Analyzing Emotion Words to Predict Severity of Software Bugs: A Case Study of Open Source Projects

Geunseok Yang[1], Seungsuk Baek[1], Jung-Won Lee[2], Byungjeong Lee[1*]
Dept. of Computer Science, University of Seoul[1], Dept. of Electrical and Computer Engineering, Ajou University[2]
{ypats87, baekss0409, bjlee}@uos.ac.kr[1], jungwony@ajou.ac.kr[2]

ABSTRACT

A successful software development project becomes an essential part of a software company's reputation. Thus, lots of project managers focus more on maintenance than on other management processes. Previous works studied how to help the maintenance process by detecting bug duplication and predicting the severity of bugs. This paper continues that kind of special work by analyzing emotion words for bug-severity prediction. In detail, we construct an emotion words–based dictionary for verifying bug reports' textual emotion analyses based on positive and negative terms. Then, we modify a machine learning algorithm, the Naïve Bayes multinomial, calling the new algorithm EWD-Multinomial. We compare this EWD-Multinomial study with our baselines, including Naïve Bayes multinomial and a Lamkanfi study, for open source projects such as Eclipse, Android, and JBoss. The result shows this study's algorithm outperforms the others.

CCS Concepts

• Software and its engineering → Software notations and tools → Software maintenance tools

Keywords

Bug Severity Prediction, Emotion Words–based Dictionary, Bug Report, Software Maintenance.

1. INTRODUCTION

A lot of software companies want their software products to be widely used over the long term in their respective fields. To achieve successful projects, many software programmers are involved, each one responsible for developing specific components in the product. As a result, the software cannot avoid mistakes and bugs during its development.

For the three years from 2013 to 2015, various software bugs—84,245 in Eclipse, 107,456 in Android, and 81,920 in JBoss—were submitted to a software bug repository. We note that bugs reported in Android and JBoss tended to increase in frequency. To efficiently manage that many reports, many projects adopted a bug-tracking system [1] that can assign bugs to developers, track the history of the bugs, and help in the bug management process.

In a general bug management process, developers, quality assurance (QA) specialists, and even end users may find software bugs or functional enhancement issues. They all might write the bug reports for the software repository with their bug explanations in the descriptions and a bug severity level based on their background knowledge. Developers who are assigned bug fixes try to manage on their own which bugs should be addressed first, based on the bug severity level assigned with the bug, and try to fix the bugs by using the descriptions in the reports. Thus, bug severity plays an essential role [2] in assigning developers to address certain bugs before others.

The motivations of this study are as follows:

- The bug severity level can be affected by a subjective decision, because bug reporters have different background knowledge. Thus, it is necessary to make an objective decision through automatic prediction of software bug severity.
- People usually write the reports, so human emotion affects the explanations in them. We observed that Eclipse and JBoss have more than non-severe in negative terms.

Due to the reporters' limited background knowledge, the bug severity in the report can be confused [3] (e.g., incorrectly assigned, which might increase the developers' workload, and increase the bug's lifetime). Recently, a lot of related studies [3, 4, 5] were proposed to resolve these problems. These studies could predict bug severity by using LDA [3], textual similarity [4], machine learning [5], and so on. However, even if they improve the accuracy of early severity prediction, overall accuracy still needs to improve. Moreover, human emotion expressed in the reports could be reflected in the severity prediction.

To resolve this problem, we propose a novel approach to predicting bug severity levels by utilizing an emotion words–based dictionary. In detail, we construct an emotion dictionary [6] for verifying bug reports' textual emotion. Then, we introduce EWD-Multinomial, which is a modified Naïve Bayes multinomial [5] classifier, to predict bug severity.

[*] Corresponding Author

Now, we expect that when a new bug is reported to a software repository, bug severity is automatically selected and recommended. Thus, the limitations of differing background knowledge and subjective decisions do not have the same effect, so developers can make own schedules, and bug lifetimes can be reduced.

This study provides the following original contributions.

