An Emotion Similarity Based Severity Prediction of Software Bugs: A Case Study of Open Source Projects

Geunseok YANG, Member, Tao ZHANG, and Byungjeong LEE, Nonmembers

SUMMARY Many software development teams usually tend to focus on maintenance activities in general. Recently, many studies on bug severity prediction have been proposed to help a bug reporter determine severity. But they do not consider the reporter’s expression of emotion appearing in the bug report when they predict the bug severity level. In this paper, we propose a novel approach to severity prediction for reported bugs by using emotion similarity. First, we do not only compute an emotion-word probability vector by using smoothed unigram model (UM), but we also use the new bug report to find similar-emotion bug reports with Kullback–Leibler divergence (KL-divergence). Then, we introduce a new algorithm, Emotion Similarity (ES)-Multinomial, which modifies the original Naive Bayes Multinomial algorithm. We train the model with emotion bug reports by using ES-Multinomial. Finally, we can predict the bug severity level in the new bug report. To compare the performance in bug severity prediction, we select related studies including Emotion Words-based Dictionary (EWD)-Multinomial, Naïve Bayes Multinomial, and another study as baseline approaches in open source projects (e.g., Eclipse, GNU, JBoss, Mozilla, and WireShark). The results show that our approach outperforms the baselines, and can reflect reporters’ emotional expressions during the bug reporting.

key words: bug severity prediction, emotion similarity, bug report, software maintenance

1. Introduction

Recently, as various software products have been released [1], many developers on a software development team, in general, try to finalize the software during the process. Since the source code and functionality are complex, they cannot avoid software bugs, major and minor [2], and they usually spend more time debugging than coding [3], [4].

For the Eclipse open source project [5], about 300 bug reports have been submitted to the software repository each day. Most software development teams adopt a bug tracking system [6] (e.g., Bugzilla) to manage the bug history. When a new bug is found, end users and developers can write a bug report and submit it to the bug repository. However, they select the bug severity using their own background knowledge. There is a difference between developers’ and common users’ background knowledge when they attempt to select an appropriate bug severity level [7]. In addition, bug severity can affect subjective decisions of reporters [7].

Our motivations are as follows.

• Bug report severity may have different meanings depending on who are reporters (e.g., developers and users). There is mismatch information between developers (who are) looking for bug fixing information, e.g., stack trace, etc., and users (who are) giving some information in their background knowledges [8]. The report also can affect subjective decisions of reporters. Thus, automatic system of bug severity prediction needs to resolve this problem.

• If bug severity is assigned incorrectly, developers cannot fix urgent bugs, and thus, the software cannot reflect the customers’ requirements.

• With automatic bug severity prediction systems, developers can reduce the time to manage their plans. The system also can lead reporters in to select the appropriate bug severity during the reporting process.

• The bug reports can be affected by the reporters’ emotional expressions. Since a bug report is not pre-formatted with bug descriptions, reporters can write their own bug descriptions with a freeform textual context.

To resolve these problems, we propose a novel bug severity prediction approach by analyzing emotion similarity. First, we construct an emotion words–based dictionary [9] and reduce emotion words to their stems through preprocessing. Next, we find similar-emotion bug reports by using smoothed unigram model (UM)-based Kullback–Leibler divergence (KL-divergence) [10] based on the new bug report. We note that an emotion bug report means they can compute an emotion score by using the terms of the emotion. Then, we apply the emotion bug reports to a Naive Bayes Multinomial algorithm that we adapt and call Emotion Similarity Multinomial (ES-Multinomial). Finally, we predict the bug severity level by using ES-Multinomial.

For a fair comparison, we select not only a Lamkanfi et al. study [11] and Naive Bayes Multinomial [7], but also Emotion Words-based Dictionary (EWD)-Multinomial [7], which is based on emotion analysis as our baseline. These results show that our ES-Multinomial outperforms others in open source projects, including Eclipse [12], GNU [13], JBoss [14], Mozilla [15], and WireShark [16].

Our contributions include the following.

• We first try to predict the bug severity by using emotion similarity.
similarity from mining the software repository. By analyzing the emotions between reporters and bug reports, we can effectively predict bug severity.

- According to ES-Multinomial results, we show more effective bug severity prediction than our baselines (EWD-Multinomial, Naïve Bayes Multinomial, and Lamkanfi et al.) in open source projects, including Eclipse, GNU, JBoss, Mozilla, and WireShark.

This paper is structured as follows. In Sect. 2, we introduce background knowledge on bug severity prediction. We propose our ES-Multinomial algorithm in Sect. 3, and then we show the results of experiments in bug severity prediction against our baselines in Sect. 4. We discuss the experimental results in Sect. 5, and we present related works in bug severity prediction in Sect. 6. We conclude this paper and suggest future work in Sect. 7.

