Applying Topic Modeling and Similarity for Predicting Bug Severity in Cross Projects

Geunseok Yang[1], Kyeongsic Min[1], Jung-Won Lee[2], Byungjeong Lee[1*]
[1]Department of Computer Science, University of Seoul
Seoul, 02504 - South Korea
[e-mail: {ypats87, ksmin1710, bjlee}@uos.ac.kr]
[2]Department of Electrical and Computer Engineering, Ajou University
Suwon, 16499 - South Korea
[e-mail: jungwony@ajou.ac.kr]
*Corresponding author: Byungjeong Lee

Received October 9, 2018; revised January 23, 2019; accepted March 6, 2019; published March 31 2019

Abstract

Recently, software has increased in complexity and been applied in various industrial fields. As a result, the presence of software bugs cannot be avoided. Various bug severity prediction methodologies have been proposed, but their performance needs to be further improved. In this study, we propose a novel technique for bug severity prediction in cross projects such as Eclipse, Mozilla, WireShark, and Xamarin by using topic modeling and similarity (i.e., KL-divergence). First, we construct topic models from bug repositories in cross projects using Latent Dirichlet Allocation (LDA). Then, we find topics in each project that contain the most numerous similar bug reports by using a new bug report. Next, we extract the bug reports belonging to the selected topics and input them to a Naïve Bayes Multinomial (NBM) algorithm. Finally, we predict the bug severity in the new bug report. In order to evaluate the performance of our approach and to verify the difference between cross projects and single project, we compare it with the Naïve Bayes Multinomial approach; the Lamkanfi methodology, which is a well-known bug severity prediction approach; and an emotional similarity-based bug severity prediction approach. Our approach exhibits a better performance than the compared methods.

Keywords: Bug Severity Prediction, Cross Projects, Topic Modeling, KL-Divergence, Bug Report

A preliminary version of this paper was presented at APIC-IST 2018, and was selected by the conference review process. This work was supported by the 2018 Research Fund of the University of Seoul for Byungjeong Lee. Also, this work 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 (NRF-2014M3C4A7030504) for Jung-Won Lee.

1. Introduction

As the complexity of software has recently continued to increase, software faults inevitably occur. For example, in the Eclipse open source project approximately 300 bug reports are submitted per day [1]. A bug tracking system can be utilized to efficiently handle these numerous bug reports [2]. In general, software bug reports can be written by end users or developers. Subsequently, project managers read the bug reports manually and then assign them to developers, considering the severity of the bug reports. This severity level can provide a measure of whether the submitted faults need to be fixed quickly. In addition, the severity may be altered by an assigned developer in some cases. Therefore, owing to the different background knowledge of end users and developers the severity level can be subjective when bug reports are written [3].

The motivations of this study are as follows.

- In the bug severity selection phase, a subjective decision can occur because the background knowledge of end users and developers may be different [3]. If an automatic severity prediction tool is available, this can be resolved.

- Owing to different background knowledge between end users and developers, there may be unnecessary information in the bug report for a discovered fault, and there may a lack of necessary information (e.g., the stack trace or detailed scenario) [4]. Therefore, the severity of the bug report may not be accurate. The cost of the bug fixing can be reduced by recommending or predicting the severity using an automatic severity prediction tool.

For resolving these problems, in this study we propose an approach for predicting bug severity using topic modeling [5] and similarity (i.e., KL-divergence [6]) in cross projects, including Eclipse [7], Mozilla [8], WireShark [9], and Xamarin [10]. We first construct topic models from the bug repositories of the cross projects using Latent Dirichlet Allocation (LDA) [5]. Then, we extract the bug reports belonging to the selected topics and apply Naïve Bayes Multinomial [11] to these. Finally, we predict the severity of a new bug report.

To verify our model and to verify the difference between cross projects and single project, we compare it with the Naïve Bayes Multinomial approach; the Lamkanfi's work [11, 12], which is a well-known prediction method; and an emotional similarity-based severity method [13]. Then, we show that our approach outperforms the methods.

The original contributions of this study are as follows.

- We effectively predict the bug severity of the report using topic modeling and the KL-divergence in cross projects including Eclipse, Mozilla, WireShark, and Xamarin. We show that our model is effective for the bug severity prediction task.