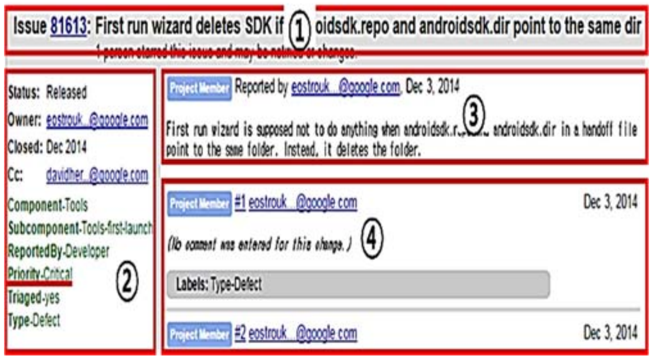
- We are first to use a textual emotion words–based dictionary to implement severity prediction. By improving the accuracy of an earlier study, we can predict bug severity with an objective decision.
- Previous studies utilized traditional machine learning algorithms (Naïve Bayes multinomial, Naïve Bayes, and so on). These algorithms cannot reflect human emotional expression. In this paper, we introduce EWD-Multinomial to resolve this issue.
- Analyzing bug reports' emotions, we found that all of the bug reports at the default level have more negative terms (about 66.83% of normal bugs in Eclipse, about 69.11% of minor bugs in Android, and about 76.79% of major bugs in JBoss) compared to other levels.

The remaining sections of this paper are as follows. Section 2 introduces background knowledge. We present our methodology in Section 3. We implement our experiment in Section 4, and discuss the results and threats in Section 5. Section 6 presents related works, as well as the differences in this study. Finally, we conclude the paper and suggest future works.

2. BACKGROUND KNOWLEDGE

2.1 Bug Report

Reporters may write a bug report to a software repository if they find a bug or functional enhancement issue. The bug report is written in a freeform textual context, and we present a sample bug report for the Android project in Figure 1 [7].



Issue 81613: First run wizard deletes SDK if oidsdk.repo and androidsdk.dir point to the same dir

1 person starred this issue and may be notified of changes.

Status: Released
Owner: eostrouk...@google.com
Closed: Dec 2014
Cc: davidher...@google.com
Component-Tools
Subcomponent-Tools-first-launch
ReportedBy-Developer
Priority-Critical
Triaged-yes
Type-Defect

Project Member Reported by eostrouk...@google.com, Dec 3, 2014

First run wizard is supposed not to do anything when androidsdk.r... androidsdk.dir in a handoff file point to the same folder. Instead, it deletes the folder.

Project Member #1 eostrouk...@google.com Dec 3, 2014

(No comment was entered for this change.)

Labels: Type-Defect

Project Member #2 eostrouk...@google.com Dec 3, 2014

Figure 1. A Summary of Android Bug Report (#81613)

Details in the bug report are as follows:

- This report was submitted Dec/3/2014 (3) and closed Dec/3/2014 (2).
- A title (1) can be outlined against the description (3).
- The report consists of meta-fields (2), e.g., Component, Priority, Type, CC, etc., and those fields can be categorized as similar in the reports.
- Developers or others can give to the community their own ideas and insight into the report (4).
- The bug severity (called *priority* in Android) is "critical" and is classified as a defect in the Android issue tracking system.
- In this paper, we utilize the fields of the report, e.g., Priority (2), Title (1), and Description (3).

To predict bug severity, we need to look over the severity levels of bug reports in detail, because the severity can be expressed by diversity. We present comparison severity levels from three open source projects, such as Eclipse [8], Android [9], and JBoss [10], as shown in Table 1.

The bug reports for Eclipse consist of priority, e.g., P1, P2, P3, P4, or P5, and severity. However, Android and JBoss do not have the same priority levels as Eclipse, and they express severity with the priority. In this paper, we use the term *severity* (not *priority*) so as to avoid conflicting ideas. We note that we exclude *enhancement* in Eclipse and *optional* in JBoss, because they are not used for bugs, but for functional improvements.

Table 1. Comparison Severity Levels Among 3 Projects

Projects	Level	Expression	Re-Expression
Eclipse	Severity	Critical, Blocker, Major, Normal, Minor, Trivial	**Severe**: Critical, Blocker, Major **Non-Severe**: Normal, Minor, Trivial
Android	Priority	Critical, Blocker, High, Medium, Low, Small	**Severe**: Critical, Blocker, High **Non-Severe**: Medium, Low, Small
JBoss	Priority	Critical, Blocker, Major, Minor, Trivial	**Severe**: Critical, Blocker, Major **Non-Severe**: Minor, Trivial

2.2 Emotion Words-based Dictionary

To predict the severity of bugs, we analyze the title and description in a report by using an emotion words–based dictionary [6]. We provide a sample from the emotion words-based dictionary in Table 2.

