22] K. Chaturvedi and V. Singh, "Determining bug severity using machine learning techniques," in *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. IEEE, 2012, pp. 1–6.

[23] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[24] M. Shoyaib, A.-A.-W. M, C. Oksam, and B. Ryu, "Skin detection using statistics of small amount of training data," *Electronics Letters*, vol. 48, no. 2, 2012.