Predicting Severity of Bug Report by Mining Bug Repository with Concept Profile

Tao Zhang[†] Geunseok Yang[‡] Byungjeong Lee[‡*] Alvin T.S. Chan[†*]
[†]Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
[‡]Department of Computer Science, University of Seoul, Seoul 130-743, South Korea
cstzhang@comp.polyu.edu.hk, {ypats87, bjlee}@uos.ac.kr, cstschan@comp.polyu.edu.hk

ABSTRACT

Recently, for large scale software projects, developers rely on bug reports for corrective software maintenance. The severity of a reported bug is an important feature to decide how fast it needs to be fixed. Therefore, to arrange a new submitted bug to an appropriate fixer, it is necessary to recognize the severity of each bug report. Unfortunately, reporters need to decide the severity of bugs manually. Even if there are guidelines on how to verify the severity of a bug, it is still a time-consuming work. Utilizing the concept profiles by mining bug repositories is a good way to resolve this problem. In this paper, we propose a concept profile-based prediction technique to assign the severity of a given bug. In detail, we analyze historical bug reports in the bug repositories and build the concept profiles from them. We evaluate the performance of our method on the bug reports from the bug repositories of popular open-source projects that include Eclipse and Mozilla Firefox, the result shows that the proposed technique can effectively predict the severity of a given bug.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Metrics—*Distribution, Maintenance, and Enhancement*

Keywords

mining bug repository, concept profile, severity prediction, bug report, bug triage, software maintenance

1. INTRODUCTION

Recent years software maintenance has become a challenging task because bug fixing consumes large amounts of project funding each year [1]. Therefore bug tracking systems (e.g., Bugzilla) [2] have been developed to help developers track and fix bugs by managing submitted bug reports each day. When a new bug report comes, a senior developer called triager assigns an appropriate developer for fixing it. This process is named bug triage [3]. During bug triage, the triager can arrange the most urgent bugs to the fixers preferentially according to the status of severity. As a feature of the bug report, severity is an indicator for deciding how soon it needs to be fixed [4]. High severity (e.g., blocker, critical and major) represents the serious errors and crashes, and low severity (e.g., normal, minor and trivial) represents the cosmetic bugs. However, severity decision is still a manual process. It is a time-consuming work for the developers who report bugs. In addition, the reporters' experience may affect the accuracy of severity decision.

For resolving these problems, in this paper, we propose a concept profile-based severity prediction technique by mining historical bug reports in the bug repository. A concept profile (CP) [5] consists of the concepts with concept terms. We consider the different statuses (e.g., blocker and minor) of severity as the concepts. Mining bug repository is an effective way for extracting the concept terms from the historical bug reports on each status. By utilizing the CPs, we can verify which status of severity a new given bug belongs to. The proposed approach is expected to be employed in large-scale software projects which utilize bug reports to track and fix the corresponding bugs. We summarize the major contributions of this paper as follows.

- We utilize a textual classification technique called CP to predict the severity of a given bug by mining the historical bug reports in the bug repository. In detail, we extract the concept terms from the historical bug reports for building the CPs. Once a new bug comes, by computing the similarity between the bug report and the severity concept in CPs, we can find the category which the given bug belongs to.

- We evaluate the performance of the proposed severity prediction algorithm on the bug reports mined from Eclipse and Mozilla Firefox and compare our approach with three classic prediction algorithms that include KNN [4], Naive Bayes [7] and Naive Bayes Multinomial [8] . The evaluation result shows that the proposed severity algorithm performed better than them.

The paper is organized as follows. First, Section 2 introduces the related work proposed by other researchers. Next, we detail our severity prediction technique in Section 3. After that, the evaluation result and corresponding discussion are shown in Section 4. Then, we introduce some threats to

*Corresponding author

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695872

validity in Section 5. Finally, we summarize this paper and point out the future work in Section 6.

