Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Bug Severity Prediction Algorithm using Topic-based Feature Selection and CNN-LSTM Algorithm

JungYeon Kim [1], Geunseok Yang [1,*]

[1]Department of Computer Science and Engineering, Kyungnam University, 51767, Gyeongsangnam-do, South Korea;

Corresponding author: Geunseok Yang (gsyang@kyungnam.ac.kr)

ABSTRACT Increasing software usage has gradually increased the occurrence of bugs. When writing a bug report, the severity of the bug can be freely selected, so the subjective judgment of the author is involved. In subjective judgment, a severity error may occur depending on the background knowledge between the user and the developer. To resolve this problem, in this paper, the severity was predicted using the feature selection algorithm of the severity of each topic. We utilize the dataset in Eclipse and Mozilla open source projects. First, we classify bug reports by topic-based severity, and extract features from the severity of each topic. The severity was predicted by learning the characteristics from the CNN-LSTM algorithm, and the F-measure was 90.62% and 93.22% of Mozilla. To evaluate the effectiveness of the proposed model, we compared the baselines including DeepSeverity and EWD-Multinomial studies with Eclipse and Mozilla open source projects and showed that the proposed model is more efficient.

INDEX TERMS CNN-LSTM, Feature Selection, Topic Modeling, Bug Severity Prediction, Software Evolution

I. INTRODUCTION

Software usage is increasing as they are being applied to various fields. Software improvements and bug outbreaks are also rapidly increasing, and approximately 300 bug reports are submitted daily for the Eclipse open source project [1].

In open source projects, when writing a bug report, the author selects the severity of the bug and submits the bug report. Subjective judgment may be involved in selecting the bug severity. However, there is an error in judging the severity of a bug because of the difference in background knowledge between end users and developers [2].

Studies related to predicting bug severity are as follows. Kumari et al. [3] evaluated the bug severity by using K-Nearest Neighbor(KNN), j48, Random Forest(RF), Random Number Generator(RNG), Naïve Bayes(NB), Convolutional Neural Network(CNN), and Multiple Linear Regression(MLR). They also proposed an approach using entropy considering the irregularity of the data. Zhang et al. [4] used several machine learning classification algorithms in bug reports to confirm that NB multi-nominal outperforms NB, 1-neighbor, and Support Vector Machine(SVM). An extended study [5], developed a new severity prediction algorithm based on text similarity. Choudhary et al. [6] used Eclipse's version prediction model, which assigns priority based on the information provided in the bug report. Features potentially affecting the priority of a bug were time, text, author-related, severity, product, and component. Tian et al. [7] predicted the severity of the bug report using the nearest neighbor approach. However, we must improve the prediction performance of bug severity and has to be verified with data from various projects.

This problem is resolved with a bug severity prediction algorithm using a topic-based feature selection [8, 9] and Convolutional Long Short-Term Memory Neural Networks (CNN-LSTM) algorithms [10]. In detail, we classify bug reports by topic. Severity features for each topic are extracted by applying the feature selection algorithm to the classified topics. The extracted features are applied to the CNN-LSTM algorithm to predict the bug severity.

To evaluate the effectiveness of the proposed method, we compared the model with the baselines (DeepSeverity [11] and EWD-Multinomial [12]) in open source projects (Eclipse [13], Mozilla [14]), and it showed superior prediction.

The contribution of this paper is as follows.

- Topic-based feature selection and CNN-LSTM algorithms predicted the bug severity. Also, bug reports

were classified by reporting topic, and features were extracted based on the severity of the topic. Additionally, it was found that predicting the severity based on the severity of each topic was superior to predicting the severity through topic classification.
- To compare the performance of the bug severity model, the baselines (DeepSeverity and EWD-Multinomial) and the open source projects (Eclipse, Mozilla) were compared, and the proposed method had superior predictions.

If accurate bug severity can be predicted, serious bugs can be corrected quickly to improve software quality.

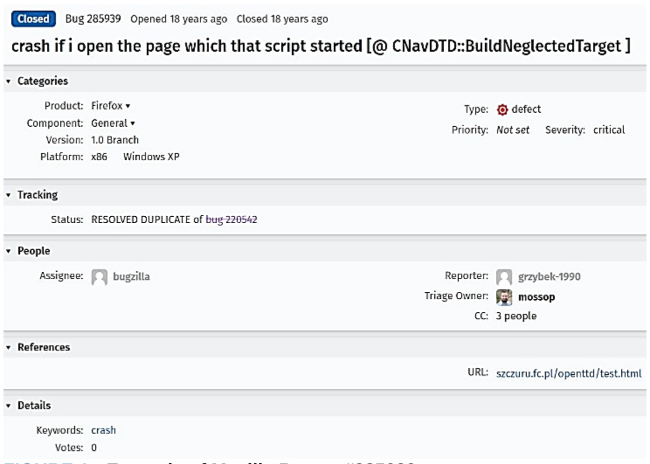
II. BACKGROUND KNOWLEDGE

When using software or programs, bugs occur in unexpected situations. When a user creates a new bug report, the specified bug severity is selected by the project manager and reassigned to an appropriate severity. Since the severity is set first by the subjective judgment of the reporter who wrote the bug report, additional maintenance costs and time may be required in resetting the bug severity. Furthermore, since the severity is reassigned by the individual project manager's judgment, subjective thoughts can be reflected.

If the bug severity is predicted automatically, the objective severity can be reflected so that an efficient software maintenance process can proceed. This study describes bug reports and bug severity for predicting bug severity.

A. Bug Report

Figure 1 shows an example of a bug report is #285939 Mozilla [15].