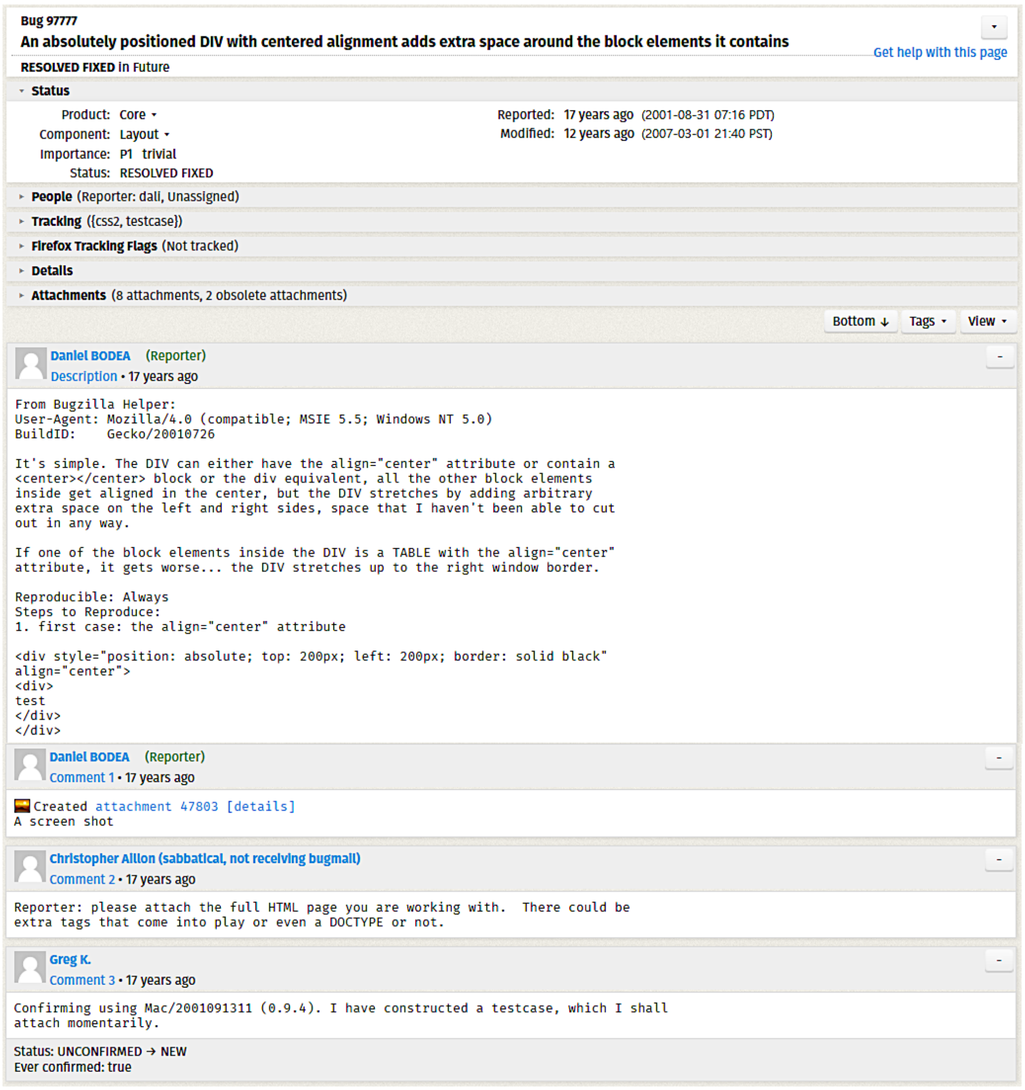
- We compare our approach with the Naïve Bayes Multinomial approach; the Lamkanfi methodology, which is a well-known severity prediction approach; and an emotional similarity-based bug severity approach. We show that our approach outperforms the compared methods as well as the difference between cross projects and single project.

The remaining sections of this paper are organized as follows. Section 2 presents some background knowledge. We present our methodology in Section 3. We describe the experimental results in Section 4. We discuss these results in Section 5. Then, we discuss related work in Section 6. Finally, we conclude the paper and suggest directions for future work in Section 7.

2. Background

2.1 Software Bug Reports

When end users or developers find a fault or functionality improvement, they may write some issues to submit to a bug repository. We present an example of a bug report [14] from Mozilla (#97777) in **Fig. 1**.



Bug 97777
An absolutely positioned DIV with centered alignment adds extra space around the block elements it contains

Get help with this page

RESOLVED FIXED in Future

Status

Product: Core
Component: Layout
Importance: P1 trivial
Status: RESOLVED FIXED

Reported: 17 years ago (2001-08-31 07:16 PDT)
Modified: 12 years ago (2007-03-01 21:40 PST)

People (Reporter: dall, Unassigned)
Tracking ([css2, testcase])
Firefox Tracking Flags (Not tracked)
Details
Attachments (8 attachments, 2 obsolete attachments)

Bottom ↓ Tags View

Daniel BODEA (Reporter)
Description • 17 years ago

```
From Bugzilla Helper:
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
BuildID:    Gecko/20010726

It's simple. The DIV can either have the align="center" attribute or contain a
<center></center> block or the div equivalent, all the other block elements
inside get aligned in the center, but the DIV stretches by adding arbitrary
extra space on the left and right sides, space that I haven't been able to cut
out in any way.

If one of the block elements inside the DIV is a TABLE with the align="center"
attribute, it gets worse... the DIV stretches up to the right window border.

Reproducible: Always
Steps to Reproduce:
1. first case: the align="center" attribute

<div style="position: absolute; top: 200px; left: 200px; border: solid black"
align="center">
<div>
test
</div>
</div>
```

Daniel BODEA (Reporter)
Comment 1 • 17 years ago

```
Created attachment 47803 [details]
A screen shot
```

Christopher Aillon (sabbatical, not receiving bugmail)
Comment 2 • 17 years ago

```
Reporter: please attach the full HTML page you are working with.  There could be
extra tags that come into play or even a DOCTYPE or not.
```

Greg K.
Comment 3 • 17 years ago

```
Confirming using Mac/2001091311 (0.9.4). I have constructed a testcase, which I shall
attach momentarily.
```

Status: UNCONFIRMED → NEW
Ever confirmed: true

Fig. 1. Example of a Bug Report (#97777 in Mozilla)

This bug report was first reported on *Aug. 31, 2001* and modified on *Mar. 1, 2007*. The title of the bug report is "*An absolutely positioned DIV with centered alignment adds extra space around the block elements it contains*," and it concerned the *Layout* component within the Mozilla *Core* product. The importance attributes shown in the figure are *P1* and *trivial*, where *P1* indicates the priority and *trivial* represents the severity of the bug report. In general (for Bugzilla), *P1* is the highest priority, which means emergency, and *P5* is the lowest priority. In this case, we focus on the severity importance attribute in the figure. In the Bugzilla case, the severity can be divided into seven levels [3] (i.e., Blocker, Critical, Major, Normal, Minor, Trivial, and Enhancement). *Enhancement* refers to an improvement of functionality issue, and so is excluded from this study. "*Daniel BODEA*" in the description refers to the reporter of the bug. From this description, developers can easily perform debugging if the reporter details the steps to reproduce the bug. However, in some cases these may not be provided or may be described in a freeform textual context. In addition, interested persons including reporters, developers, and users can freely post comments in the figure. Owing to the different background knowledge of end users and developers, the severity can be subjective when bug reports are written. In this study, we predict the severity ("*trivial* in Mozilla bug report # 97777) using the summary and description of the bug report.