The emotion words have positive and negative scores, so we can give emotion scores to the bug reports based on the positive and negative word scores.

Table 2. Top 10 Score-based Emotional Terms in Dictionary

	Positive Term (Positive / Negative) Score	Negative Term (Positive / Negative) Score
TOP 1	Good (**15.375** / 0.125)	Bad (0.875 / **10.625**)
TOP 2	Clear (**9.625** / 1.5)	Wrong (0.75 / **8.125**)
TOP 3	Well (**7.792** / 0.708)	Suffer (0 / **7.75**)
TOP 4	Clean (**7.625** / 3.875)	Dead (2.125 / **7.375**)
TOP 5	Right (**7.0** / 0.75)	Rough (1.125 / **7.375**)
TOP 6	Better (**6.75** / 0.25)	Hurt (0 / **7.25**)
TOP 7	Light (**6.25** / 5.25)	Lose (0.25 / **7**)
TOP 8	Respect (**4.5** / 0)	Sting (0 / **7**)
TOP 9	Love (**4.375** / 0.125)	Black (0.625 / **6.875**)
TOP 10	Sound (**4.375** / 0.375)	Trouble (0.625 / **6.5**)

3. METHODOLOGY

In this section, we present an overview of our approach, as shown in Figure 2. First, we collect bug reports from the bug repositories for Eclipse [8], Android [9], and JBoss [10] by using JSON and CSV parsing. Next, we compile an emotion words–based dictionary [6] from an open library into our database system. Then, we perform a preprocessing technique [11], because the reports are based on freeform textual contexts, and we need to break out word tokens. Next, we propose a bug severity prediction model by using emotion analysis. Finally, we modify the Naïve Bayes multinomial algorithm, calling it EWD-Multinomial, to predict bug severity.

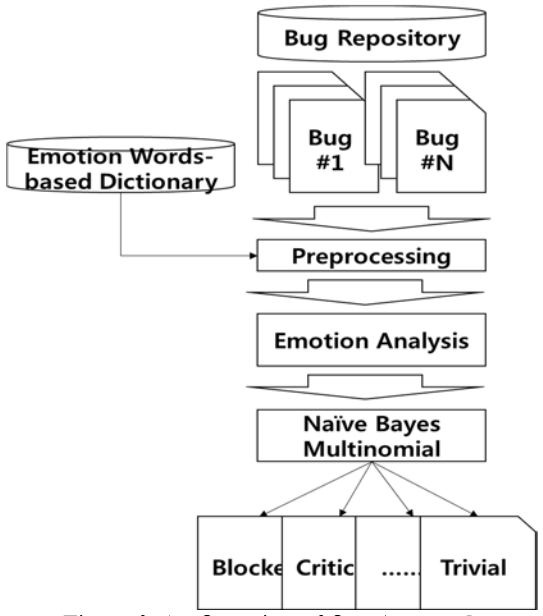


Figure 2. An Overview of Our Approach

3.1 Preprocessing

3.1.1 Bug Report Preprocessing

We briefly introduce a preprocessing technique, including stemming, stop word removal, and special character removal. First, we utilize a stop word list, in general, and we increased the number of stop words based on a manual bug report analysis. In this paper, we designate 760 words as stop words in the reports. The stop words include terms such as *a*, *about*, *am*, *and*, *as*, *are*, and so on. Then we remove special characters, including the hyphen, @, and the abbreviation *etc*. Finally, we use Stanford coreNLP tools [11] to remove word stems. For example, after stemming, the comparative and superlative terms are transformed into the root word, because they have similar meanings. After preprocessing, the bug report sentences are broken into word tokens based on the space delimiter, in general.

3.1.2 Emotion Words-based Dictionary Preprocessing

To analyze bug report emotion, we compile an emotion words–based dictionary into our system. However, we found that emotion terms consist of various words with similar meanings. For example, *patched* (Positive 0 / Negative 0.25), *patching* (Positive 0 / Negative 0), and *patch* (Positive 0 / Negative 0.125) have different weight scores. We perform stemming on the emotion words, and we have the term *patch* scored as Positive 0 and Negative 0.375, e.g., (Positive 0 + 0 + 0), (Negative 0 + 0.25 + 0.125).

3.2 Emotion Analysis

In order to predict severity in the bug reports, first, we assign the severity for those bug reports as follows.

$$BugRPT_{severity} = \{Critical, Blocker, Major, High, Normal, Medium, Minor, Low, Trivial, Small\}$$

- Bug Report consists of 10 severity levels in Eclipse, Android, and JBoss.