2. Background

2.1 Bug Management

Software development teams usually adopt a bug-tracking system like Bugzilla to manage bug reports in general [6]. Bugzilla was created for the Mozilla project and supports various mother languages, cross-projects, and public licenses [17]. We introduce the Bugzilla bug management workflow [17] in Fig. 1, and others are very much alike.

First, users and developers find the bugs and write reports with descriptions. Then, they submit the bug reports to a repository and label them NEW. Triagers will assign the bug report to developers according to the descriptions, changing the status to ASSIGNED. Next, developers try to fix the bugs (which will then be RESOLVED), and quality assurance people will verify whether the bugs are fixed or not. The subsequent labels are VERIFIED and CLOSED if bugs are fixed correctly. If not, the bugs are reassigned to the same, or other, developers by the triagers. In this case, the status is changed to REOPENED.

In terms of bug reports, one of the main tasks of developers is to write freeform textual-based bug reports when they find bugs. We provide an example of a Mozilla bug report in Fig. 2. This report is related to ‘thread-safe’ in a java loader component. The reporter (Jon Smirl) wrote the bug report on October 15, 2000. The developer (Igor Kushnirskiy) fixed the bug April 30, 2001. The priority and severity are P3 and normal (labeled Importance in the bug report), respectively. In this paper, we are going to predict this severity level by using our model. The report was recorded with a bug description by the reporter and was discussed by commenters. This report has two comments.

2.2 Bug Severity

Bug severity is a key to decide which bugs should be fixed first [2], [10]. When reporters submit a bug report, they can select the severity level based on their background knowl-

![Fig. 1 Bugzilla bug management workflow](image)

Fig. 1 Bugzilla bug management workflow

![Bug 56738 bcjavacomponentloader not thread-safe](image)

RESOLVED FIXED

<table>
<thead>
<tr>
<th>Status</th>
<th>Product: Core Graveyard</th>
<th>Component: java to NPOM Bridge</th>
<th>Importance: P3 normal</th>
<th>Status: RESOLVED FIXED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported: 17 years ago</td>
<td>Modified: 3 years ago</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>People</th>
<th>Reporter: Jon Smirl</th>
<th>Assignee: Igor Kushnirskiy</th>
</tr>
</thead>
<tbody>
<tr>
<td>QA Contact</td>
<td>rajendra.pallath</td>
<td>Triage Owner: ---</td>
</tr>
<tr>
<td>CC</td>
<td>Nobody</td>
<td></td>
</tr>
</tbody>
</table>

Jon Smirl [Reporter]

Description • 17 years ago

Ignore the other failures about bcjavastubs.dll I caused them.

--bcJavaComponentLoader::bcJavaComponentLoader
--bcJavaComponentLoader::Init
--bcJavaComponentLoader::GetFactory

Asa Dotzler [asa]

Comment • 17 years ago

setting Jon Smirl’s bugs to New

| Status: UNCONFIRMED → NEW |
| Ever confirmed: true |

Igor Kushnirskiy [Assignee]

Igor Kushnirskiy [Assignee]

Comment • 17 years ago

bcJavaComponentLoader is thread-safe. the only thing to change is to

![Fig. 2 An example of mozilla bug report #56738](image)

†https://bugzilla.mozilla.org/show_bug.cgi?id=56738
Table 1 Comparison bug severity among 5 projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Field</th>
<th>Expression(Ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBoss</td>
<td>Priority</td>
<td>Blocker (6.57%)  Critical (9.26%) Major (74.24%) Minor (9.33%) Trivial (0.59%)</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Normal (78.31%)</td>
<td>Blocker (1.79%)  Critical (3.65%) Major (10.36%)</td>
</tr>
<tr>
<td>GNU</td>
<td>Normal (85.83%)</td>
<td>Critical (13.36%) Minor (0.80%) Trivial (0.62%)</td>
</tr>
<tr>
<td>Mozilla</td>
<td>Normal (65.57%)</td>
<td>Blocker (2.57%)  Critical (8.32%) Major (10.14%)</td>
</tr>
<tr>
<td>WireShark</td>
<td>Major (35.08%)</td>
<td>Blocker (1.14%)  Critical (10.03%)</td>
</tr>
</tbody>
</table>

2.3 Emotion Dictionary

Recently, machine emotional intelligence has become a challenge for researchers in artificial intelligence and other fields [19], [20]. A previous study [7] is related to predicting bug severity by using emotion analysis of a bug report. We also adopt an emotion words–based dictionary [9]. The dictionary consists of positive, negative, and non-emotion words. Each word has an emotion score if it is an emotion word. If not, the words do not have such a score. We introduce examples from the emotion dictionary in Table 2.

We note that the term ‘tiny’ is not an emotion word. However, ‘good’ is a positive term—e.g., positive score > negative score. We can analyze the reporters’ emotion in the bug report (Summary and Description) by using positive and negative scores from the emotion dictionary.