Closed Bug 285939 Opened 18 years ago Closed 18 years ago

crash if i open the page which that script started [@ CNavDTD::BuildNeglectedTarget]

▾ Categories

Product:	Firefox ▾	Type:	defect
Component:	General ▾	Priority:	*Not set* Severity: critical
Version:	1.0 Branch		
Platform:	x86 Windows XP		

▾ Tracking

Status: RESOLVED DUPLICATE of ~~bug 220542~~

▾ People

Assignee:	bugzilla	Reporter:	grzybek-1990
		Triage Owner:	mossop
		CC:	3 people

▾ References

URL: szczuru.fc.pl/openttd/test.html

▾ Details

Keywords: crash
Votes: 0

FIGURE 1. Example of Mozilla Report #285939.

Figure 1 shows an example of a bug report is #285939 Mozilla [15]. The report for Mozilla Firefox was submitted on March 13, 2005. The bug reporter is *grzybek-1990*, and the developer is *mossop*. The status of the bug report is *RESOLVED DUPLICATE*, and the current bug is *CLOSED*.

B. Bug Tracking System

The bug tracking system [16] enables the communication between a user and a developer and can efficiently manage a bug. Bugzilla, one of the representative bug tracking systems, is an open source tool, and there are meta fields for priority and severity in bug reports. The priority provides information to help developers prioritize bugs that must be fixed. As the attribute value of priority, the priority [17] comprises immediate, urgent, normal, low, and none in five steps of P1 to P5. There are seven levels of severity [18] attribute values. Blocker means that development or testing cannot proceed, and Critical means that the program is broken, data corruption, or memory leak occurs. Major means that a major functional flaw is found, and Normal means a bug that must be fixed as a general problem. Minor is a problem that can be easily solved or has less significant functional flaws, and Trivial is a simple external problem with typos or text misalignment. Enhancements are not a bug but rather represent function and performance improvement requirements.

III. RELATED WORK

Sahin et al. [19] used word embedding to perform feature extraction. Severity predictions were performed using CNN, LSTM, and extreme gradient boosting (XGBoost) methods, and severity scores were measured to substitute the severity levels of the severity predictions.

Kumari et al. [3] proposed an approach using entropy considering the uncertainty and irregularity of the data to evaluate the severity of bug reports. KNN, j48, RF, RNG, NB, CNN, and MLR were applied as training classifiers and verified using PITS, Eclipse, and Mozilla projects. They showed that the proposed method improved the performance.

Kukkar et al. [20] proposed a novel method for assigning severity levels of bugs using swarm intelligence and machine learning. A Naïve Bayes model extracted the optimal feature set from the bug summary and classified the bug report.

Kudjo et al. [21] proposed a method to use Bellwether analysis to extract terms from bug summaries and trained four models (deep neural network, logistic regression, k-nearest neighbors, and random forest). The model's performance was evaluated using standard indicators such as precision, recall, and F-measure. The results showed that the model achieved an F-measure between 14% and 97%.

Lamkanfi et al. [22] determined whether text mining techniques could accurately predict the severity of bug reports and showed that the results could be more accurate.

Tian et al. [23] predicted granular severity levels (e.g., major, major, and minor) using various attributes of past bug reports (e.g., text summary, descriptive text, and title). They proposed an algorithm based on the combination of the BM25Fext similarity method to calculate the similarity between two bug reports and K-NN considering duplicate bugs. The approach was validated using various data sets related to the Mozilla, OpenOffice, and Eclipse open source projects.

Menzies et al. [24] A new technique called SEVERIS was developed to allow test engineers to set the appropriate severity level for a specific bug while validating NASA's closed-source project. SEVERIS mainly used text mining techniques to process textual descriptions of flawed bugs and a rule learning approach to classify bug reports using the RIPPER rule learner method.

Arokiam et al. [25] proposed a method for assigning severity levels of bugs in the early stages of a development project. To predict the severity level of a new bug, they considered how a bug report was written rather than its content. The project results demonstrated that this method could predict the severity of bugs at an early stage of development and outperforms the existing keyword-based models used in many NASA projects.

Roy et al. [26] proposed a feature selection method to improve the prediction accuracy of the severity prediction model. The dataset was developed with Eclipse and Mozilla and using bi-grams and text component-based technologies. This was trained on the Naïve Bayes model and showed improved severity prediction performance. They also proved that the accuracy could vary from project to project and adding more bigrams can degrade performance.

Sharma et al. [27] proposed an approach to predict the priority of new bug reports. This study's accuracy shows that it can successfully predict bug priorities in less than 70% of Eclipse and OpenOffice projects, excluding Naïve Bayes.

Sabor et al. [28] improved the quality of bug reports by integrating the stack trace of bug reports and features by categories such as component, OS, and product. The class imbalance distribution problem is addressed by classifying classes into undersampling and oversampling based on the number of instances representing each class. They then assigned a high misclassification cost to the under sampled class and a low misclassification cost to the oversampled class. The misclassification cost is the ratio of the number of instances of multiple classes to the number of instances of the defined class. They used KNN) to obtain the probability of classifying an unknown instance into a specified number of classes. Finally, they compute the cost of assigning an instance to a given class by multiplying the misclassification cost by the probability of assigning an instance to the class by KNN. The class label with the lowest cost of misclassification was chosen as the class label for the unknown instance. The model's accuracy was measured using precision, recall, and F-measures as usual. The accuracy (F-measure) of the model improved by up to 35% when integrating the stack trace information into the category function of the bug report.