Then, we try to compare the terms between the bug report and the emotion words–based dictionary. If they match, this can be represented as follows.

$$EmoTERM = \left\{term \middle| \begin{matrix} term \in emolist, \\ where\ term = term_{emotion\ words} \end{matrix} \right\}$$

- Emotion Term (*EmoTERM*) is a set of terms, which means we can get the emotion score by using those terms.
- *term* represents the word tokens in the bug report.
- *emolist* denotes the terms in the emotion words–based dictionary.

Next, we analyze the emotion of the bug reports by computing the *EmoTERM* as follows.

$$EmoTScore_{severity} = \left\{\frac{pos_{score} + 1}{neg_{score} + 1}\right\}, EmoTScore > 0 \quad (1)$$

- Emotion Term Score is represented by the bug report emotion score.

- pos_{score} means positive score in the $EmoTERM$.
- neg_{score} denotes negative score in the $EmoTERM$.
- To avoid a weight of 0, we add 1 to each score.

Finally, if a new bug report is submitted to software repository, we can analyze the emotion of the report by using Equation 1.

Here, we provide an example of an emotion analysis for Android bug report #81613. First, we looked at the severity in the report, and found that the severity of this bug report was *critical*. After preprocessing, we obtained the seven terms shown in Table 3.

Terms like *sdk*, *androidsdkrepo*, *androidsdkdir*, and *dir* cannot be computed in the emotion analysis because they are not in the emotion words–based dictionary. Finally, the report was scored at 1.875 positive and 0.625 negative.

As defined in Equation 1, Android report #81613 has an $EmoTScore$ of 1.77 (1.875 + 1 / 0.625 + 1).

Table 3. Emotion Analysis Android Report in #81613

Before Preprocessing	After Preprocessing	Emotion Score (Positive/Negative)
First run wizard deletes SDK if androidsdk.repo and androidsdk.dir point to the same dir	wizard	**0.125** / 0.375
	delete	0.0 / **0.125**
	sdk	-
	androidsdkrepo	-
	androidsdkdir	-
	point	**1.75** / 0.125
	dir	-
Sum of Scores	-	**1.875 / 0.625**

3.3 EWD-Multinomial

In order to predict the bug severity, we modify the Naïve Bayes multinomial algorithm, calling it EWD-Multinomial. Before we introduce EWD-Multinomial, we introduce briefly the Naïve Bayes multinomial algorithm. The algorithm considers a term's existence as well as a term's frequency of occurrence.

However, we modify the Naïve Bayes multinomial algorithm by using emotion analysis. In EWD-Multinomial, we assign a class, which means this could be predicted as the target severity or not from a numeric number as follows.

EWD-Machine Learning can be represented by 0 and 1, where 1 means target severity, and 0 is the opposite of the target category. For example, in Eclipse, if the target severity is *critical*, we assign 0 to the severity, including normal, minor, and trivial. We note this idea of class in machine learning to our baselines for equitably comparing.

4. EXPERIMENT

4.1 Summary

The reports are collected via JSON and CSV parsing from the software bug repositories in the Eclipse [8], Android [9], and JBoss [10] tracking systems. The experiment data is shown in Table 4. We note that the Android bug (issue) tracking system does not support exporting to a CSV file with description. Thus, in Android bug severity prediction, we only use summary (title) to predict the severity of bugs.

Table 4. A Summary of Data Set

	# of Bug Reports	Period
Eclipse	43,377	'13/01/01 ~ '15/12/31
Android	93,386	'13/01/01 ~ '15/11/12
JBoss	65,535	'13/01/01 ~ '15/12/30

The bug severity is not labeled with *enhancement* in Eclipse and *optional* in JBoss, because those terms are not for bugs; they indicate a functional improvement. The emotion words–based dictionary data set is depicted in Table 5.

We note that *Emotion Word* means the number of emotion words that have a score as positive or negative. *Non-Emotion Word* denotes the number of words that have no score value, neither positive nor negative.

Our experiment environment was a server machine (database system), a client workstation, and a client personal computer.