2.2 Topic Modeling

We use LDA to build the topic models [5]. When the topic model is established, each topic is constructed using the word distribution and co-occurrence frequency. The LDA parameters can be divided into four types. Alpha describes the rate at which a document can be included in multiple topics. Beta is the rate at which many words can be included in classified topics. Gamma defines the number of times that the building of a topic model is repeated. Finally, N is the number of topics that will be constructed. In this paper, we extract the topics by using summary and description in the bug reports. The topics have word tokens related to the topic, and the word tokens have topic distribution scores. In details, we compare the word tokens between each topic and a new bug report. Then, we compute distribution score in each topic. Next, we find the similar topic related to a new bug report.

3. Methodology

We propose an approach for predicting bug severity using topic modeling [5] and similarity (i.e., the KL-divergence [6]) in cross projects including Eclipse [7], Mozilla [8], WireShark [9], and Xamarin [10], as shown in **Fig. 2**.

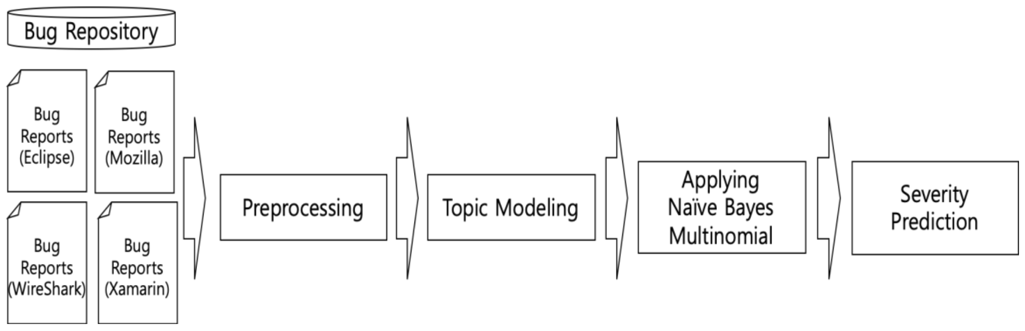


Fig. 2. Overview of Our Approach

First, we apply a preprocessing technique [1] to the bug repository. Next, we construct a topic model using LDA from a bug repository for the cross projects. Then, we find similar topics from the cross projects, and determine the topics that have the most numerous similar bug reports by using a new bug report in any project.

Next, we extract the bug reports belonging to the selected topics and input them into an NBM algorithm [11]. Finally, we predict severity of new bug report.

3.1 Preprocessing

In general, a bug report can be freely described (i.e., freeform textual contexts), and so we employ a preprocessing technique [1] including tokenization, stop-word removal, and word stemming. We present an example of this preprocessing for the Mozilla bug report #97777 [14]. A summary of the bug report is given as "An absolutely positioned DIV with centered alignment adds extra space around the block elements it contains." Upon applying the preprocessing technique, we obtain word tokens such as "position DIV center alignment add extra space around block element.".

3.2 Similar Topic

In this paper, we use predefined fields (e.g., the summary and description) of a bug report to construct a topic model. The built-in topics are illustrated in Fig. 3.