Umer et al. [29] performed another study considering sentiment analysis using SentiWordNet dictionary. Sentiment analysis was applied to bug reports related to the Eclipse open source project to predict the priority of bug reports. Their approach trained the SVM algorithm using sentiment scores and extracted features from bug reports. This approach outperformed other machine learning algorithms, including Naïve Bayes, Naïve Bayes Multinomial, and Linear Regression.

Ramay et al. [30] were the first to use a deep learning algorithm to predict the severity level of a bug report, in addition to the sentiment analysis of bug reporter sentiment expressed in the bug report summary field. They used a popular emotion dictionary called Senti4SD when using SentiWordNet for emotion analysis. We then calculated the emotion score for each bug report on a specific dataset extracted from Bugzilla. Their work significantly improved over the EWD-Multinomial.

IV. OUR APPROACH

This study implements the feature selection algorithm for each topic severity. The CNN-LSTM algorithm is used. Figure 2 shows a schematic of our overall approach. We first classify the topic of bug reports from the bug repository. We proceed with the feature selection algorithm using the severity of bug reports on the topic. The severity-specific features of each topic are put as the input of the CNN algorithm, and the output becomes the input of the LSTM. Finally, the output of the LSTM is the severity.

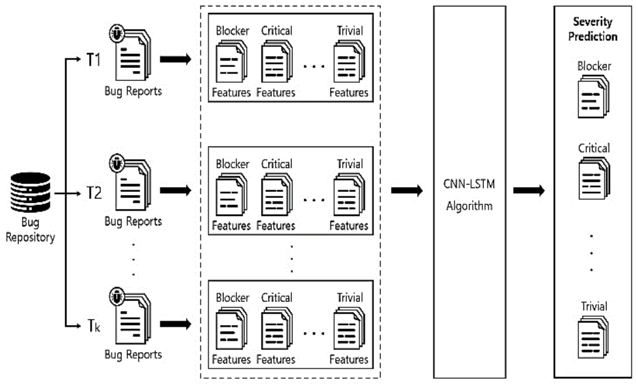


FIGURE 2. Overview of Our Approach.

A. Preprocessing

Bug reporters freely express errors in various forms. There are a summary and a description, and the reporter can write it quickly. To use these bug reports, we must proceed with the preprocessing process among natural language processing technique [31]. In the preprocessing process, tokenization of sentences, stopwords removal, and lemmatization is performed. From the results, sentences were extracted as words, and stop words such as “should”, “be”, “in”, “all”, and “more” were removed.

If there is a sentence of *“Callisto site should be in (all/more) Eclipse Project featues”*, the preprocessed result is as follows. “callisto”, “site”, “all”, “more”, “eclipse”, “project”, “featue”. Also, the word “featues” is extracted as the root of “featue”.

B. Topic Modeling

Usually, to proceed with the topic classification of documents, the topic classification of documents is performed by applying

Latent Dirichlet Allocation(LDA) algorithm [8]. This study regarded a bug report as a document, and the topic of the bug report is classified. Therefore, from the bug report, we can grasp the mainly used
words for each topic and the proportion of the bug report in the topic.

The bug report combines topic distribution and topic word distribution and is calculated through the set parameters. As a representative parameter, α is a Dirichlet distribution parameter for generating a topic distribution of a document, and β refers to a Dirichlet distribution for generating a word distribution of a topic. Furthermore, K means the number of topics designated by the user.

The topics classified in this study are shown in Table I and Table III, and the critical words for each topic are shown in Table II and Table IV. Table I shows samples from Topic-1 to Topic-10. Bug reports for each topic are classified, and because of the set parameters, we can confirm that the bug reports do not belong to various topics. Table II shows the distribution of words by topic. For example, the word Rank#1 in Topic-1 has a distribution of 3878.1783(dialog) and means the distribution of the most important word in Topic-1. Rank #2 in Topic-1 means the distribution of the second most important word in Topic-1 with a distribution of 2711.5182(save). In the same vein, Table 3 and Table 4 are the same as Table 1 and Table 2.

TABLE I
EXAMPLE OF TOPIC CLASSIFICATION IN MOZILLA

Topic-1	Topic-2	Topic-3	Topic-4	Topic-5
803963	72543	321847	261143	722730
590202	117648	675189	703110	410023
254234	709566	112979	424775	418290
802867	445496	622053	224797	287220
Topic-6	**Topic-7**	**Topic-8**	**Topic-9**	**Topic-10**
678030	779036	448369	413028	597714
756903	423275	512743	459150	358268
748587	215608	756292	14462	304928
73603	46656	626919	471008	679266

TABLE II
EXAMPLE OF TOPIC TERM DISTRIBUTION IN MOZILLA

Rank	Topic-1	Topic-2	Topic-3	Topic-4	Topic-5
Rank#1	dialog (3878.17)	event (3764.42)	bug (5665.07)	load (7332.29)	message (10059.73)
Rank#2	save (2711.51)	javascript (3644.33)	character (3240.36)	image (5500.15)	mail (5755.93)
Rank#3	site (2603.86)	element (2802.06)	error (3053.88)	browser (3778.00)	folder (5749.46)
Rank#4	user (2524.96)	attribute (2368.51)	name (2765.44)	time (3170.30)	new (4712.45)
Rank#5	cookie (2422.88)	document (2252.56)	url (2746.93)	cause (3084.26)	email (4314.72)