Table 5. A Summary of Emotion Data Set

	Before Preprocessing	After Preprocessing
Emotion Word	39,887	38,287
Non-Emotion Word	106,706	104,072

4.2 Evaluation

4.2.1 Metrics

We utilize popular evaluation metrics, such as Precision [12], Recall [12], and F-Measure [12], to verify our study as follows.

$$\text{Precision (Severity)} = \frac{PredTruePositive}{PredTruePositive + PredFalsePositive} \quad (2)$$

$$\text{Recall (Severity)} = \frac{PredTruePositive}{PredTruePositive + PredFalseNegative} \quad (3)$$

$$\text{F-Measure (Severity)} = 2 * \frac{Precision(Severity) * Recall(Severity)}{Precision(Severity) + Recall(Severity)} \quad (4)$$

- *Severity* is the target severity.

- *PredTruePositive* means the severity is correctly predicted.
- *PredFalsePositive* denotes a severity that is not correctly predicted (unexpected).
- *PredFalseNegative* represents a severity that is incorrectly predicted, however, it is not the actual severity (missing).

To reduce data set bias, we perform 10-fold cross validation [13], which divides the data category into nine (training) samples to one (testing) sample that are not involved in each category. Then, we compute an average score from the results. We note that this 10-fold cross validation idea is adopted to our baselines for equitably comparing.

We note that we would like to make a severity prediction tool in the future. Thus, in this paper, we more focus on recall than F-Measure metric. When a new bug report is submitted, we predict whether or not the severity is correct by using our model (we do not consider TOP 1 to TOP x concept).

4.2.2 Baseline
To ensure a fair experiment, we think our baselines should adopt a machine learning technique, just as Lamkanfi (2011) found that a Naïve Bayes multinomial outperforms other machine learning algorithms in bug severity prediction [5]. Thus, we compare the Naïve Bayes multinomial [5] and Lamkanfi study (2010) [14] with our study. We present a summary of the baselines as follows.

- **Lamkanfi (2010) [14]:** First, this study adopted meta-fields of the reports, such as Product and Component. Next, they performed preprocessing on the reports, and compiled two data sets: training and evaluation. Finally, they utilized the Naïve Bayes algorithm.
- **Naïve Bayes Multinomial [5]:** This algorithm considers a term's existence as well as a term's frequency of occurrence. If this technique has lots of terms, it can outperform Naïve Bayes, in general.

4.2.3 Research Question
We form research questions to construct the experiment as follows.

- **RQ1: Do we successfully predict the bug severity?**
- **RQ2: Can this study be adopted for bug severity prediction?**

Based on those research questions, we address the effectiveness of our algorithm's performance. In addition, we compare our study with our baselines, such as Naïve Bayes multinomial and Lamkanfi, to verify prediction performance. Statistical tests (e.g., Wilcoxon signed-rank test [15], T-Test [16], and Shapiro–Wilk test [17]) are also adopted to verify whether this study has a significant difference from the baselines or not.

4.3 Result
In this section, we present the bug severity prediction results by using emotion analysis, as shown in Figure 3, Figure 4, and Figure 5, and our research questions.

RQ1: Do we successfully predict the bug severity?