2. RELATED WORK

In the previous studies, some researchers utilized a series of methods for predicting the severity of the given bug reports. Menzies and Marcus designed and built a tool named SEVERIS which is based on a rule learning technique [6]. Lamkanfi et al. investigated whether they could accurately predict the severity of a reported bug by analyzing its textual description [7]. Based on the evaluation results on three cases from Mozilla, Eclipse and GNOME respectively, they demonstrated that it was possible to predict the severity with a reasonable accuracy. In a later work, Lamkanfi et al. used four machine learning algorithms including Naive Bayes, Naive Bayes Multinomial, K-Nearest Neighbor and Support Vector Machine for predicting the severity of a reported bug [8]. The evaluation results showed Naive Bayes Multinomial performed better than the other algorithms. Tian et al. first measured the similarity of different bug reports using BM25 document similarity function, and then based on a set of k nearest neighbors, the severity labels of these k similar bug reports were used to decide the appropriate severity label for a new bug report [4]. Bhattacharya et al. [9]constructed a set of graph metrics that captured interesting properties of these graphs via using two features that include the product and the process of bug reports. Yang et al. [10] proposed a method to predict the severity of given bugs by utilizing topic model and multi-feature. The results of experiments on Eclipse and Mozilla showed that the proposed approach can effectively predict the severity of the given bugs.

Our present study is different from these previous research. We build the CPs for each severity by mining historical bug reports that have different statuses of severity. The proposed algorithm does not depend on the classical machine learning algorithms such as KNN and Naive Bayes, but performs better than them.

3. METHODOLOGY

We introduced CP as a text classification method in the early research for arranging a new bug report into an appropriate category [5]. Different from the early research which utilized CP and social network for executing bug triage, our current work refines CP and focuses on severity prediction. A CP is formed by a 4-tuple (s, θ, C, B) where s denotes a severity status, θ is a threshold value, C is a set of concept terms, and B represents a set of bug reports related to the severity status. Figure 1 shows an example of a concept profile. As a severity status 'blocker' (s), when the weight values of the terms 'eclipse'(0.0278), 'jdt'(0.0241), 'compiler'(0.0227) and 'debug'(0.0211) are more than a threshold value ($\theta = 0.011$), we choose them (C) as the concept terms. The set of bug reports B (b_1, b_2,. . . , b_n) whose severity statuses are 'blocker' forms a CP with s, θ and C. The core idea of CP is different from topic model [10] because CP is a classification algorithm while topic model is a clustering technique. One bug report can only belong to one concept in CP, but it can belong to several topics in topic model. We believe that CP is suited to be utilized for predicting the severity of the new bug.

We predict bug severity according to the following steps:

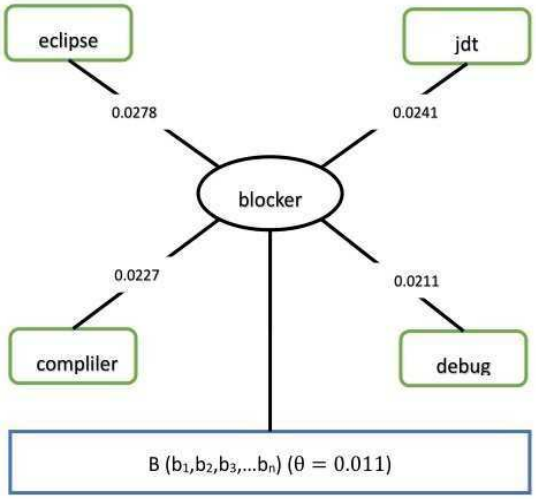

Figure 1: An example of concept profile

- **Bug reports collection and pre-processing**: At first, by collecting the historical bug reports which have the different status of severity in the bug repository, our data sets that include the training sets and the testing sets are formed. Then we pre-process the bug reports in the training set using Natural Language Processing (NLP) techniques [11] such as tokenization, stemming and stop words removal. Tokenization is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. Stop words are insignificant words for information retrieval task. These words include pronouns such as 'it' and 'she', and link verbs such as 'are' and 'is'. Stemming is the process to transform the words to their ground forms. For example, a stemmer can transform both 'loves' and 'loving' to 'love'.