Rank	Topic-6	Topic-7	Topic-8	Topic-9	Topic-10
Rank#1	plugin (4235.35)	link (6126.31)	text (6366.98)	assertion (4521.22)	menu (9627.80)
Rank#2	install (3208.89)	click (5537.99)	table (5971.91)	failure (4064.81)	bookmark (8941.51)
Rank#3	update (3183.16)	can (5092.39)	display (4276.05)	test (4056.28)	bar (7680.90)
Rank#4	mozillum (2920.24)	not (4965.95)	render (4110.07)	code (2780.43)	button (4728.85)
Rank#5	build (2510.09)	text (4930.85)	font (3890.85)	fail (2721.50)	toolbar (4708.79)

TABLE III
EXAMPLE OF TOPIC CLASSIFICATION IN ECLIPSE

Topic-1	Topic-2	Topic-3	Topic-4	Topic-5
23334	36141	70167	207148	285282
74354	80076	23464	83367	245707
75811	158334	10933	4951	14709
119414	197421	220609	26218	15642
Topic-6	**Topic-7**	**Topic-8**	**Topic-9**	**Topic-10**
822	197605	166449	84152	4602
85625	31321	124550	26079	64530
98420	276732	6924	39488	74515
147615	129725	46148	83705	296708

TABLE IV
EXAMPLE OF TOPIC TERM DISTRIBUTION IN ECLIPSE

Rank	Topic-1	Topic-2	Topic-3	Topic-4	Topic-5
Rank#1	code (2870.37)	search (3476.93)	preference (3759.44)	eclipse (4997.15)	key (2235.00)
Rank#2	javadoc (2780.66)	help (3233.03)	text (3199.47)	can (2900.97)	perspective (2131.75)
Rank#3	wizard (2505.60)	update (2007.35)	page (2331.46)	not (2886.97)	open (2125.04)
Rank#4	import (2165.76)	plugin (1799.29)	color (1360.91)	crash (1820.56)	window (1807.10)
Rank#5	new (2155.10)	property (1691.22)	font (1228.30)	browser (1732.74)	show (1685.02)
Rank	**Topic-6**	**Topic-7**	**Topic-8**	**Topic-9**	**Topic-10**
Rank#1	menu (3117.50)	need (2743.51)	build (3983.34)	npe (2904.08)	method (4963.73)
Rank#2	set (2733.52)	test (2729.77)	source (2570.61)	exception (1778.28)	class (3525.86)
Rank#3	package (2553.41)	apus (2195.19)	launch (2098.56)	compare (1655.72)	assist (2717.85)
Rank#4	action (2042.72)	junit (1963.54)	path (1958.65)	change (1652.24)	content (1995.42)
Rank#5	select (1831.73)	breakpoint (1497.58)	folder (1789.92)	resource (1543.80)	refactor (1964.99)

C. Feature Selection Algorithm

The feature selection algorithm [9] selects useful features from data, filters out duplicate or unrelated features, and automatically selects the most relevant feature subset. This study extracts features by the severity of topics using the feature selection algorithm. First, duplicate features are filtered through a feature subset, and a subset selected with the highest classification performance or correlation is created. Then, features are all input variables of the data model, and a subset (subgroup) of existing features is maintained to reduce the dimension of the data. A subset with a high score to be obtained creates a feature subset. Finally, we combine the feature subset of features with high performance. Therefore, we extract the features of bug reports by topic severity.

D. CNN-LSTM Algorithm

This study applies the features to the CNN-LSTM algorithm, as shown in Figure 3. We put the feature extraction result of topic-based severity as the input of the CNN algorithm [10] and the output again as the input of the LSTM algorithm [10]. The output of LSTM is severity.

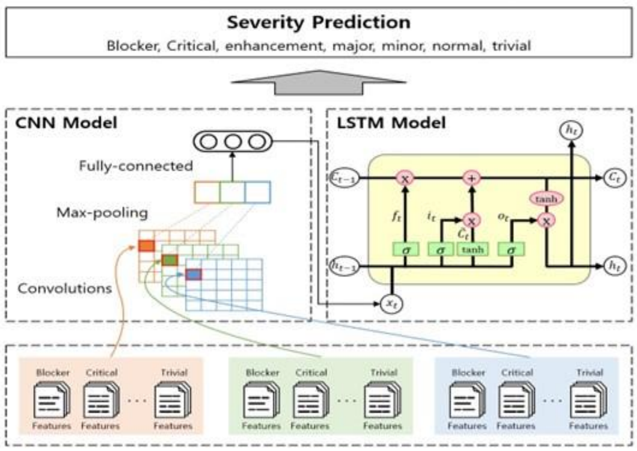


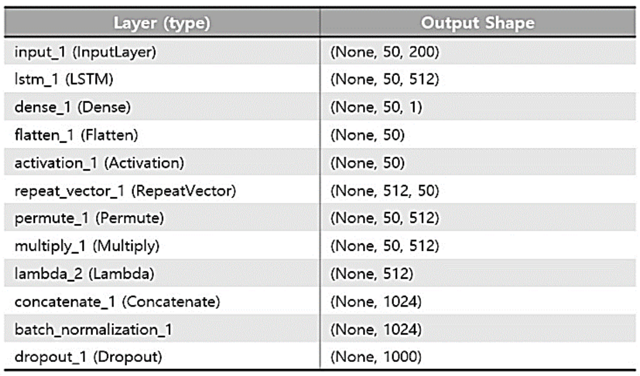
FIGURE 3. **Overview of the CNN-LSTM Algorithm.**

The CNN algorithm comprises a convolution layer that extracts features, a repeating pool layer that compresses the extracted features, and a multilayer perceptron layer. The convolution layer is called a convolutional neural network, and the output data of the current location is a value obtained by multiplying an adjacent cell by a convolution filter. The pooling layer receives the output data of the convolution layer and reduces the output data size (Activation Map) or emphasizes specific data. Pooling layer methods include max pooling, min pooling, and average pooling. In this study, using Max Pooling, the value of the largest feature is used as a representative value of other features. The LSTM algorithm, as one of the leading models of Recurrent Neural Networks(RNN), predicts future data considering previous and past data. LSTM has a structure similar to RNN, but instead of having one neural network layer, there is an interactive 4-way gate. LSTM also has gates between input and output so that situation control can be performed according to the data structure. In addition, since the sequence vector can be modeled identically, the F-measure can be improved when predicting severity. Therefore, we predict the severity of the bug by putting the feature subset into the CNN-LSTM algorithm.