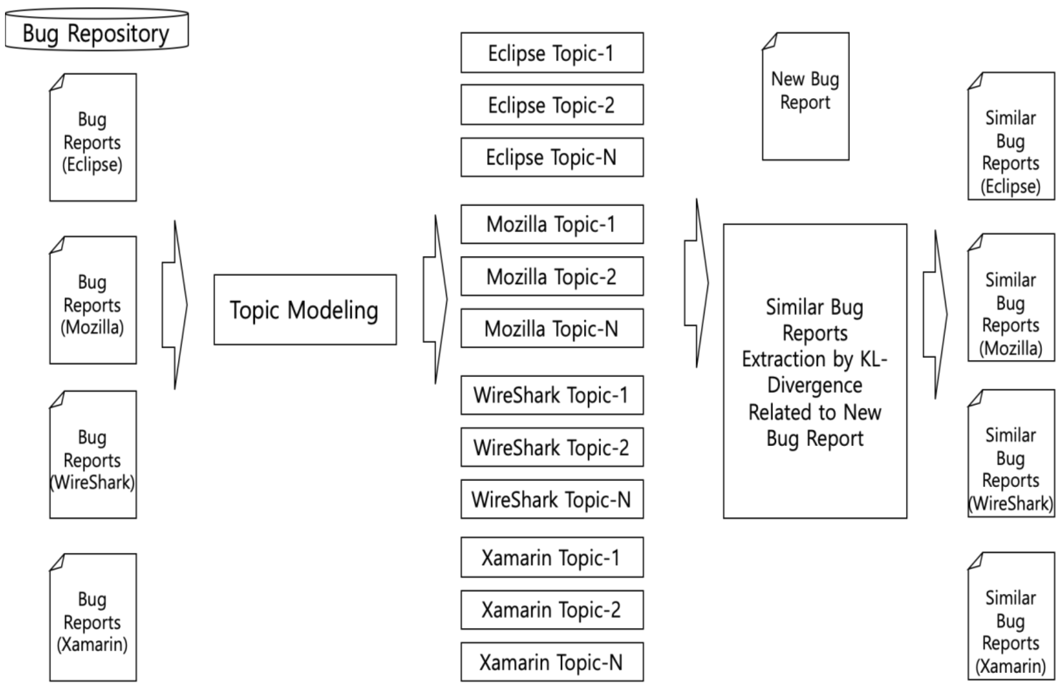


Fig. 3. Example of a Construction Topic

To find a similar topic, we first use a smoothed unigram model [6] for converting bug reports into probability vectors. The formula of the smoothed unigram model is given in **Formula 1**.

$$P_{brt}(wv|\overrightarrow{wv_k}) = (1-\mu)\frac{wv_k(n)}{\sum_{n=1}^{|R_k|} wv_k(n)} + \mu\frac{\sum_{l=1}^{K} wv_l(n)}{\sum_{l=1(l\neq k)}^{K}\sum_{n=1}^{|R_l|} wv_l(n)} \quad (1)$$

- wv is a term and $\overrightarrow{wv_k}$ is a weight vector for report k.
- $|R_k|$ is the number of words in report k.
- wv_k(n) is the occurrence frequency of the n^{th} vocabulary term in report k.
- K is the total number of reports, and μ defines the different weights of the two parts in this formula.

Then, using the KL-divergence [6], we compute the similarity between a new bug report and historical bug reports for the selected similar topic(s). The formula is given in **Formula 2**.

$$\text{SimilarityBRT}(\overrightarrow{wv_q},\overrightarrow{wv_k}) = -\text{KL}\left(P_{brt}(wv|\overrightarrow{wv_q}), P_{brt}(wv|\overrightarrow{wv_k})\right)$$
$$= -\sum_i^{i=|\omega_i|} P_{brt}(wv_i|\overrightarrow{wv_q}) * \log\frac{P_{brt}(wv_i|\overrightarrow{wv_q})}{P_{brt}(wv_i|\overrightarrow{wv_k})} \quad (2)$$

- $P_{brt}(wv|\overrightarrow{wv_q})$ is the probability of term wv appearing in query q.
- $P_{brt}(wv|\overrightarrow{wv_k})$ is the probability of term wv appearing in bug report k.

As a result, we find similar topics from cross projects including Eclipse, Mozilla, WireShark, and Xamarin that are related to a new bug report. In detail, we suppose that similar topic(s) contains the most numerous similar bug reports.

3.3 Applying Naïve Bayes Multinomial

In this paper, to predict the severity of a bug we find similar topics from cross projects including Eclipse, Mozilla, WireShark, and Xamarin that are related to a new bug report and extract bug reports from these similar topics. Then, we input these into the Naïve Bayes Multinomial algorithm.

Topic-Similar_BugReports =
$$SimBRt_{Eclipse} \cup SimBRt_{Mozilla} \cup SimBRt_{WireShark} \cup SimBRt_{Xamarin} \quad (3)$$

- *Topic-Similar_BugReports* are extracted and input into the Naïve Bayes Multinomial algorithm.

- $SimBRt_{Projects}$ represents similar bug reports to a new bug in a given project using the KL-divergence. For example, $SimBRt_{Eclipse}$ is extracted from an Eclipse topic related to a given new bug.

Finally, we can apply our model to predict the severity of a new bug.

4. Experiment

4.1 Dataset

In this study, we collected bug reports from the Eclipse [7], Mozilla [8], WireShark [9], and Xamarin [10] open source projects. In detail, we extracted the bug reports that had the status "Fixed and Resolved," and excluded those with the severity of "Enhancement." The dataset is summarized in **Table 1**.

Table 1. Summary of Our Dataset

Projects	Size	Period
Eclipse	**5,000**	**2002-01-01 ~ 2016-02-20**
Mozilla	**5,000**	**2002-09-27 ~ 2005-09-03**
WireShark	**5,000**	**2005-04-06 ~ 2017-07-19**
Xamarin	**5,000**	**2011-07-14 ~ 2014-11-19**

4.2 Evaluation Metrics

To evaluate the bug severity prediction performance, we use the Precision [15], Recall [15], and F-measure [15], as defined in **Formulas 4, 5, and 6**, respectively.

$$Precision\ (Severity) = \frac{SeverityTrueP}{SeverityTrueP + SeverityFalseP} \quad \textbf{(4)}$$

$$Recall\ (Severity) = \frac{SeverityTrueP}{SeverityTrueP + SeverityFalseN} \quad \textbf{(5)}$$

$$F - Measure\ (Severity) = 2 * \frac{Precision\ (Severity) * Recall\ (Severity)}{Precision (Severity) + Recall\ (Severity)} \quad \textbf{(6)}$$

- *Severity* is the severity of a bug report.

- *SeverityTrueP* is the number of correct predictions of bug severities, and *SeverityFalseP* is the number of incorrect predictions. *SeverityFalseN* is the number of bug reports for which the severity status was not correctly predicted using our approach.