- **Concept terms selection**: After collecting and preprocessing the bug reports, we select the concept terms that appear at a high frequency in the bug reports belonging to the same CP. In detail, we first normalize the occurrence frequency of each term in the bug reports to a corresponding weight value. Then if the weight value of the term is more than a predefined threshold, we select this term as a concept term. This selection process finishes when the terms in each concept are selected. We decide the threshold which can produce the best performance for severity prediction, which is described in Table 1. This table shows all of severity concepts with concept terms based on the bug reports in Eclipse and Mozilla Firefox data sets. The reason of choosing the different thresholds is that they can produce the highest F measure. The concept terms of each CP are shown in the last column. The values in the parenthesis present the weight of the relevant terms. Obviously, the weight value of the concept term is more than the corresponding threshold. Different concept terms and thresholds of each CP can guarantee that each bug report belongs to a unique CP.

- **Severity prediction**: We consider each bug report in

Table 1: Severity Concepts for Eclipse Bug Reports

Severity Concepts	Threshold	Concept Terms
blocker	0.011	eclipse(0.0278), jdt(0.0241), compiler(0.0227), debug(0.0211), variable(0.0197), target(0.0188), data(0.0164), reject(0.0132), shape(0.0124), link(0.0113)
critical	0.021	formatter(0.0610), member(0.0312), character(0.0303), insert(0.0279), check(0.0257), illegal(0.0228), expression(0.0223), sort(0.0210)
major	0.011	compliance(0.0391), time(0.0311), clear(0.0309), setting(0.0275) file(0.0234), slow(0.0217), commend(0.0189), inline(0.0176) large(0.0156), match(0.0143), bug(0.0129), task(0.0115)
normal	0.011	remove(0.0283), resource(0.0247), one(0.0198), button(0.0151), strange(0.0135), folder(0.0135), bin(0.0128), directory(0.0125), find(0.0121), action(0.0119)
minor	0.010	name(0.0595), convert(0.0407), loop(0.0288), refactor(0.0234), clean(0.0219), enhance(0.0196), case(0.0175), drop(0.0143), save(0.0123), trail(0.0109)
trivial	0.012	declaration(0.0306), point(0.0287), state(0.0253), throw(0.0223), change(0.0211), add(0.0184), constant(0.0175), project(0.0143), view(0.0125)

the testing set as a query. We need to classify them to the corresponding category so that we can predict the severity of them. Similarity measure between the query and CPs helps us achieve the prediction task. We utilize KL-divergence [12] to compute the textual similarity. It measures a distance between the probability vectors of documents. The following definitions detail our approach.

Definition 1: Severity Prediction Algorithm

$$PredictS(q,c) = sim(\vec{\omega}_q, \vec{\omega}_c)$$

$$= -KL(p_{suni}(\omega \mid \vec{\omega}_q), p_{suni}(\omega \mid \vec{\omega}_c))$$

$$= -\sum_{i}^{|\omega_i|} p_{suni}(\omega_i \mid \vec{\omega}_q) \times log \frac{p_{suni}(\omega_i \mid \vec{\omega}_q)}{p_{suni}(\omega_i \mid \vec{\omega}_c)} \quad (1)$$

where

- $p_{suni}(\omega \mid \vec{\omega}_q)$ denotes the probability vector representation for query q.
- $p_{suni}(\omega \mid \vec{\omega}_c)$ stands for the probability vector representation for CP c.

In Definition 1, the query and the CP need to be transformed into the probability vector. Rao and Kak [13] have demonstrated that smoothed Unigram Model (UM) shows better performance than VSM and other statistical models (e.g., LDA) when IR-based algorithms are performed. Therefore, we adopt it to transformeach document to a $\mid V \mid$-dimensional probability vector. For the original UM, if a word has never occurred in a document, the probability of occurrence in the probability vector for that document is 0. To avoid this problem, the original UM needs to be smoothed by using the collection model that measures the probability of occurrence of a word in the whole collection of documents. Definition 2 details the process. After measuring the similarity between the query (new bug report) and the CP, we choose the CP with the highest similarity measure as the category the new bug report belongs to. Thus we can predict the corresponding severity of this new bug report.