In general, CNN extracts data features well, and LSTM shows good classification characteristics when the correlation of the time series data is large. However, LSTM rather degrades performance when the length of input data is long, and CNN does not solve the long-distance dependence between the data spread out on the time axis. Therefore, in this paper, we input the severity features for each topic into the CNN model and extract the features of the text, taking advantage of both algorithms. The LSTM model is trained by putting the output of the CNN algorithm. The results showed that CNN-LSTM algorithms performed better than existing CNN and LSTM algorithms.

V. EXPERIMENTAL RESULT

To proceed with the experiment, CPU i9-12K, RAM 64 GB, GPU 3090 equipment was used, and the following process is performed. First, we sort bug reports from the bug repository. Build topic modeling using classified bug reports. We apply the feature selection algorithm based on the bug severity criteria for each topic and extract the feature subset. We put the extracted features as the input of the CNN algorithm and the output again as the input of the LSTM algorithm. Finally, we predict the severity. In the model, the hyperparameter is following: *embedding = 200, conv1 = 128, max_pooling = 2, lstm = 512, learning_rate = 1e-4.* The detailed LSTM hyperparameters are shown in Figure 4.



Layer (type)	Output Shape
input_1 (InputLayer)	(None, 50, 200)
lstm_1 (LSTM)	(None, 50, 512)
dense_1 (Dense)	(None, 50, 1)
flatten_1 (Flatten)	(None, 50)
activation_1 (Activation)	(None, 50)
repeat_vector_1 (RepeatVector)	(None, 512, 50)
permute_1 (Permute)	(None, 50, 512)
multiply_1 (Multiply)	(None, 50, 512)
lambda_2 (Lambda)	(None, 512)
concatenate_1 (Concatenate)	(None, 1024)
batch_normalization_1	(None, 1024)
dropout_1 (Dropout)	(None, 1000)

Figure 4. **Our Model the Hyperparameter.**

A. Our Dataset

In this paper, we used Eclipse and Mozilla projects. The total number of Eclipse bug reports was 165,547 (October 10, 2001, to June 14, 2005), and the total number of Mozilla bug reports was 394,878 (July 16, 1999 to September 16, 1999). Also, the severity ratio is the same as that of DeepSeverity [11].

B. Evaluation Metrics

The formula (1) is used to evaluate the proposed model's performance. We note that *S_TP* means that the actual correct answer (True) is predicted as the correct answer (True). *S_TN* means that the actual correct answer (False) is predicted to be not the correct answer (False). *S_FN* means that the actual correct answer (True) is predicted to be not the correct answer (False). *S_TP* means that the actual correct answer (False) is predicted as the correct answer (True). The formula (2) represents the ratio of quality to how well it predicted a result related to the actual 'positive'. The formula (3) represents the positive measure of how well the predicted result predicted in relation to the correct answer. The formula (4) shows the harmonic mean of the formula (2) and (3).

$$S\_Acc = \frac{S\_TP + S\_TN}{S\_TP + S\_FN + S\_FP + S\_TN} \quad (1)$$

$$S\_Pre = \frac{S\_TP}{S\_TP + S\_FP} \quad (2)$$

$$S\_Rec = \frac{S\_TP}{S\_TP + S\_FN} \quad (3)$$

$$S\_F = \frac{2 \times S\_Pre \times S\_Rec}{S\_Pre + S\_Rec} \quad (4)$$

To avoid the data bias, we use 10-fold cross validation in the experiment. The experimental results show the average value for 10 times.

C. Baseline

To evaluate the model performance in this paper, the following baselines and performance are compared. This study uses DeepSeverity [11] as the baseline. DeepSeverity runs Word2Vector on the text and predicts bug severity by applying a CNN algorithm based on text sentiment analysis. In the proposed model, applying the severity feature selection and CNN-LSTM algorithms in the topic has the same logic as the baseline called DeepSeveirty, and predicting the appropriate severity is similar, so it is set as the baseline. Also, we adopt the EWD-Multinomial algorithm [12] as a baseline. They analyze the text based on emotional text analysis. Then, they utilize a Naïve Bayes Multinomial to predict the bug severity.

D. Research Questions

The research experiment is conducted with the following research questions. In this study, research questions are established as follows.

RQ1: Does the proposed model perform adequately in predicting bug severity?

Before comparing with the baselines, we first evaluate the proposed model's performance. If the proposed model shows adequate performance, the performance is compared with the baselines in severity prediction.

RQ2: Can the proposed model be applied in predicting bug severity?

If the proposed model shows good performance by comparing the performance with the baselines, it can be applied in predicting the severity of bugs. This study plans to verify significant differences in bug severity prediction between the proposed model and baseline by performing statistical verification [32] between baselines and simple performance comparison.

E. Result

1) RESULT OF OUR APPROACH

We present the performance of our model in Figure 5.