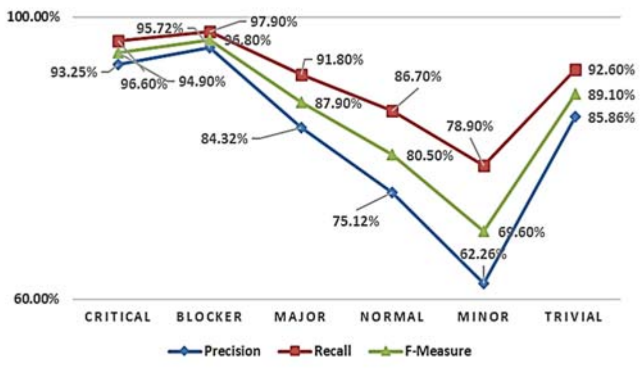


Figure 3. Accuracy of Severity Prediction in Eclipse

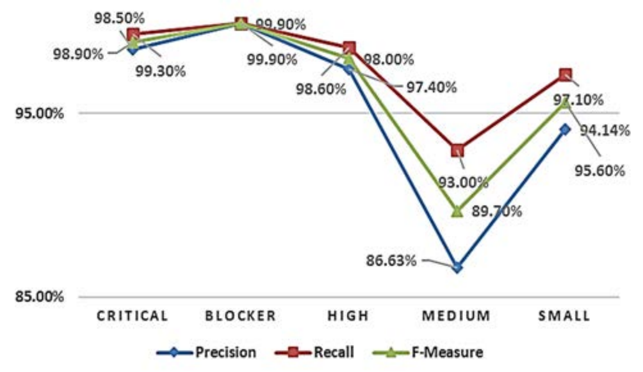


Figure 4. Accuracy of Severity Prediction in Android

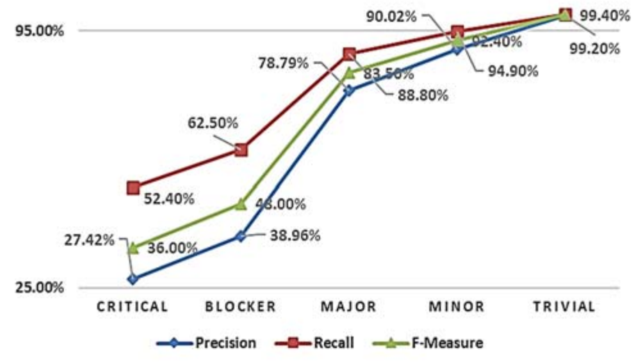


Figure 5. Accuracy of Severity Prediction in JBoss

We note that the X-axis denotes each category of severity in the projects, and the Y-axis is the average 10-fold recall score from 2013 to 2015. The result shows that the overall accuracy is more than about 90.75%, 97.98%, and 79.64% for Eclipse, Android, and JBoss, respectively.

Answer 1: *The result shows that our approach performs the task of bug severity prediction effectively.*

Now, we compare this study with the baselines (see Section 4.2.2), as shown in Figure 6, and we also address the second research question for this comparison.

RQ2: Can this study be adopted for bug severity prediction?

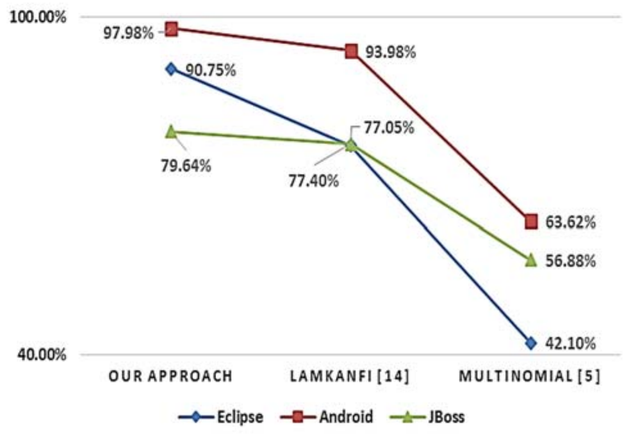


Figure 6. Comparison of Results with Our Baselines

We note that the X-axis is the name of the approach (our approach is EWD-Multinomial), and the Y-axis denotes the average 10-fold recall scores from 2013 to 2015. The result shows that our approach outperforms the others.

> ***Answer 2:*** *This study outperforms Eclipse, Android, and JBoss projects in bug severity prediction and can be adopted.*

In addition, we run a statistical test to verify the difference between this study and the baselines.

The null hypotheses are expressed as follows:

- $H1_0$, $H2_0$, and $H3_0$: There is no difference in this study between Naïve Bayes multinomial in Eclipse, Android, and JBoss, respectively.
- $H4_0$, $H5_0$, and $H6_0$: There is no difference in this study between our baseline and Lamkanfi (2010) in Eclipse, Android, and JBoss, respectively.

The corresponding alternative hypotheses are:

- $H1_a$, $H2_a$, and $H3_a$: There is a difference in this study between Naïve Bayes multinomial in Eclipse, Android, and JBoss, respectively.
- $H4_a$, $H5_a$, and $H6_a$: There is a difference in this study between our baseline and Lamkanfi (2010) in Eclipse, Android, and JBoss, respectively.

First, we compute the normality of results by using a Shapiro–Wilk test in R language [18]. If the return score is larger than or equal to 0.05, we adopt a T-Test. If not, we adopt a Wilcoxon signed-rank test on each project. The results are shown in Table 6. In $H1_0$, we have a p-value of 0.0005972, so our study is significantly different than Naïve Bayes multinomial in Eclipse.