2.4 Emotional Bug Term on Severity

In this Section, we present the emotional terms of bug report depending on bug severity including ‘Blocker’, ‘Critical’, ‘Major’, ‘Normal’, ‘Minor’, and ‘Trivial’ in our Eclipse dataset as shown in Table 3.

Our model reproduces the results (emotional similarity between a given new bug report and historical bug reports) when we set the parameter of smoothed UM to 0.0 in Eclipse. Then, we extract the emotional terms to analyze the results for presenting as an example.

We note that the terms (e.g., break, wrong, incorrect, bad, dirty, and wrong) have the highest negative scores. In details, the terms (e.g., break, crash, and error) can indicate an urgent level. In terms of trivial level, the terms (e.g., minor, incorrect, and warn) can suggest a trivial level.

In addition, researchers [21] investigated whether the bug reports can be expressed by human emotion or not.
They verified the developers could express emotions on the reports. According to the results, the emotion expression can reflect in design selection, software maintenance activity, or their partners. Another study [22] focused on whether the developers’ emotion affects the colleagues and software artefacts, and they investigated these influences can result in software development. Other study [23] found the relationship between the commit message size and the expressed emotions on the message. In details, they can classify the developers who express more emotion terms by using positive/negative emotions in bug fixing activity. In this paper, we would like to predict the bug severity level by using these emotion terms.

### Table 3  An example of emotional term in bug report

<table>
<thead>
<tr>
<th>Blocker (Positive/Negative Score)</th>
<th>Critical (Positive/Negative Score)</th>
<th>Major (Positive/Negative Score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>break (2.5 / 5.875)</td>
<td>wrong (0.75 / 8.125)</td>
<td>incorrect (0.25 / 3)</td>
</tr>
<tr>
<td>crash (0.25 / 2)</td>
<td>break (2.5 / 5.875)</td>
<td>error (0.5 / 1.625)</td>
</tr>
<tr>
<td>error (0.5 / 1.625)</td>
<td>unconventional (0.25 / 2.125)</td>
<td>ignore (0.375 / 1.25)</td>
</tr>
<tr>
<td>lock (0.5 / 0.625)</td>
<td>inconsistent (0.375 / 1.75)</td>
<td>inaccessible (0.5 / 0.75)</td>
</tr>
<tr>
<td>-</td>
<td>error (0.5 / 1.625)</td>
<td>null (0.25 / 0.75)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normal (Positive/Negative Score)</th>
<th>Minor (Positive/Negative Score)</th>
<th>Trivial (Positive/Negative Score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad (0.875 / 10.625)</td>
<td>dirty (1 / 6.375)</td>
<td>wrong (0.75 / 8.125)</td>
</tr>
<tr>
<td>confuse (0.75 / 4.625)</td>
<td>unused (0.625 / 1.5)</td>
<td>minor (0.958 / 3.292)</td>
</tr>
<tr>
<td>failure (0.625 / 2.125)</td>
<td>conflict (0.875 / 1.375)</td>
<td>incorrect (0.25 / 3)</td>
</tr>
<tr>
<td>disable (0.125 / 1.5)</td>
<td>smoke (0.125 / 0.375)</td>
<td>warn (0.375 / 1.125)</td>
</tr>
<tr>
<td>deprecate (0.25 / 1)</td>
<td>unexpected (0.125 / 0.375)</td>
<td>-</td>
</tr>
<tr>
<td>disappear (0.125 / 0.875)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### 3. Methodology

We introduce a novel approach to bug severity prediction by using emotion similarity, as shown in Fig. 3. First, we apply a preprocessing technique in all the bug reports. Next, we compute not only the probability vector of emotion terms by using smoothed UM [10], but we also determine emotion distribution between a new bug report and historical bug reports by using KL-divergence [10]. Then, we introduce Emotion Similarity Multinomial, which modifies the original Naïve Bayes Multinomial algorithm [7], [24]. Finally, when a new bug is submitted to our ES-Multinomial model, the severity can be predicted and recommended by our model.

#### 3.1 Preprocessing

Since bug reports consist of free-form textual contexts, we adopt a preprocessing technique, including stop word removal, special character removal, and word stemming. In this step, we utilize Stanford CoreNLP Tools (CoreNLP) [25]; we do not adjust any parameters, and keep the default parameters in CoreNLP. We apply the preprocessing technique to all the bug reports with the emotion word–based dictionary. We provide examples of bug reports in Table 4.

In the process, we get a word token from the #437094...
The parameter $\mu$ will be used to find the best severity prediction performance.

**Emotion-based KL-Divergence:** (which we call KL-divergence in this paper):

$$S_{EKLĐ}(\vec{ω}_q, \vec{ω}_k) = -KL\left(P_{ESUM}(ω|\vec{ο}_q), P_{ESUM}(ω|\vec{ο}_k)\right)$$

$$= -\sum_{i}^{(ω=|\vec{ω}|)} P_{ESUM}(ω|\vec{ο}_q) \log \frac{P_{ESUM}(ω|\vec{ο}_q)}{P_{ESUM}(ω|\vec{ο}_k)}$$

(2)

where

- $P_{ESUM}(ω|\vec{ο}_q)$ is the probability of emotion term $ω$ appearing in query $q$.
- $P_{ESUM}(ω|\vec{ο}_k)$ is the probability of emotion term $ω$ appearing in report $k$.