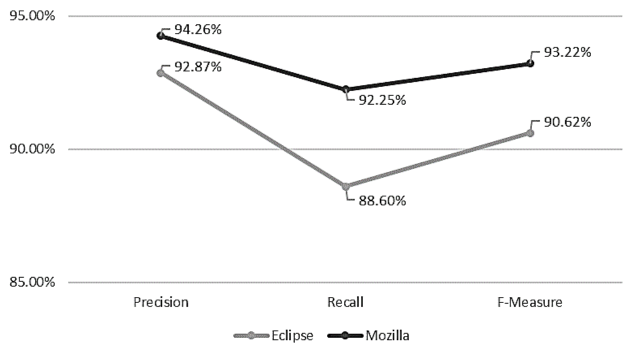


FIGURE 5. **Performance of the Proposed Model.**

We note that the X axis represents Precision and Recall, and F-Measure. In this paper uses Eclipse and Mozilla projects, and overall, it shows performance of about 91% or more. Therefore, our model predicts the severity of bug report well (90.62% in Eclipse and 93.22% in Mozilla). We also compared our model with CNN and LSTM algorithms. The results are shown in Figure 6.

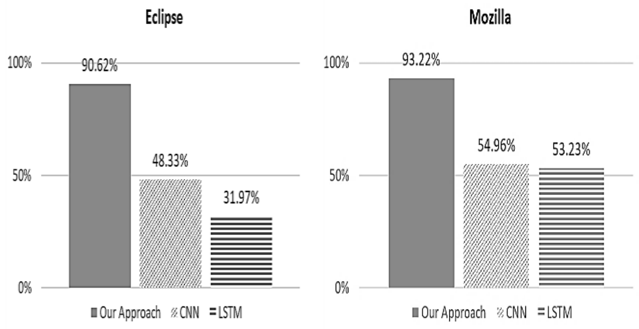


FIGURE 6. **Comparison of Our Model and CNN and LSTM.**

We note that the X axis represents models including our approach CNN and LSTM, and the Y axis represents F-measure. The proposed CNN-LSTM model predicts the bug severity well (Eclipse 90.62% and Mozilla 93.22%). In this paper, when simple CNN and LSTM algorithms were trained, Eclipse 48.33% and Mozilla 54.96% came out for CNN model, and Eclipse 31.97% and Mozilla 53.23% for LSTM model came out. Therefore, the proposed CNN-LSTM was shown to be good in predicting the bug severity.

Also, we compared the performance of those with and without the feature selection algorithm by topic-based severity, as shown in the Figure 7. The performance of the algorithm without applying the topic-based feature selection is about 59%. However, our model predicted the severity of about 91%. Therefore, in this paper, we compare with the baseline by applying the topic-based severity feature selection.

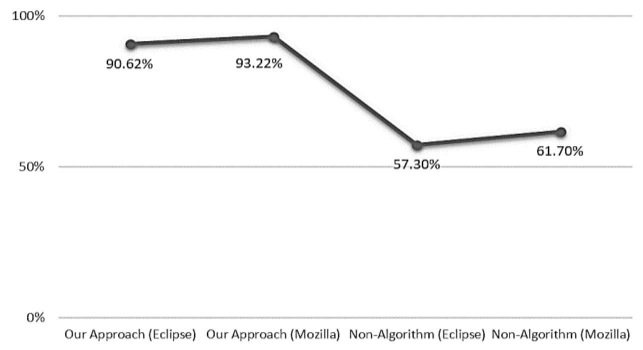


FIGURE 7. Comparison of Our Model and Non-Algorithm (without topic-based feature selection).

2) COMPARISON RESULTS

The performance comparison between the proposed model and baseline (Deep Severity and EWD-Multinomial) is shown in Figure 8. The X axis represents the algorithms including our approach, Deepseverity and EWD-Multinomial, and the Y axis represents F-measure. The proposed model shows the performance of Eclipse 90.62% and Mozilla 93.22%, and the baseline DeepSeverity shows Eclipse 84.79% and Mozilla 84.04%. Also, EWD-Multinomial shows Eclipse 80.97% and Mozilla 76.09%. Therefore, the proposed model predicts better than the baseline (Deep Severity and EWD-Multinomial).

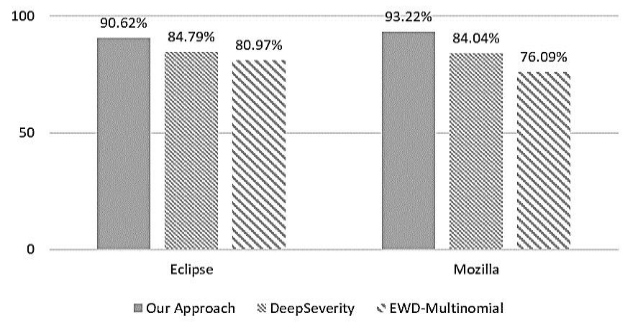


FIGURE 8. Performance Comparison between Baselines.

In addition, statistical verification performs that the proposed model has a statistically significant difference from the baseline.

The null hypothesis for statistical verification is as follows.

$H1_0$: There is no significant difference between DeepSeveirty and Eclipse, where the proposed model is the baseline.

$H2_0$: There is no significant difference between EWD-Multinomial and Eclipse, where the proposed model is the baseline.

$H3_0$: There is no significant difference between DeepSeveirty and Mozilla, where the proposed model is the baseline.

$H4_0$: There is no significant difference between EWD-Multinomial and Mozilla, where the proposed model is the baseline.