In addition, we apply 10-fold cross validation [16] to reduce the bias of the data. That is, we divide the bug reports into nine training sets and one test set, and this test set is not included in the training set. For a fair comparison, the cross validation is also applied to our baselines. To compare our approach with the baselines, we conduct the experiments of the baselines as follows. First, we perform the experiments of the baselines in each project. Next, we compute the average values of the baselines in all projects. Finally, we compare our study (e.g., cross projects) with the baselines (e.g., average values of all projects). For evaluating the bug severity prediction performance, we considered the Naïve Bayes Multinomial approach [11], the Lamkanfi methodology [12], and an emotional similarity-based bug severity prediction [13] method as baselines.

■ Naïve Bayes Multinomial [11]: This represents the existence of a term and its number of occurrences in documents. Lamkanfi et al. showed that the Naïve Bayes Multinomial approach achieved the best performance for bug severity prediction.

■ Lamkanfi [12]: This approach classifies bug reports using the meta fields (Product and Component) of a bug report. Then, the Naïve Bayes algorithm is employed to train the model and predict the severity of a new bug.

■ Emotional similarity-based bug severity [13]: This approach calculates the emotion similarity based on an emotion dictionary. By considering bug reports with a similar emotion, the authors applied the Naïve Bayes Multinomial algorithm and predicted the severities of new bugs.

We address the following research questions in conducting the experiment:

■ RQ1. How accurate is the bug severity prediction performance for our approach?

This question is considerably important, and should be addressed before comparing with other baselines. In answering this question, we can evaluate our model.

■ RQ2. Can the proposed methodology be adopted for the bug severity prediction task?

We compare the effectiveness of our approach for predicting bug severity with those of other baselines as well as the difference between cross projects and single project. Then, we conduct a statistical test [17, 18, 19] to determine whether there is a difference between the proposed methodology and the baselines.

4.4 Results

■ RQ1. How accurate is the bug severity prediction performance using our approach?

The bug severity prediction performance of our approach is illustrated in **Fig. 4**. The x-axis represents the number of x-fold cross validations (1-fold, 2-fold, etc.), and the y-axis represents the average accuracy (e.g., all severity levels) in each cross validation. The average accuracy of our approach is 83.02% in terms of the F-Measure.

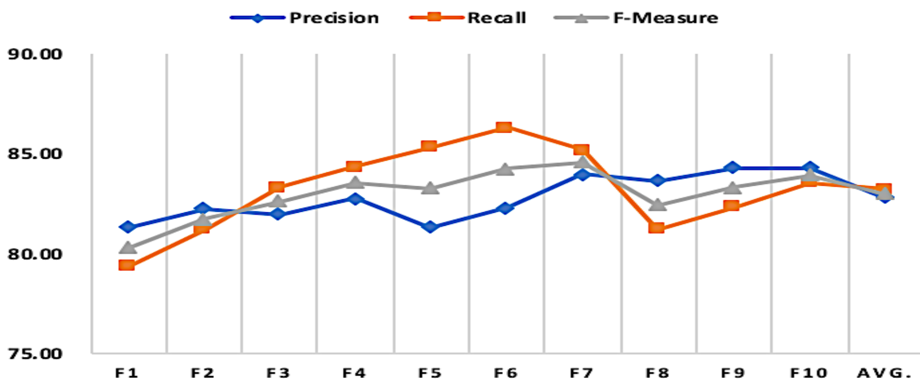


Fig. 4. Accuracy Results for Our Approach

■ RQ2. For the prediction of bug severity, can the proposed methodology be adopted?

The bug severity prediction performances of our approach and the baselines are illustrated in **Fig. 5**. The x-axis represents the baseline name (e.g., Naïve Bayes Multinomial, Lamkanfi, and ES-Multinomial), and the y-axis represents the average of the 10-fold cross validation (e.g., average values in all projects including Eclipse, Mozilla, WireShark, and Xamarin). Our approach exhibits a better performance than the baselines.

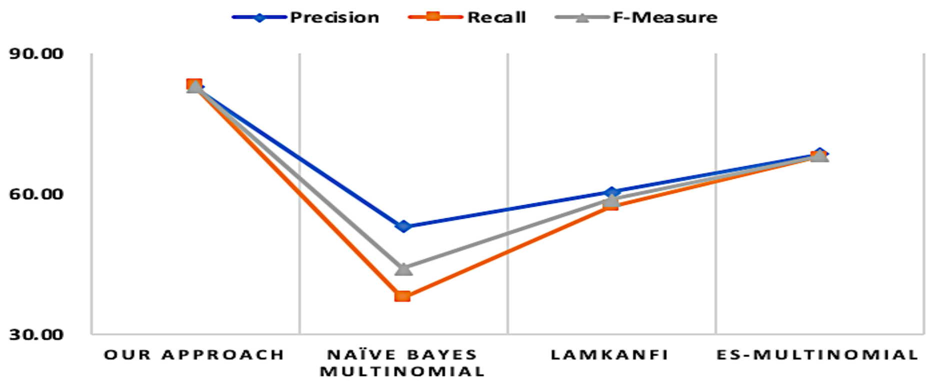


Fig. 5. Comparison of Results for Our Approach and the Baselines

In addition, we provide the results for our baselines (Naïve Bayes Multinomial, Lamkanfi, and ES-Multinomial) in details as shown in **Fig. 6**, **Fig. 7**, and **Fig. 8**, respectively. The x-axis represents the project name (e.g., Eclipse, Mozilla, WireShark, and Xamarin), and the y-axis represents the average of the 10-fold cross validation (for each project).