#### 3.3 ES-Multinomial

From an earlier study [24], Naïve Bayes Multinomial outperforms other algorithms, including Naïve Bayes, K-nearest neighbor (KNN) and support vector machines (SVM) in bug severity prediction. So, we adopt the Naïve Bayes Multinomial in this paper. We provide a comparison between Naïve Bayes [24] and Naïve Bayes Multinomial [7], [24] as follows.

- **Naïve Bayes** considers independence between predictors, in general. This algorithm computes the a posteriori probability of each feature in the groups, and classifies the feature into the group that has a much larger probability score than other groups.
- **Multinomial Naïve Bayes** considers a multinomial distribution in each feature. Lamkanfi et al. [24] showed that Naïve Bayes Multinomial performs better in bug severity prediction than Naïve Bayes, because Naïve Bayes Multinomial represents the word counts in documents, and a term’s existence, in general.

In this study, we propose the ES-Multinomial model to predict bug severity level. The severity in ES-Multinomial, $ES_{Severity}(project)$, can be expressed by severities extracted from the bug reports in open source projects, including Eclipse, GNU, JBoss, Mozilla, and WireShark, as follows.

$ES_{Severity}_{(Eclipse)}$ = {Blocker, Critical, Major, Normal, Minor, Trivial}
$ES_{Severity}_{(GNU)}$ = {Critical, Normal, Minor, Trivial}
$ES_{Severity}_{(JBoss)}$ = {Blocker, Critical, Major, Minor, Trivial}
$ES_{Severity}_{(Mozilla)}$ = {Blocker, Critical, Major, Normal, Minor, Trivial}
$ES_{Severity}_{(WireShark)}$ = {Blocker, Critical, Major, Normal, Minor, Trivial}

### Table 4 An example of bug preprocessing

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Summary</th>
<th>Description</th>
<th>Preprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>#437094</td>
<td>Fix EMFilter</td>
<td>The EMFilter must be updated to filter out new models added to luna.</td>
<td>update filter model add luna fix emfilter</td>
</tr>
<tr>
<td>#397442</td>
<td>Frozen UI (Deadlock?)</td>
<td>During editing the IDE got frozen.</td>
<td>editing ide frozen frozen frozen ui lrb deadlock rrb</td>
</tr>
</tbody>
</table>

* https://bugs.eclipse.org/bugs/show_bug.cgi?id=397442

Bug report in Eclipse. We note that the bug report sentence is broken. In detail, the special characters and stop words were removed. Also, each word token was changed to the root word.

#### 3.2 Emotion Similarity

Similar-document search algorithms are widely adopted in the software evolution area and in others. In this paper, we compute the probability vector of emotion terms by using smoothed UM [10], [26], and we find the distribution between a new bug and historical bug reports by using KL-divergence [10], [26]. We provide formulas as follows.

**Smoothened Unigram Model** (which we call Smoothed UM in this paper):

$$P_{ESUM}(ω|\vec{ο}_k) = (1 - \mu) \frac{ω_k(n)}{\sum_{i=1}^{|R_k|} ω_i(n)} + \mu \frac{K \sum_{i=1}^{K} ω_i(n)}{\sum_{l=1}^{K} \sum_{i=1}^{|R_k|} ω_i(n)}$$

(1)

where

- $ω$ is an emotion term, and $\vec{ο}_k$ is a weight vector of report $k$.
- $|R_k|$ is the number of emotion words in report $k$.
- $ω_k(n)$ is the value for the occurrence frequency of the $n^{th}$ emotion term in report $k$.
- $K$ denotes the total number of reports.
- $\mu$ is the different weight of two parts in this equation.

\(^{1}\)https://bugs.eclipse.org/bugs/show_bug.cgi?id=437094
ESSeverity_{project} is a multi-label severity level. We also adopt this idea as our baseline in the machine learning algorithm when we train the model.

- project stands for a bug report project.
- severity level represents a bug severity level in the bug report for the corresponding project, and denotes all the bug severity levels in Eclipse, GNU, JBoss, Mozilla, and WireShark. This consists of Blocker, Critical, Major, High, Normal, Medium, Minor, Low, Trivial, and Small.

Next, we find similar-emotion bug reports by using emotion-based KL-divergence as follows:

\[ RPT_{query} = \{ \omega_k | \omega_k \in B \text{ and } S_{\text{EKL}}(\tilde{\omega}_q, \tilde{\omega}_k) > \theta \}, \quad (k = 1 \ldots N) \]

- \( RPT_{query} \) is similar reports based on emotion words.
- \( B \) denotes a set of bug reports in the bug repository.
- \( N \) is the number of bug reports in \( B \).
- \( \theta \) is a similarity threshold. By adjusting the threshold, we can extract the emotion bug reports according to emotion similarity.
- \( \tilde{\omega}_q \) is a weight vector for the new bug report.