The alternative hypothesis to the null hypothesis is as follows.

$H1_a$: There is a significant difference between DeepSeveirty and Eclipse, where the proposed model is the baseline.

$H2_a$: There is a significant difference between EWD-Multinomial and Eclipse, where the proposed model is the baseline.

$H3_a$: There is a significant difference between DeepSeveirty and Mozilla, where the proposed model is the baseline.

$H4_a$: There is a significant difference between EWD-Multinomial and Mozilla, where the proposed model is the baseline.

In details, for statistical verification, we check the results of the data normal distribution. To test for normality, the Shapiro–Wilk test [33] is used. The Shapiro–Wilk test states that the null hypothesis of any test is that the population follows a normal distribution. If the p-value is less than the selected alpha level (0.05), the null hypothesis is rejected and the tested data do not follow a normal distribution. If the result of the normal distribution was 0.05 or more, the t-test [34] statistical verification method was used, and if the result was 0.05 or less, the Wilcoxon signed-rank test [35] statistical verification method was used.

TABLE V
STATISTICAL VERIFICATION RESULTS.

Hypothesis	P-Value	Result
$H1_0$	(T test) 4.885e-16	$H1_a$: Accept
$H2_0$	(T test) 2.106e-12	$H2_a$: Accept
$H3_0$	(T test) 1.874e-14	$H3_a$: Accept
$H4_0$	(T test) 4.062e-12	$H4_a$: Accept

The null hypothesis $H1_0$ means that the proposed model does not significantly differ from DeepSeverity. However, because of the null hypothesis test, a value of 4.885e-16 was obtained, below 0.05, so the null hypothesis was rejected, and the alternative hypothesis was accepted. Therefore, the proposed model has a significant difference in Eclipse than DeepSeverity. The proposed model rejects the entire null hypothesis and accepts the alternative hypothesis.

VI. DISCUSSION

A. Experiment Results

We compared the proposed model with baselines (DeepSeverity and EWD-Multinomial) and severity prediction performance, and the proposed method showed better prediction performance. There was a statistically significant difference. The proposed method applied the feature selection algorithm for each severity in the topic and improved the severity prediction performance with the CNN-LSTM algorithm. It shows the following differences:

- Feature selection by the severity of each topic was performed, and severity was predicted by learning the CNN-LSTM model. It showed better performance than DeepSeverity using CNN algorithm. In addition, the performance of the simple CNN algorithm and LSTM algorithm was compared, and the proposed model showed better performance.
- This study compared performance with that of not applying feature selection by the severity of each topic, and it was confirmed that the application improved the performance. Words characterized by features within the severity of the topic were extracted, and performance was improved.

B. Threats and Validity

This study used the Eclipse, Mozilla open source project with approximately 560,000 data. Since we used some data rather than the whole data, the proposed method cannot always show good performance. Moreover, bug management for business and open source projects is clearly different, such as data structure and severity levels. Therefore, we will further verify other business projects' applicability before performance verification. In the proposed model, the K value is set to 1 in the feature selection algorithm, and the topic modeling hyperparameter is set as the default value. Even when verifying open source projects with different hyperparameters and K values, it cannot always be guaranteed to show good performance. In the future, we plan to improve the model by using various datasets to find K values and optimal hyperparameters.

VII. CONCLUSION

The user's subjective judgment is included in setting the bug severity of the software. If automatic bug severity setting is possible, severity resetting can be improved from the project manager's perspective, and the bug report author must write specifically for the expression of the bug. Therefore, the severity is assigned as an objective element, reducing software maintenance costs and improving software quality. To resolve this problem, in this paper, we applied the feature selection algorithm of each topic severity and put the extracted features as the input of the CNN algorithm. Then, the output is input to the LSTM algorithm to predict the severity. To evaluate the proposed model, performance was compared with the baselines (DeepSeverity and EWD-Multinomial), achieving superior predictions. Additionally, through statistical verification, the proposed model showed a significant difference from the baseline. In the future, we plan to validate the model using various datasets and expand the research using automatic parameter adjustment and various features.

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 IEEE Annual Computer Software and Applications Conference, pp. 97-106, 2014.
[2] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, "Bug Report Severity Level Prediction in Open Source Software: A Survey and Research Opportunities", In Journal of Information and Software Technology, Vol. 115, pp. 58-78, 2019.
[3] M. Kumari, M. Sharma, and V. B. Singh, "Severity Assessment of a Reported Bug by Considering its Uncertainty and Irregular State", In Journal of Open Source Software and Processes (IJOSSP), Vol. 9(4), pp. 20-46, 2018.
[4] T. Zhang, G. Yang, B. Lee, and A. T. Chan, "Predicting Severity of Bug Report by Mining Bug Repository with Concept Profile", In Proc. of Annual ACM Symposium on Applied Computing, pp. 1553-1558, 2015.
[5] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards More Accurate Severity Prediction and Fixer Recommendation of Software Bugs", In Journal of Systems and Software, Vol. 117, pp. 166-184, 2016.
[6] P. A. Choudhary, and S. Singh, "Neural Network based Bug Priority Prediction Model Using Text Classification Techniques", In Journal of Advanced Research in Computer Science, Vol. 8(5), 2017.
[7] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated Prediction of Bug Report Priority Using Multi-Factor Analysis", In Journal of Empirical Software Engineering, Vol. 20(5), pp. 1354-1383, 2015.
[8] Y. Song, S. Pan, S. Liu, M. X. Zhou, and W. Qian, "Topic and Keyword Re-ranking for LDA-based Topic Modeling", In Proc. of ACM conference on Information and knowledge management, pp. 1757-1760, 2009.
[9] W. Shang, H. Huang, H. Zhu, Y. Lin, Y. Qu, and Z. Wang, "A Novel Feature Selection Algorithm for Text Categorization", In Journal of Expert Systems with Applications, Vol. 33(1), pp. 1-5, 2007.
[10] J. Zhang, Y. Li, J. Tian, and T. Li, "LSTM-CNN Hybrid Model for Text Classification", In Proc. of IEEE Advanced Information Technology, Electronic and Automation Control Conference, pp. 1675-1680, 2018.
[11] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B. G. Kang, and N. Chilamkurti, "A Novel Deep-learning-based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting", In Sensors, Vol. 19(13), p. 2964, 2019.
[12] 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 the Symposium on Applied Computing, pp. 1280-1287, 2017.
[13] Eclipse, https://bugs.eclipse.org/bugs, (accessed 30 May 2022).
[14] Mozilla, https://bugzilla.mozilla.org/home, (accessed 30 May 2022).
[15] Mozilla #285939 https://bugzilla.mozilla.org/show_bug.cgi?id=285939, (accessed 30 May 2022).
[16] N. Serrano, and I. Ciordia, "Bugzilla, ITracker, and Other Bug Trackers", In IEEE Software, Vol. 22(2), pp. 11-13, 2005.
[17] Bugzilla Priority, https://wiki.mozilla.org/Bugzilla:Priority_System, (accessed 30 May 2022).