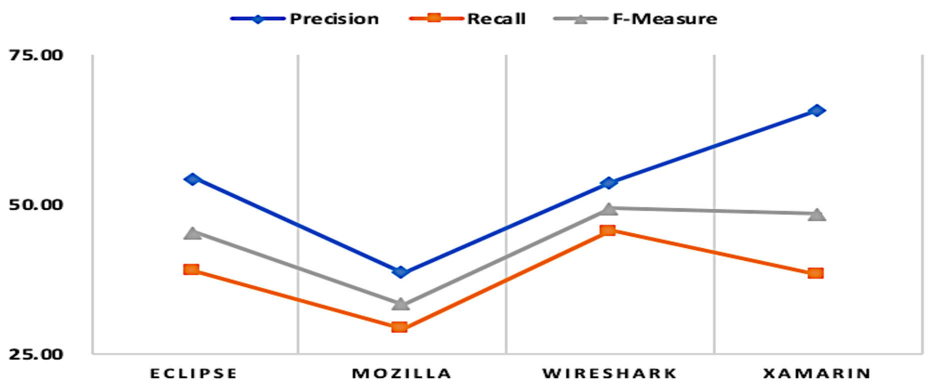


Fig. 6. Accuracy Results for Naïve Bayes Multinomial in Details

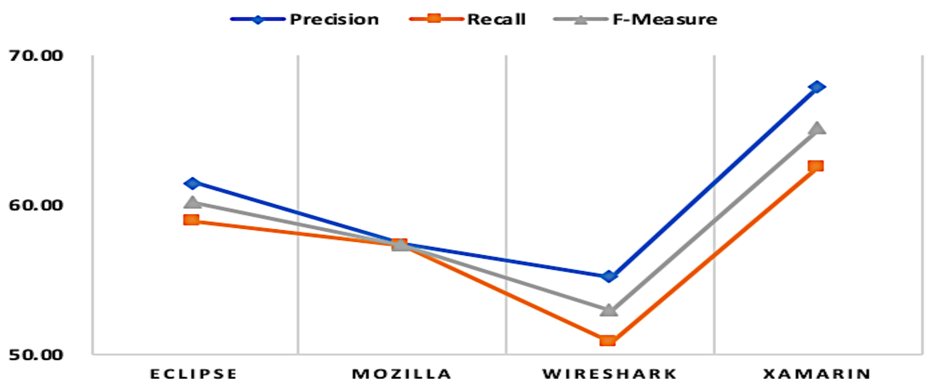


Fig. 7. Accuracy Results for Lamkanfi in Details

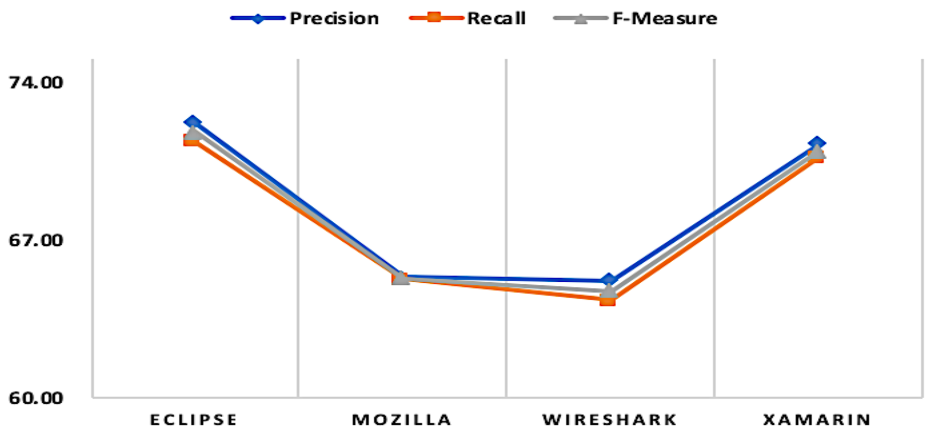


Fig. 8. Accuracy Results for ES-Multinomial in Details

Moreover, we conduct a statistical test [17, 18, 19] to verify whether there is a significant difference between our approach and the baselines. We establish the null hypothesis as follows.

■ $H1_0, H2_0, H3_0, H4_0$: There is no difference between this method and Naïve Bayes

Multinomial in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

- $H5_0$, $H6_0$, $H7_0$, $H8_0$: There is no difference between this method and Lamkanfi in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

- $H9_0$, $H10_0$, $H11_0$, $H12_0$: There is no difference between this method and emotional similarity-based bug severity in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

The alternative hypothesis is established as follows.

- $H1_a$, $H2_a$, $H3_a$, $H4_a$: There is a difference between this method and Naïve Bayes Multinomial in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

- $H5_a$, $H6_a$, $H7_a$, $H8_a$: There is a difference between this method and Lamkanfi in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

- $H9_a$, $H10_a$, $H11_a$, $H12_a$: There is a difference between this method and emotional similarity-based bug severity in Eclipse, Mozilla, WireShark, and Xamarin, respectively.

First, we conduct a statistical test on the average F-scores of the cross validations on our approach and the baselines. We verify the normality using the Shapiro–Wilk test [19] for each null hypothesis. If the normality is greater than or equal to 0.05, we run the T-test [17]. Otherwise, we run with the Wilcoxon signed-rank test [18]. The results of the statistical tests are presented in **Table 2**.

For example, the score of the null hypothesis $H1_0$ is 5.483e-15 (0.000000000000005483), and because this is less than 0.05, we can conclude that our approach exhibits a difference from Naïve Bayes Multinomial, and we accept the alternative hypothesis $H1_a$. We note that all the alternative hypotheses are accepted.