Definition 2: Probability Vector Transformation for CP via Smoothed UM

$$P_{suni}(\omega \mid \vec{\omega}_c) = (1-\mu)\frac{\omega_c(n)}{\sum_{n=1}^{|T_c|}\omega_c(n)} + \mu\frac{\sum_{l=1}^{NC}\omega_l(n)}{\sum_{l=1(l\neq c)}^{NC}\sum_{n=1}^{|T_l|}\omega_l(n)} \quad (2)$$

where

- ω is a term and $\vec{\omega}_c$ is a weight vector of CP c.
- $|T_c|$ is a number of topic terms in CP c.
- $\omega_c(n)$ stands for the occurrence frequency of the n^{th} topic term in CP c.
- NC denotes the total number of CPs, in this study, it is set to 5.
- μ is the difference weight of two parts in this equation.

4. EXPERIMENT

4.1 Experiment Setup

In order to demonstrate whether the proposed approach can be effectively employed in the real situation, we collected the bug reports with the different status of severity from the bug repositories of Eclipse[1] and Mozilla Firefox[2], which are shown in Table 2. We selected about 10% of bug reports as the testing set, and chose other bug reports as the training set. The third and fourth column show the number of bug reports in the training set and testing set, respectively. The last column 'Period' indicates that the collected bug reports were submitted during the period.

[1]https://bugs.eclipse.org/bugs/
[2]https://bugzilla.mozilla.org/

Table 2: Data Set

Projects	#All	#Train	#Test	Period[1]
Eclipse	9,939	8,945	994	01/10/11-14/09/30
Mozilla	4,440	3,996	444	03/12/05-14/10/07

[1]The format of Period is year/month/date

4.2 Research Questions

In order to evaluate the effectiveness of our approach in above data sets, we need to give the answers of the following research questions.

Q1: Is the severity of each given bug successfully predicted in the proposed approach?

The ultimate goal of our work is to predict the severity of the given bug. To complete the purpose, we develop a tool to perform severity prediction. We need to demonstrate that our approach is feasible and effective by executing the experiments.

Q2: Does the proposed approach perform better than other severity prediction methods by comparing the performance of them? If so, why?

To answer this question, we need to compare the performance of our approach against those of other severity prediction methods. We select some representative studies that include KNN [4], Naive Bayes [7] and Naive Bayes Multinomial [8] as the baselines in our experiment. Moreover, we explain the possible reasons why the proposed approach performs better than others.

4.3 Evaluation Result

In order to demonstrate that the proposed approach can be utilized to effectively execute the severity prediction, we need to compare the performance of previous studies that include KNN, Naive Bayes, Naive Bayes Multinomial with our method. In this paper, we adopt Precision ($\frac{TP}{TP+FP}$), Recall ($\frac{TP}{TP+FN}$) and F measures ($2 \times \frac{Precision \times Recall}{Precision+Recall}$) [14] to conduct the performance evaluation, where TP indicates the true positive instances, which are the number of bug reports predicted correctly; FP represents the false positive instances, which are the number of bug reports predicted incorrectly; FN denotes the false negative instances, which are the number of actual bug reports having the corresponding severity level, but are not predicted as the correct severity using the prediction methods.

We execute our method on the bug reports in the testing set. In addition, we also implement KNN, Naive Bayes and Naive Bayes Multinomial using R language [15] on the same testing set. Table 3 and Table 4 show the evaluation results of four severity prediction approaches for each CP of Eclipse and Mozilla Firefox data set, respectively. We note that our approach reaches higher average F measures that are more than 80% for the both of Eclipse and Mozilla Firefox data sets. According to the result, we can answer the first research question **Q1** as follows:

A1: Our approach can predict the severity level of the given bugs effectively, especially on normal due to a large number of bug reports denoted as normal. The size of training set affects the experimental results. For example, a small number of the 'blocker' bug reports lead to a relatively lower F-measure.