[18] Bugzilla Severity, https://wiki.eclipse.org/WTP/Conventions_of_bug_priority_and_severity, (accessed 30 May 2022).
[19] S. E Sahin, and A. Tosun, "A Conceptual Replication on Predicting the Severity of Software Vulnerabilities", In Proc. of the Evaluation and Assessment on Software Engineering, pp. 244-250, 2019.
[20] A. Kukkar, R. Mohana, and Y. Kumar, "Does Bug Report Summarization Help in Enhancing the Accuracy of Bug Severity Classification?", In Journal of Procedia Computer Science, Vol. 167, pp. 1345-1353, 2020.
[21] P. K. Kudjo, J. Chen, S. Mensah, R. Amankwah, and C. Kudjo, "The Effect of Bellwether Analysis on Software Vulnerability Severity Prediction Models", In Journal of Software Quality Journal, Vol. 28(4), pp. 1413-1446, 2020.
[22] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the Severity of a Reported Bug", In Proc. of IEEE Working Conference on Mining Software Repositories, pp. 1-10, 2010.
[23] Y. Tian, D. Lo, and C. Sun, "Information Retrieval based Nearest Neighbor Classification for Fine-grained Bug Severity Prediction", In Proc. of IEEE Working Conference on Reverse Engineering, pp. 215-224, 2012.
[24] T. Menzies, and A. Marcus, "Automated Severity Assessment of Software Defect Reports", In Proc. of IEEE International Conference on Software Maintenance, pp. 346-355, 2008.
[25] J. Arokiam, and J. S. Bradbury, "Automatically Predicting Bug Severity Early in the Development Process", In Proc. of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results, pp. 17-20, 2020.
[26] N. K. S. Roy, and B. Rossi, "Towards an Improvement of Bug Severity Classification", In Proc. of IEEE EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 269-276, 2014.
[27] M. Sharma, P. Bedi, K. K. Chaturvedi, and V. B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation", In Proc. of International Conference on Intelligent Systems Design and Applications (ISDA), pp. 539-545, 2012.
[28] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic Prediction of The Severity of Bugs Using Stack Traces and Categorical Features", In Journal of Information and Software Technology, Vol. 123, 106205, 2020.
[29] Q. Umer, H. Liu, and Y. Sultan, "Emotion based Automated Priority Prediction for Bug Reports", In Proc. of IEEE Access, Vol. 6, pp. 35743-35752, 2018.
[30] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep Neural Network-based Severity Prediction of Bug Reports", In Proc. of IEEE Access, Vol. 7, pp. 46846-46857, 2019.
[31] A. Kao, and S. R. Poteet, "Natural Language Processing and Text Mining", in Springer Science & Business Media, 2007.
[32] Statistical Hypothesis Testing, Wikipedia, https://en.wikipedia.org/wiki/Statistical_hypothesis_testing, (accessed 30 May 2022).
[33] Shapiro–Wilk test, Wikipedia, https://en.wikipedia.org/wiki/Shapiro-Wilk_test, (accessed 30 May 2022).
[34] T-statistic, Wikipedia, https://en.wikipedia.org/wiki/T-statistic, (accessed 30 May 2022).
[35] Wilcoxon signed-rank test, Wikipedia, https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test, (accessed 30 May 2022).

FIRST A. JungYeon Kim is an Undergraduate Student in the Department of Computer Science and Engineering at Kyungnam University, South Korea from 2019 to the present. She is currently a Researcher in the Artificial Intelligence Software Lab. in Kyungnam University, South Korea from 2021 to the Present. Her research interests are Artificial Intelligence, Big Data, and Software Engineering.

SECOND B. Geunseok Yang is an Assistant Professor in the Department of Computer Science and Engineering, Kyungnam University, South Korea from 2021 to the present. He obtained his Ph.D. and M.Eng. from the Department of Computer Science, University of Seoul, South Korea in 2020 and 2015, respectively, and his bachelor degree from the Department of Computer Science and Engineering, Korea University of Technology and Education, South Korea in 2013. His research interests are AI-based Software Engineering and Software Mining.