Table 2. Results of Statistical Tests

Null Hypothesis	P-Value	Alternative Hypothesis
$H1_0$	5.483e-15	$H1_a$: Accept
$H2_0$	2.2e-16	$H2_a$: Accept
$H3_0$	3.811e-14	$H3_a$: Accept
$H4_0$	2.695e-14	$H4_a$: Accept
$H5_0$	6.141e-12	$H5_a$: Accept
$H6_0$	7.256e-14	$H6_a$: Accept
$H7_0$	8.466e-14	$H7_a$: Accept
$H8_0$	2.675e-11	$H8_a$: Accept
$H9_0$	2.185e-09	$H9_a$: Accept
$H10_0$	1.063e-10	$H10_a$: Accept
$H11_0$	0.003906	$H11_a$: Accept
$H12_0$	6.447e-10	$H12_a$: Accept

5. Discussion

5.1 Experiment Analysis

- Performance of our approach: we predict the severity of a bug using topic modeling and a similarity (i.e., the KL-divergence) in cross projects including Eclipse, Mozilla, WireShark, and Xamarin. We achieve a better performance in terms of the precision, recall, and F-measure compared with our baselines, and exhibits a statistically significant difference. In addition, we verify the difference between cross projects and single project (from **Fig. 5**). We combine the projects (e.g., Eclipse, Mozilla, WireShark, and Xamarin) and then, we can achieve a better performance. In details, first, we construct the topic models from the cross projects. Then, we compare the words between each topic model in cross projects and a new bug report. Next, we not only find the similar topics related to the new given bug report, but also extract a large number of multiple word tokens from the similar topics by using KL-Divergence. Then, we utilize the Naïve Bayes Multinomial concept, considering the existence of a term and its number of occurrences in documents, to improve the performance of original Naïve Bayes Multinomial. In **Fig. 5**, we show that our approach has a better performance than the traditional algorithms (Naïve Bayes Multinomial and Naïve Bayes). Also, we have a higher performance than ES-Multinomial (Emotion Similarity-based Naïve Bayes Multinomial). In the future, we plan to improve the prediction performance using various attributes of bug reports.

5.2 Threats to Validity

- Internal threat: In this study, we set alpha to 0.01, beta to 0.01, gamma to 1500, and N to 30 in the topic modeling process. In addition, we set a parameter of the smoothed unigram model to 0.7 and a threshold to 0 in the KL-divergence. However, we cannot conclude that these settings are appropriate for all other projects. In the future, we will investigate appropriate parameters considering various projects.

- External threat: This study used Eclipse, Mozilla, WireShark, and Xamarin datasets for predicting bug severity. However, our approach may not be suitable for all other projects, such as business projects, because these may be different from our datasets.

6. Related Work

Many researchers have investigated approaches to predict the severities of software bugs. We present a qualitative comparison of bug severity prediction approaches in **Table 3**.

Table 3. Qualitative Comparison of Related Approaches

Study	Methodology	Project Type
Yang [3, 13]	Emotion Similarity, Emotion Score	Single Project
Zhang [20]	LDA, BM25, KNN	
Yang [1]	LDA, Meta Fields, KL-Divergence	

Tian [21]	BM25F, KNN	
Yang [22]	Feature Selection	
Lamkanfi [11, 12]	Traditional Machine Learning Algorithm, Meta Fields	
Our Approach	LDA, KL-Divergence	Cross Projects

Yang et al. (2018 & 2017) [3, 13] proposed an approach for predicting bug severity using emotion words in a bug report. In the first study, an emotion score for a bug report was computed using an emotion dictionary, and Naïve Bayes Multinomial was applied. In the further study, emotion dictionary-based emotional similarity was executed and Naïve Bayes Multinomial was applied using similar emotion bug reports.

Zhang et al. (2016) [20] used the LDA, BM25, and K-nearest neighbors (KNN) algorithms to predict bug severity. First, similar bug reports are obtained using a topic model and the meta fields of bug reports, and then the KNN algorithm is applied. In addition, the authors applied a tuning process to determine an appropriate KNN parameter.

Yang et al. (2014) [1] proposed a bug severity prediction approach by using LDA, predefined meta-fields, the KL-divergence, and the KNN algorithm. In details, the authors constructed a topic model using LDA, compared the words in a new bug report with those in the topics, and assumed that the most often matched topic terms correspond to a similar topic. They applied the meta fields to extract bug reports, and finally predicted the bug severity using the KNN algorithm.

Tian et al. (2012) [21] noted the problem of the computation time for the existing BM25 approach, and proposed a new methodology called BM25F. In order to improve the bug severity prediction performance, they determined an appropriate K value for KNN.

Yang et al. (2012) [22] proposed the bug severity prediction using feature selection. In detail, the information gain (IG) and correlation coefficient (CC) were used to compute a score, and Naïve Bayes Multinomial was applied.

Lamkanfi et al. (2011 & 2010) [11, 12] used the meta fields of bug reports to predict bug severity. In the latter study, they showed that Naïve Bayes Multinomial achieved a better performance than Naïve Bayes, KNN, and Support Vector Machines (SVM) in predicting bug severity.

7. Conclusion

In this study, we proposed a novel technique for bug severity prediction in cross projects. In detail, we first construct topic models by using LDA from bug repositories in cross projects including Eclipse, Mozilla, WireShark, and Xamarin. Then, we find topics in each project that contain the most numerous similar bug reports by using a new bug report. Next, we extract the bug reports belonging to the selected topics, and input them into the Naïve Bayes Multinomial algorithm. Finally, we predict the severity of a new bug report. For evaluating the performance of our model and to verify the difference between cross projects and single project, we compared it with Naïve Bayes Multinomial; the Lamkanfi methodology, which is a well-known bug severity prediction approach; and an emotional similarity-based bug severity prediction approach. We showed that our model achieved a better performance than the compared methods as well as verified the difference between cross projects and single project. In addition, we conducted a statistical test with null hypotheses and accepted all alternative hypotheses. From our motivation, a subjective decision of bug severity selection can occur because of the different background knowledge between end users and developers. In the future, we will use additional bug report attributes to improve the bug severity prediction performance and to make an automatic bug severity prediction tool for helping the reporters.