According to \( RPT_{query} \), we find similar-emotion bug reports between a new bug report and historical bug reports by using emotion terms. We set a similarity threshold to \( \theta \). By adjusting the threshold, we can extract emotion bug reports according to emotion similarity. To convert the string text into weight vectors, we adopt StringToWordVector class that represents the word occurrence of the text [27]. Then, we train the model with the emotion bug reports extracted from \( RPT_{query} \) in Naïve Bayes Multinomial, calling it ES-Multinomial. Finally, when the bug reports are submitted, the severity level will be predicted by ES-Multinomial.

4. Experiments

4.1 Data Set

We collected bug reports from open source projects, including Eclipse [12], GNU [13], JBoss [14], Mozilla [15], and WireShark [16], via the parsing technique shown in Table 5.

We note that all the bug reports are labeled Fixed, Resolved and Closed. A severity level of Enhancement in Table 5 A summary of our data

<table>
<thead>
<tr>
<th>Project</th>
<th># of Bug Reports</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>5,000</td>
<td>2001/10/10–2002/06/06</td>
</tr>
<tr>
<td>GNU</td>
<td>5,000</td>
<td>1999/08/03–2003/07/21</td>
</tr>
<tr>
<td>JBoss</td>
<td>3,929</td>
<td>2013/01/02–2015/12/30</td>
</tr>
<tr>
<td>Mozilla</td>
<td>5,000</td>
<td>2002/09/27–2004/09/22</td>
</tr>
<tr>
<td>WireShark</td>
<td>5,000</td>
<td>2005/04/06–2016/01/13</td>
</tr>
</tbody>
</table>

To avoid data bias, we utilize 10-Fold cross validation [7], [29], divided into nine training samples and one testing sample that was not included in the training sample. We also adopt this idea for our baseline in the experiment.

To the best of our knowledge, EWD-Multinomial [7] is the closest related study to our study with ES-Multinomial in emotion-based bug severity prediction. Also, we consider the basic machine learning algorithm, e.g., Naïve Bayes Multinomial [7], [24], and the Lamkanfi et al. study [24], which are well-known, can easily execute the process, and

4.2 Metrics and Baseline

To measure the performance of bug severity prediction, we adopt the evaluation metrics Precision [7], [28], Recall [7], [28], and F-measure [7], [28] which are well-known in the text mining area.

\[
\begin{align*}
\text{Precision (BugSeverity)} &= \frac{\text{SeverityTP}}{\text{SeverityTP} + \text{SeverityFP}} \\
\text{Recall (BugSeverity)} &= \frac{\text{SeverityTP}}{\text{SeverityTP} + \text{SeverityFN}} \\
\text{F-Measure (BugSeverity)} &= \frac{2 \times \text{Precision (BugSeverity)} \times \text{Recall (BugSeverity)}}{\text{Precision (BugSeverity)} + \text{Recall (BugSeverity)}}
\end{align*}
\]

- \( \text{BugSeverity} \) is a multi-label severity level.
- \( \text{SeverityTP} \) means the severity is correctly predicted.
- \( \text{SeverityFP} \) denotes a severity that is incorrectly predicted (unexpected).
- \( \text{SeverityFN} \) represents severity that is incorrectly predicted and which is not the actual severity (missing).

To avoid data bias, we utilize 10-Fold cross validation [7], [29], divided into nine training samples and one testing sample that was not included in the training sample. We also adopt this idea for our baseline in the experiment.

To the best of our knowledge, EWD-Multinomial [7] is the closest related study to our study with ES-Multinomial in emotion-based bug severity prediction. Also, we consider the basic machine learning algorithm, e.g., Naïve Bayes Multinomial [7], [24], and the Lamkanfi et al. study [24], which are well-known, can easily execute the process, and

Table 6 A summary of our emotion data

<table>
<thead>
<tr>
<th>Emotion Word</th>
<th>Non-Emotion Word</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Words</td>
<td>38,287</td>
</tr>
</tbody>
</table>
provide the highest citation in bug severity prediction. We provide a summary of each study as follows.

- **EWDMultinomial** [7]: It computes the emotion score by using each emotion score and trains the model with Naive Bayes Multinomial.
- **Lamkanfi et al.** [11]: This study first organized the bug reports by using Product and Component meta-fields in the bug reports. Then, the model was trained with Naive Bayes.
- **Naive Bayes Multinomial** [7], [24]: This algorithm considers word existence as well as the frequency of a word in the documents. Lamkanfi et al. found that Naive Bayes Multinomial performed better than other algorithms in bug severity prediction [24].