In Table 3 and Table 4, it can be seen that our approach shows better performance due to higher Precision, Recall and F measures for each CP than others. The precision, recall and F measures of our method are the best for all severity categories among four approaches. Figure 2 and Figure 3 show the average Prediction, Recall and F measures for all severity concepts. The precision, recall and F measures are 81.24%, 89.02% and 84.86% respectively which are higher than KNN, Naive Bayes and Naive Bayes Multinomial for Eclipse data set. In the same way for Mozilla Firefox data set, the precision, recall and F measure are the

Table 3: Performance Evaluation for Different Severity Prediction Approaches in Eclipse

	Measures	CPs	KNN	NB[1]	NB-M[2]
blocker	Precision	63.64%	57.14%	62.5%	54.55%
	Recall	77.78	44.44%	55.56%	66.67%
	F Measure	70%	50%	58.83%	60%
critical	Precision	86.49%	72.73%	79.41%	81.08%
	Recall	91.43%	68.57%	77.14%	85.71%
	F Measure	88.89%	70.59%	78.26%	83.33%
major	Precision	87.13%	72.34%	84.27%	87.91%
	Recall	95.65%	73.91%	81.52%	86.96%
	F Measure	91.19%	73.12%	82.87%	87.43%
normal	Precision	97.39%	88.73%	86.01%	93.33%
	Recall	96.02%	78.34%	82.08%	91.69%
	F Measure	96.7%	83.21%	84%	92.5%
minor	Precision	77.78%	42.86%	45.45%	66.67%
	Recall	87.5%	37.5%	62.5%	75%
	F Measure	82.35%	40%	52.63%	70.59%
trivial	Precision	75%	37.5%	44.44%	83.33%
	Recall	85.71%	42.86%	57.14%	71.43%
	F Measure	80%	40%	50%	76.92%

[1]Naive Bayes
[2]Naive Bayes Multinomial

Table 4: Performance Evaluation for Different Severity Prediction Approaches in Mozilla

	Measures	CPs	KNN	NB[1]	NB-M[2]
blocker	Precision	78.95%	53.33%	55.56%	63.16%
	Recall	88.24%	47.06%	58.82%	70.59%
	F Measure	83.34%	50%	57.14%	66.67%
critical	Precision	84.62%	38.46%	53.85%	57.14%
	Recall	91.67%	41.67%	58.33%	66.67%
	F Measure	88%	40%	56%	61.54%
major	Precision	73.33%	42.86%	53.33%	58.85%
	Recall	84.62%	46.15%	61.54%	76.92%
	F Measure	78.57%	44.44%	57.14%	66.66%
normal	Precision	92.39%	69.5%	79.11%	84.7%
	Recall	95.14%	59.73%	76.76%	86.76%
	F Measure	93.74%	64.25%	77.92%	85.72%
minor	Precision	71.43%	55.56%	77.78%	61.54%
	Recall	90.91%	45.45%	63.64%	72.73%
	F Measure	80%	50%	70%	66.67%
trivial	Precision	90.48%	57.14%	73.68%	73.91%
	Recall	86.36%	54.55%	63.64%	77.27%
	F Measure	88.37%	55.81%	68.29%	75.55%

[1]Naive Bayes
[2]Naive Bayes Multinomial

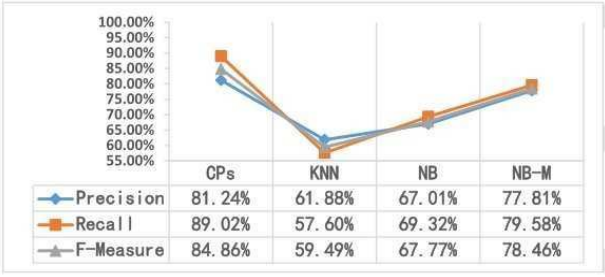

Figure 2: Performance Comparison in Eclipse

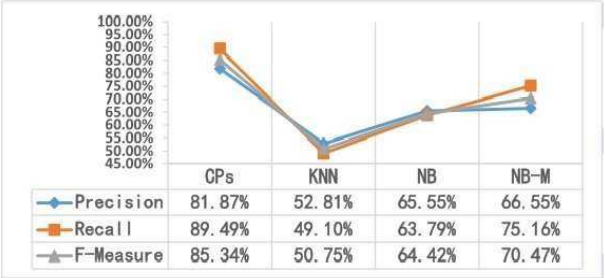

Figure 3: Performance Comparison in Mozilla

highest among four approaches.