Table 6. A Result of Statistical Test

Null Hypothesis	P-Value	Result
$H1_0$	0.0005972	$H1_a$: Accept
$H2_0$	0.01076	$H2_a$: Accept
$H3_0$	0.01076	$H3_a$: Accept
$H4_0$	0.0004031	$H4_a$: Accept
$H5_0$	0.04375	$H5_a$: Accept
$H6_0$	0.04262	$H6_a$: Accept

> *This study shows more difference between other baselines in Eclipse, Android, and JBoss.*

5. DISCUSSION

5.1 Results in Experiment

In the previous section, we verified the effectiveness of this study, and our study outperforms the baselines. Now we discuss *how can we more correctly predict bug severity than the others*? We present the differences as follows.

- **Experiment Result:** The results of recall are more than 90%, 97%, and 79% in Eclipse, Android, and JBoss, respectively. We investigated the bug reports which are predicted with severity, and then we can classify into 2 categories including human expression report and source code report. We found out some bug reports with source code (just source code copy & paste) in the bug description cannot predict well, but the other ones without source code can predict the correct severity level more than others. For example, if they have a human expression, it could predict the bug severity well. We are going to verify this issue strongly in the future.
- **Emotion Analysis:** In general, bug reports are written by humans, so they can reflect human emotional expression. In this paper, we utilized an emotion analysis technique to verify the bug reports' emotion expressions so we can predict bug severity by analyzing the emotion of the reports. As a result, we can assign a new bug report with a possible bug severity according to historical reports based on emotion analysis.

5.2 Threats and Validity

We present the threats and validity including internal and external aspects as follows.

- **Prediction or Judgement:** There is room between the terms *prediction* and *judgement* in presenting this paper. However, the recall of results for severity prediction is not more than 95%. Therefore, we usually use the term *prediction* instead of *judgement*. In the future, we plan to make a tool and system for bug severity prediction that we would like to automatically recommend to bug reporters.

- **Open Source Project:** In our experiment, we used the bug reports in open source projects, such as Eclipse, Android, and JBoss. We suggest that this study can be more adoptable than others in these projects. However, we need to verify other projects and commercial projects, because they may provide different fields and data.
- **Emotion Words–based Dictionary:** To apply the emotion words–based dictionary, we first remove stems of words, e.g., the terms *patched and patching* become *to patch*, and we compute the sum of the reduced words' scores by grouping similar meanings. This way could interrupt the original meaning of emotion words, and we would like to investigate this in the future.
- **Android Data without Description:** The Android bug (issue) tracking system does not support export to a CSV file with description (it only supports a summary field). Thus, we only use the summary field without description in the Android experiment. However, we will redesign the parsing method to export information, and we will find a way to use the open library in the Android bug tracking system.

6. RELATED WORK

Some researchers have utilized specific techniques, including LDA [3], machine learning [5], and others, in bug severity prediction.

We provide a qualitative comparison with our related works, as shown in Table 7. We note that NB is the abbreviation for Naïve Bayes [5] and M stands for *multinomial*.

Table 7. Technique and Algorithms for Predicting Severity

	Technique	Machine Learning
Zhang (2016)	LDA, BM25, KNN	-
Yang (2014)	LDA, Meta-Field (Product, Component, Priority), KL-Divergence, KNN	
Yang (2012)	Feature Selection	NB. M.
Tian (2012)	BM25, KNN	-
Lamkanfi (2010)	**Meta-Field (Product, Component)**	**NB.**
Menzies (2008)	Rule-learning	-
This study	**Emotion Analyze**	**NB. M.**

In terms of LDA, **Zhang et al. (2016)** [3] proposed bug severity prediction by using topic-based K nearest neighbor (KNN) to find neighboring bug reports and textual similarities.

Yang et al. (2014) [4] proposed an approach by using the topic and meta-fields in reports and KL-divergence. Then, they predicted a new given bug with a KNN algorithm.

In terms of machine learning, **Lamkanfi et al. (2011)** [5] found that the Naïve Bayes multinomial algorithm outperforms other algorithms, such as Naïve Bayes, KNN, and SVM, in severity prediction. Earlier work from 2010 [14] combined text mining and Naïve Bayes, whether the bug was severe or non-severe.

In terms of other algorithms, **Yang et al. (2012)** [19] utilized feature selection, and selected the best features to apply Naïve Bayes multinomial as input.

Tian et al. (2012) [20] proposed a methodology by using a BM25-based KNN algorithm to predict bug severity.

Menzies et al. (2008) [21] predicted bug severity by using rule-learning techniques and text mining.

In the developer prioritization and bug triage area, they found data reduction technique is more effective than other features [23, 24].