### 4.3 Research Questions

In this paper, we conduct an experiment with the following research questions (RQs).

**RQ1. Which parameters more affected than others in the bug severity prediction are?**

When we predict the bug severity, the parameter of smoothed UM needs to be selected that more affects the predicting performance. Thus, we verify the parameter of smoothed UM to find the best performance.

**RQ2. How accurate do our prediction model achieve?**

Before comparing it with our baseline, we need to verify the performance of bug severity prediction from using ES-Multinomial. This question should be addressed to verify the effectiveness of ES-Multinomial.

**RQ3: Could ES-Multinomial be adopted for bug severity prediction?**

To verify the effectiveness of ES-Multinomial, we need to compare it with our baselines, including EWD-Multinomial, the Lamkanfi et al. study, and Naive Bayes Multinomial. Moreover, we execute a statistical test [30], [31] to verify the difference between our approach and others. In this paper, we conduct an experiment to answer these research questions.

### 4.4 Result

**RQ1. Which parameters more affected than others in the bug severity prediction are?**

We execute ES-Multinomial by adjusting the parameter in smoothed UM, as shown in Fig. 4 for Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively. The X-axis is the values of parameter $\mu$ in smoothed UM, and the Y-axis denotes the ratio of the F-measure to the 10-Fold score for Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively. We present the values including the maximum value (MAX) and minimum value (MIN) in each project. We note that there is little difference between 0.0 and 1.0 at each parameter. In this paper, we will adjust these max parameters (0.0, 0.0, 0.3, 0.0, and 0.3) to execute ES-Multinomial in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively. For example, we set the parameter to 0.3 in JBoss.

**Answer 1:** We verified the suitable parameters (0.0, 0.0, 0.3, 0.0, and 0.3) by using our approach in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.

**RQ2. How accurate do our prediction model achieve?**

In Fig. 5, the number of folds are shown on the X-axis. The last column on the X-axis means the average score of each fold (1 to 10). The Y-axis denotes the prediction performance from the F-Measure. We note that the overall accuracy values on average are 70.86%, 80.32%, 88.12%, 55.65%, and 41.63% for F-measure in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.

**Answer 2:** The overall accuracy values of bug severity prediction on average are 70.86%, 80.32%, 88.12%, 55.65%, and 41.63% for F-measure in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.

**RQ3: Could ES-Multinomial be adopted for bug severity prediction?**

In Fig. 6, Fig. 7, and Fig. 8, the X-axis is the project
The null hypotheses are following:
- \( H_{10}, H_{20}, H_{30}, H_{40}, H_{50} \): There is no difference between this study and EWD-Multinomial in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.
- \( H_{60}, H_{70}, H_{80}, H_{90}, H_{100} \): There is no difference between this study and Lamkanfi in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.
- \( H_{110}, H_{120}, H_{130}, H_{140}, H_{150} \): There is no difference between this study and Multinomial in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.

The corresponding alternative hypotheses are:
- \( H_{1a}, H_{2a}, H_{3a}, H_{4a}, H_{5a} \): There is a difference between this study and EWD-Multinomial in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.
- \( H_{6a}, H_{7a}, H_{8a}, H_{9a}, H_{10a} \): There is a difference between this study and Lamkanfi in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.
- \( H_{11a}, H_{12a}, H_{13a}, H_{14a}, H_{15a} \): There is a difference between this study and Multinomial in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively.

First, we adopt each 10-Fold score from the F-Measure in ES-Multinomial, EWD-Multinomial, Lamkanfi, and NaiveBayes Multinomial. Then we compute the normality via a Shapiro-Wilk test [32] in R language [33]. If the normality value is larger than or equal to 0.05, we run a T-Test. Otherwise, we run a Wilcoxon signed-rank test. The results are shown in Table 7.

For example, the value of 0.003906 in \( H_{10} \) is smaller than 0.05. Thus, we can accept \( H_{1a} \). It means that there is a difference between this study and EWD-Multinomial in Eclipse. We note that we accept the alternative hypotheses in all projects.

Answer 3: According to the results, ES-Multinomial can be adopted for bug severity prediction.

5. Discussion

5.1 Experiment Analysis

Performance of ES-Multinomial: ES-Multinomial can predict the bug severity level well in Eclipse, GNU, and JBoss. Overall accuracy with ES-Multinomial is higher than baseline in Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively. We also present the statistical test by using 10-Fold values from the F-Measure in each project. We verified the difference between our approach and our baselines. However, the values of 55.65% and 41.63% (in Mozilla and WireShark, respectively) are low to predict the bug severity. In the future, we plan to investigate this issue, and use more features to our model.