To further demonstrate the conclusion that our approach performed better than KNN, Naive Bayes and Naive Bayes Multinomial, we use the statistical test methods that include t-test [16] and wilcoxon signed-rank test [17] for achieving this aim. We formulated the null hypotheses ($H1_0$, $H2_0$ and $H3_0$) and alternative hypotheses ($H1_a$, $H2_a$ and $H3_a$) as follows:

- $H1_0$, $H2_0$ and $H3_0$: The prediction performance of proposed method has no acceptability difference from KNN, Naive Bayes and Naive Bayes Multinomial, respectively.
- $H1_a$, $H2_a$ and $H3_a$: The prediction performance of proposed method is more acceptable than KNN, Naive Bayes and Naive Bayes Multinomial, respectively.

We input the overall F measures of all severity concepts as a feature for executing statistical test. This result is given in Table 5.

In Table 5, we note that the t-test is adopted for $H1_0$ and $H3_0$ in Eclipse data set due to the normal distribution of the data, and the wilcoxon signed-rank test is utilized for $H2_0$ since the data distribution is non-normal. The normality value can verify whether the data distribution is non-normal or normal. In detail, the data distribution is normal if the normality value is more than 0.05; otherwise, the data distribution is non-normal. The same is true for deciding to adopt the t-test or wilcoxon signed-rank test in Mozilla Firefox data set.

Whether by the t-test or wilcoxon signed-rank test, if the p-value is lower than 0.05, we reject the null hypothesis and accept the alternative hypothesis. For example, the p-value is 0.004143 that is lower than 0.05, we reject $H1_0$ and accept $H1_a$ in Eclipse data set. According to the p-values shown in Table 5, we reject all of null hypotheses $H1_0$, $H2_0$ and $H3_0$, and accept all of alternative hypotheses $H1_a$, $H2_a$ and $H3_a$ for both of Eclipse and Mozilla data sets. This result indicates that our approach has the significant difference with KNN, Naive Bayes and Naive Bayes Multinomial. By executing the performance comparison (Table 3-4 and Figure 2-3) and t-test (Table 5) among four prediction methods, we can draw a conclusion for providing an answer to the research question **Q2** as follows:

A2: our method for severity prediction using concept profile outperforms KNN, Naive Bayes and Naive Bayes Multinomial.

We summarize the possible reason for explaining why the proposed approach performed better than KNN, Naive Bayes and Naive Bayes Multinomial. Different from other approaches, our method extracts the concept terms from the historical bug reports in the training set. These concept terms can provide more accurate prediction than the classical supervised learning algorithms which rely on the features (e.g., the textual similarity between bug reports). Therefore, for the severity prediction on each CP, our approach shows better performance than KNN, Naive Bayes and Naive Bayes Multinomial.

5. THREATS TO VALIDITY

Even though our approach can predict the severity of given bug reports effectively, however there are some threats need to be verified as follows.

5.1 Generalization

We make the experiments on the bug reports which are extracted from one representative large-scale open source projects that include Eclipse and Mozilla Firefox, but we are not sure the proposed algorithm can also be used to commercial projects effectively. Because in business projects, the definition of the severity is different from open source projects, current algorithms may not be apply to commercial projects. In addition, we are not sure our approach can be used for predicting the severity of the defect which belongs to a new feature as well. This is because the existing bug repositories did not contain the possible known defects which are related to new feature defects.

5.2 Empirical analysis

Even if we explain that why the proposed algorithm performed better than KNN, Naive Bayes and Naive Bayes Multinomial, because this empirical analysis derives from our good understanding for the data distribution of the data set on each severity concept, this analysis may not be accurate. Moreover, the evaluation result may be impacted by the quality of bug reports and the life-cycle of projects. For example, our research objects are well-written bug reports. However in real time there are not all bug reports are complete and high-quality, some poor-written and unfinished bug reports may affect the prediction accuracy. We need to look for the strong evidence to support our analysis.