References

[1] 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 *Proc. of Computer Software and Applications Conference*, 2014. Article (CrossRef Link).

[2] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu, "Improving Bug Tracking Systems," in *Proc. of ICSE Companion*, pp. 247-250, 2009. Article (CrossRef Link).

[3] G. Yang, S. Baek, J. W. Lee, and B. Lee, "Analyzing Emotion Words to Predict Severity of Software Bugs: A Case Study of Open Source Projects," in *Proc. of ACM Symposium on Applied Computing*, 2017. Article (CrossRef Link).

[4] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What Makes a Good Bug Report?," *IEEE Transactions on Software Engineering*, Vol. 36, No. 5, pp. 618-643, 2010. Article (CrossRef Link).

[5] David M. Blei, Andrew Y. Ng and Michael I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, Vol. 3, pp. 993-1022, 2003. Article (CrossRef Link).

[6] S. Rao and A. Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models," in P*roc. of Working Conference on Mining Software Repositories*, pp. 43-52, 2011. Article (CrossRef Link).

[7] Eclipse, "https://bugs.eclipse.org/bugs," Retrieved March 10, 2019. Article (CrossRef Link).

[8] Mozilla, "https://bugzilla.mozilla.org," Retrieved March 10, 2019. Article (CrossRef Link).

[9] WireShark, "https://bugs.wireshark.org/bugzilla," Retrieved March 10, 2019. Article (CrossRef Link).

[10] Xamarin, "https://bugzilla.xamarin.com," Retrieved March 10, 2019. Article (CrossRef Link).

[11] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug," in *Proc. of Software Maintenance and Reengineering*, pp. 249-258, 2011. Article (CrossRef Link).

[12] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the Severity of a Reported Bug," in *Proc. of 7th IEEE Working Conference on Mining Software Repositories*, pp. 1-10, 2010. Article (CrossRef Link).

[13] G. Yang, T. Zhang, and B. Lee, "An Emotion Similarity Based Severity Prediction of Software Bugs: A Case Study of Open Source Projects," *IEICE Transactions on Information and Systems*, Vol. E101.D, Issue. 8, pp. 2015-2026, 2018. Article (CrossRef Link).

[14] Mozilla Bug Report #97777, https://bugzilla.mozilla.org/show_bug.cgi?id=97777, Retrieved March 10, 2019. Article (CrossRef Link).

[15] C. Goutte, and E. Gaussier, "A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation," in *Proc. of European Conference on Information Retrieval*, pp. 345-359, 2005. Article (CrossRef Link).

[16] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *Proc. of the 14th international joint conference on Artificial intelligence*, Vol. 14, No. 2, pp. 1137-1145, 1995. Article (CrossRef Link).

[17] The T-Test, "Research Methods Knowledge Base," "http://www.socialresearchmethods.net/kb/stat_t.php," Article (CrossRef Link).

[18] Wilcoxon, F., "Individual Comparisons by Ranking Methods," *Biometrics bulletin*, Vol. 1, No. 6, pp. 80-83, 1945. Article (CrossRef Link).
[19] Shapiro, S. S., and Wilk, M. B., "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika*, Vol. 52, No. 3/4, pp. 591-611, 1965. Article (CrossRef Link).
[20] 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. Article (CrossRef Link).
[21] Y. Tian, D. Lo, and C. Sun, "Information Retrieval based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction," in *Proc. of 19th Working Conference on Reverse Engineering*, pp. 215-224, 2012. Article (CrossRef Link).
[22] C. Z. Yang, C. C. Hou, W. C. Kao, and X. Chen, "An Empirical Study on Improving Severity Prediction of Defect Reports using Feature Selection," in *Proc. of 19th Asia-Pacific Software Engineering Conference*, pp. 240-249, 2012. Article (CrossRef Link).

Geunseok Yang, He received the B.E. degree in Department of Computer Science and Engineering from KOREATECH in 2013 and M.E. degree in Department of Computer Science from University of Seoul in 2015. Currently, he is a Ph.D. student at the department of Computer Science, University of Seoul. His research interest includes Software Evolution, Machine Learning, and Natural Language Processing.

Kyeongsic Min, He received his B.S degree from University of Seoul, Korea, in 2018. He is currently studying toward the M.S degree in Computer Science at University of Seoul, Korea. His research areas of interest include software engineering and blockchain technology.

Jung-Won Lee, is an associate professor of the Department of Electrical and Computer Engineering at Ajou University, Korea. She received her PhD. Degree in Computer Science and Engineering from Ewha Womans University, Korea, in 2003. She was a researcher of LG Electronics and did an internship in the IBM Almaden Research Center, USA. Her areas of research include context-aware, embedded software and software engineering.

Byungjeong Lee, He received the B.S., M.S., and Ph.D. degrees in Computer Science from Seoul National University in 1990, 1998, and 2002, respectively. He was a researcher of Hyundai Electronics, Corp. from 1990 to 1998. Currently, he is a professor of the Department of Computer Science and Engineering at the University of Seoul, Korea. His research areas include software engineering and web science.