Emotion Similarity: As our motivation for bug severity prediction, the prediction method needs to consider the reporters’ emotional expressions, because bug reports are written by end users and developers, who are human. Thus,
the reports cannot avoid inevitable emotional expressions. In this paper, we find similar-emotion bug reports by using KL-divergence. To compute the probability of the emotion word vector, we adjusted the parameter in the smoothed UM. By using emotion similarity, we can predict the bug severity level well. In addition, we verified that the difference is very tiny in 0.0 to 1.0 in smoothed UM. In the future, we will adjust the parameter by using the similarity threshold of KL-divergence.

**Emotion Analysis:** EWD-Multinomial considers emotion analysis by using emotion-term scoring in the bug reports. But the performance of EWD-Multinomial is lower than ES-Multinomial. We analyzed this issue and found that if we adopt emotion similarity, the number of emotion words is higher than with EWD-Multinomial, because the emotion frequency in historical bug reports is greater than in emotion bug reports. We provide the number of emotional terms in each project as shown in Table 8. EWD-Multinomial can also reflect human emotion expression, but our approach outperforms EWD-Multinomial. However, we need to investigate the emotional terms of bug report when we predict the bug severity in the future.

### Table 7: A statistical test result

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>P-Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H_{10} )</td>
<td>0.003906</td>
<td>( H_{1a} ): Accept</td>
</tr>
<tr>
<td>( H_{20} )</td>
<td>0.001953</td>
<td>( H_{2a} ): Accept</td>
</tr>
<tr>
<td>( H_{30} )</td>
<td>0.001953</td>
<td>( H_{3a} ): Accept</td>
</tr>
<tr>
<td>( H_{40} )</td>
<td>0.001953</td>
<td>( H_{4a} ): Accept</td>
</tr>
<tr>
<td>( H_{50} )</td>
<td>3.494E-07</td>
<td>( H_{5a} ): Accept</td>
</tr>
<tr>
<td>( H_{60} )</td>
<td>1.214E-14</td>
<td>( H_{6a} ): Accept</td>
</tr>
<tr>
<td>( H_{70} )</td>
<td>6.683E-12</td>
<td>( H_{7a} ): Accept</td>
</tr>
<tr>
<td>( H_{80} )</td>
<td>0.001953</td>
<td>( H_{8a} ): Accept</td>
</tr>
<tr>
<td>( H_{90} )</td>
<td>1.121E-15</td>
<td>( H_{9a} ): Accept</td>
</tr>
<tr>
<td>( H_{100} )</td>
<td>0.0009764</td>
<td>( H_{10a} ): Accept</td>
</tr>
<tr>
<td>( H_{110} )</td>
<td>0.001953</td>
<td>( H_{11a} ): Accept</td>
</tr>
<tr>
<td>( H_{120} )</td>
<td>1.918E-15</td>
<td>( H_{12a} ): Accept</td>
</tr>
<tr>
<td>( H_{130} )</td>
<td>5.767E-09</td>
<td>( H_{13a} ): Accept</td>
</tr>
<tr>
<td>( H_{140} )</td>
<td>2.807E-13</td>
<td>( H_{14a} ): Accept</td>
</tr>
<tr>
<td>( H_{150} )</td>
<td>0.0001042</td>
<td>( H_{15a} ): Accept</td>
</tr>
</tbody>
</table>

### Table 8: The number of emotional terms

<table>
<thead>
<tr>
<th></th>
<th>ES-Multinomial</th>
<th>EWD-Multinomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>255,263</td>
<td>6,247</td>
</tr>
<tr>
<td>GNU</td>
<td>9,444</td>
<td>6,856</td>
</tr>
<tr>
<td>JBoss</td>
<td>6,302</td>
<td>4,358</td>
</tr>
<tr>
<td>Mozilla</td>
<td>12,544</td>
<td>8,668</td>
</tr>
<tr>
<td>WireShark</td>
<td>10,109</td>
<td>7,187</td>
</tr>
</tbody>
</table>

#### 5.2 Threats to Validity

We present some possible threats of this study including internal threat, external threat, and terminology threat as follows.

**Construct threat:** We adopt a common evaluation metrics including Precision, Recall, and F-Measure when we evaluate the effectiveness of our approach. In addition, we execute a statistical test including T-Test and Wilcoxon signed-rank test.

**Internal threat:** In this study, we achieved the suitable parameter by using adjusting smoothed UM. In details, we set the parameter to 0.0, 0.0, 0.3, 0.0, and 0.3 in the smoothed unigram model for Eclipse, GNU, JBoss, Mozilla, and WireShark, respectively. However, we cannot conclude the parameters are the suitable performance in all the other projects. Moreover, another feature needs to consider to improve the performance of our model.

**External threat:** We perform an experiment with ES-Multinomial to show the effectiveness of severity prediction performance for Eclipse, GNU, JBoss, Mozilla, and WireShark. But we cannot conclude that our approach can be adopted in all of the open source projects and business projects owing to the differences in the data types and other processes.

### 6. Related Work

Recently, various techniques have been proposed by researchers into bug severity prediction. We provide a qualitative comparison with related works in bug severity prediction, as shown in Table 9.