6. CONCLUSION

In this paper, we proposed a concept profile-based prediction approach for predicting the severity of the given bug reports. We first built the CPs by mining the historical bug reports of the bug repositories. Next, when a new bug report comes, by measuring the similarity between the bug

Table 5: Result of statistical test

Project	Null hypothesis	Normality value	Test type	p-value	Inverse hypothesis
Eclipse	$H1_0$	0.05518	t-test	0.004143	$H1_a$: Accept
	$H2_0$	0.0231	wilcoxon signed-rank test	0.03125	$H2_a$: Accept
	$H3_0$	0.1619	t-test	0.007458	$H3_a$: Accept
Mozilla Firefox	$H1_0$	0.01605	wilcoxon signed-rank test	0.03125	$H1_a$: Accept
	$H2_0$	0.9967	t-test	0.001159	$H2_a$: Accept
	$H3_0$	0.2553	t-test	0.002213	$H3_a$: Accept

report and the severity concepts in the CPs, we can decide which severity concept this new report belongs to so that we can predict the corresponding severity. The evaluation result showed the proposed algorithm performed better than KNN, Naive Bayes and Naive Bayes Multinomial due to the higher Precision, Recall and F measures.

In the future, we plan to implement our algorithm to commercial projects by extracting more data and corresponding features. For example, we can collect emails between triagers and fixers to analyze the text messages. Moreover, we will develop a new more effective machine learning algorithm or text classification technique to predict the severity of bug reports.

7. ACKNOWLEDGMENTS

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT& Future Planning(No.2014M3C4A 7030504), and by Seoul Creative Human Development Program funded by Seoul Metropolitan Government(No.HM120 006).

8. REFERENCES

[1] April, A., Hayes, J. H., Abran, A., and Dumke, R., "Software Maintenance Maturity Model(SMmm): the software maintenance process model," *Journal of Software Maintenance and Evolution: Research and Practice*, vol.17, pp.197-223, 2005.

[2] Jalbert, N. and Weimer, W., "Automated duplicate detection for bug tracking system," in *Proceedings of International Conference on Dependable System & Networks*, pp.52-61, 2008.

[3] Xuan, J., Jiang, H., Ren, Z., Yan, J. and Luo, Z., "Automatic bug triage using semi-supervised text classification," in *Proceedings of the 22th International Conference on Software Engineering and Knowledge Engineering*, pp.209-214, 2010.

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

[5] Zhang, T. and Lee, B., "An Automated Bug Triage Approach: A Concept Profile and Social Network Based Developer Recommendation," in *Proceedings of the 8th International Conference on Intelligent Computing Technology*, LNCS 7389, pp.505-512, 2012.

[6] Menzies, T., and Marcus, A., "Automated severity assessment of software defect reports," in *Proceedings of the 24th International Conference on Software Maintenance*, pp.346-355, 2008.

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

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

[9] Bhattacharya, P., Iliofotou, M., Neamtju, I. and Faloutsos, M., "Graph-based analysis and prediction for software evolution," in *Proceedings of the 34th International Conference on Software Engineering*, pp.419-429, 2012.

[10] Yang, G., Zhang, T. and Lee, B, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proceedings of the 38th Annual International Computer Software and Applications Conference*, pp.97-106, 2014.

[11] Friedman, C., Kra, P., Yu. H., Krauthammer, M. and Rzhetsky, A., "GENIES: A natural-language processing system for the extraction of molecular pathways from journal articles," Bioinformatics, vol.17, pp.S74-S82, 2001.

[12] Zhang, T. and Lee, B., "A hybrid bug triage algorithm for developer recommendation," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp.1088-1094, 2013.

[13] Rao, S. and Kak, A., "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *Proceedings of the 8th IEEE Working Conference on Mining Software Repositories*, pp.43-51, 2011.

[14] Goutte, C. and Gaussier, E., "A probabilistic interpretation of precision, recall and f-Score, with implication for evaluation," *LNCS*, vol.3408, pp.345-359, 2005.

[15] R Core Team, "R: A Language and Environment for Statistical Computing," *R Foundation for Statistical Computing*, 2012.

[16] The T-Test, "Research Methods Knowledge Base," http://www.socialresearchmethods.net/kb/contents.php.

[17] Wilcoxon, F.,"Individual comparison by ranking methods," *Biometrics Bulletin*, vol.1, no.6, pp.80-83, 2012.