The main differences are as follows.

- **Emotion Analysis:** The previous studies were more focused on the question, "*How can we predict the bug severity?*" than on the reporters' expressions. The bug reports are generated by humans, who cannot avoid emotional expressions, in general. In this paper, we can resolve this issue by using emotion analysis.
- **EWD-Multinomial:** Previous studies utilized machine learning algorithms, such as Naïve Bayes multinomial, Naïve Bayes, and so on. These algorithms only use word terms in the bug reports. However, in this paper, we modified Naïve Bayes multinomial by using emotion analysis, which we call EWD-Multinomial.

7. CONCLUSION

In this paper, we proposed an approach to predicting bug severity to reduce developers' efforts by introducing EWD-Multinomial. First, we constructed an emotion words–based dictionary to analyze the emotions expressed in bug reports. Then, we proposed emotion analysis by matching term frequency based on the emotion words–based dictionary and a new bug. Finally, we modified Naïve Bayes multinomial based on the emotion analysis, which we call EWD-Multinomial, to predict bug severity. The results showed that the EWD-Multinomial model outperforms baselines, including Naïve Bayes multinomial and the Lamkanfi study, and our study can reduce developers' tasks.

In the future, we will apply cross-projects to predict bug severity, which means this study does not affect specific projects. We would like to adopt a deep-learning algorithm [22], as well as other types of feature extraction, and provide meta-fields in reports, such as product, component, attachments, and so on.

8. 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 (NRF-2014M3C4A7030504).

9. REFERENCES

[1] Zimmermann, T., Premraj, R., Sillito, J., and Breu, S, "Improving bug tracking systems". In Proc. ICSE Companion, pp. 247-250, 2009.

[2] Zhang, T., Yang, G., Lee, B., and Chan, A. T., "Predicting severity of bug report by mining bug repository with concept profile", In Proc. of the 30th Annual ACM Symposium on Applied Computing, pp. 1553-1558, 2015.

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

[4] 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 Proc. Computer software and applications conference, pp. 97-106, 2014.

[5] Lamkanfi, A., Demeyer, S., Soetens, Q. D., and Verdonck, T., "Comparing mining algorithms for predicting the severity of a reported bug", In Proc. Software Maintenance and Reengineering, pp. 249-258, 2011.

[6] Baccianella, S., Esuli, A., and Sebastiani, F., "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining", In LREC, Vol. 10, pp. 2200-2204, 2010.

[7] https://code.google.com/p/android/issues/detail?can=2&q=81613&id=81613

[8] Eclipse, https://bugs.eclipse.org/bugs/

[9] Android, https://code.google.com/p/android/issues/list

[10] JBoss, https://issues.jboss.org/

[11] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D., "The Stanford CoreNLP Natural Language Processing Toolkit", In Proc. ACL (System Demonstrations), pp. 55-60, 2014.

[12] Goutte, C., and Gaussier, E., "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation", In Proc. European Conference on Information Retrieval, pp. 345-359, 2005.

[13] Kohavi, R., "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.

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

[15] Wilcoxon, F., "Individual comparisons by ranking methods", Biometrics bulletin, Vol. 1, No. 6, pp. 80-83, 1945.

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

[17] Shapiro, S. S., and Wilk, M. B., "An analysis of variance test for normality (complete samples)", In Biometrika, Vol. 52, No. 3/4, pp. 591-611, 1965.

[18] Team, R. C., "R: A language and environment for statistical computing", 2013.

[19] Yang, C. Z., Hou, C. C., Kao, W. C., and Chen, X., "An empirical study on improving severity prediction of defect reports using feature selection", In Proc. 19th Asia-Pacific Software Engineering Conference, pp. 240-249, 2012.

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

[21] Menzies, T., and Marcus, A., "Automated severity assessment of software defect reports", In Proc. IEEE International Conference on Software Maintenance, pp. 346-355, 2008.

[22] LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning", Nature, 521(7553), pp. 436-444, 2015.

[23] Xuan, J., Jiang, H., Ren, Z., and Zou, W, "Developer prioritization in bug repositories", In Proc. 34th International Conference on Software Engineering, pp. 25-35, 2012.

[24] Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., and Wu, X, "Towards effective bug triage with software data reduction techniques", IEEE Transactions on Knowledge and Data Engineering, Vol. 27, No. 1, pp. 264-280, 2015.