Yang et al. (2017) [7] proposed EWD-Multinomial to predict severity with the given bugs. In detail, they construct an emotion dictionary and compute emotion scores between historical bug reports and new bug reports. Then, they apply Naïve Bayes Multinomial.

Zhang et al. (2016) [2] utilized various techniques including Topic-LDA, BM25, and the KNN algorithm to predict
bug severity. First, they execute a preprocessing technique and compute the similarity between topic model and meta-field (including Product and Component) in the bug report. Then, they executed a preprocessing technique and divided the data set into a training set and a test set to construct the experiment. They implemented the training model, and evaluated the model by using test bug reports. 

Yang et al. (2014) [34] predicted bug severity by using Topic-LDA, meta-field, KL-divergence, and a KNN algorithm. First, they construct the topic model by using historical bug reports, and find which topic belongs to the new bug reports by using term-matching frequency. Next, they apply a meta-field (including Product, Component, and Priority), and find similar bug reports by using KL-divergence. Finally, they execute the KNN algorithm, and then predict the bug severity.

Yang et al. (2014) [34] predicted bug severity by using Topic-LDA, meta-field, KL-divergence, and a KNN algorithm. First, they construct the topic model by using historical bug reports, and find which topic belongs to the new bug reports by using term-matching frequency. Next, they apply a meta-field (including Product, Component, and Priority), and find similar bug reports by using KL-divergence. Finally, they execute the KNN algorithm, and then predict the bug severity.

**Table 9**  A qualitative comparison

<table>
<thead>
<tr>
<th>Publisher (Published Year)</th>
<th>Technique</th>
<th>Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAC (2017)</td>
<td>Emotion Analyze</td>
<td>Naïve Bayes Multinomial</td>
</tr>
<tr>
<td>JSS (2016)</td>
<td>LDA, BM25, KNN</td>
<td>-</td>
</tr>
<tr>
<td>COMPSAC (2014)</td>
<td>LDA, Meta-field, KL, KNN</td>
<td>-</td>
</tr>
<tr>
<td>APSEC (2012)</td>
<td>Feature Selection</td>
<td>Naïve Bayes Multinomial</td>
</tr>
<tr>
<td>WCRE (2012)</td>
<td>BM25, KNN</td>
<td>-</td>
</tr>
<tr>
<td>MSR (2010)</td>
<td>Meta-field</td>
<td>Naïve Bayes</td>
</tr>
<tr>
<td>ICSM (2008)</td>
<td>Rule-learning</td>
<td>-</td>
</tr>
<tr>
<td>Our study</td>
<td>Emotion Similarity</td>
<td>Naïve Bayes Multinomial</td>
</tr>
</tbody>
</table>

The main differences are the following.

- Other researchers (excluding EWD-Multinomial) do not consider the reporters' emotional expressions as a feature. Because the bug reports are written in a freeform textual context, they need to verify them with emotion analysis. Therefore, the bug report should reflect human emotional expressions when predicting bug severity.
- EWD-Multinomial can reflect human expressions when they predict bug severity. However, it only considers a bug reports’ emotion score. If a number of emotion bug reports are considered, we can reflect the emotion expression well. In this paper, we find the emotion bug reports by using emotion similarity to predict the bug severity effectively.

7. Conclusion

ES-Multinomial was proposed in this paper to reflect the bug reporters’ emotion effects when predicting bug severity levels. In detail, we construct an emotion dictionary. By adopting a smoothed unigram model and KL-divergence, we computed the probability of emotion words and found similar emotion reports between a new bug report and historical emotional bug reports. Next, we modified the Naïve Bayes Multinomial algorithm, calling it Emotion Similarity Multinomial. We compared our approach with our baselines (including EWD-Multinomial, Naïve Bayes Multinomial, and a Lamkanfi et al. study) in Eclipse, GNU, JBoss, Mozilla, and WireShark open source projects. The results show that our ES-Multinomial can reflect the reporters’ emotional expressions, and it outperforms the others. In the future, we would like to investigate a deep-learning algorithm [38] to predict bug severity levels.

Acknowledgments

This research was supported by Next-Generation Information Computing Development Program (NRF-2014M3C4A7030504) and by Basic Science Research Program (NRF-2017R1A2B4009937) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning.

References


Tao Zhang received the B.S. and M.E. degrees in Automation and Software Engineering from Northeastern University, China, in 2005 and 2008, respectively. He got the Ph.D. degree in Computer Science from University of Seoul, South Korea in Feb., 2013. He was a postdoctoral fellow at the Department of Computing, Hong Kong Polytechnic University from November 2014 to November 2015. Currently, he is an associate professor at the College of Computer Science and Technology, Harbin Engineering University. His research interest includes mining software repositories and software security. Dr. Zhang is a member of IEEE and ACM.

Byungjeong Lee received the B.S., M.S., 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, South Korea. His research areas include software engineering and data engineering.