applied sciences

MDPI

Article

A Bug Triage Technique Using Developer-Based Feature Selection and CNN-LSTM Algorithm

Jeongmin Jang and Geunseok Yang *

Department of Computer Science and Engineering, Kyungnam University, Changwon 51767, Korea
* Correspondence: gsyang@kyungnam.ac.kr

Abstract: With an increase in the use of software, the incidence of bugs and resulting maintenance costs also increase. In open source projects, developer reassignment accounts for approximately 50%. Software maintenance costs can be reduced if appropriate developers are recommended to resolve bugs. In this study, features are extracted by applying feature selection for each developer. These features are entered into CNN-LSTM algorithm to learn the model and recommend appropriate developers. To compare the performance of the proposed model, open source projects (Google Chrome, Mozilla Core, and Mozilla Firefox) were used to compare the performance of the proposed method with a baseline for developer recommendation. In this paper, the performance showed 54% for F-measure and 52% for accuracy in open source projects. The proposed model has improved and showed about a 13% more effective performance improvement than with DeepTriage. It was discovered that the performance of the proposed model was better.

Keywords: CNN-LSTM; feature selection; developer recommendation; bug triage; software evolution

Citation: Jang, J.; Yang, G. A Bug Triage Technique Using Developer-Based Feature Selection and CNN-LSTM Algorithm. *Appl. Sci.* **2022**, *12*, 9358. https://doi.org/10.3390/app12189358

Academic Editors: Dusica Marijan and Dimitris Mourtzis

Received: 5 August 2022
Accepted: 13 September 2022
Published: 18 September 2022

1. Introduction

Recently, software is being increasingly used in various fields and by many users. Functional improvements and unforeseen bugs occur in response to user demands, and the submission of bug reports is increasing continually. When a bug occurs, developers must correct it to increase software quality, but in the Eclipse and Mozilla projects, about 50% of the bug reports are reallocated to other developers because they cannot correct it [1]. If an appropriate developer is recommended for a bug, the software maintenance cost can be reduced.

The following are some related studies on bug triage: Hu et al. [2] proposed a developer recommendation method that uses a data center bridging network, through which they categorized a list of developers who could correct bugs. Cubranic et al. [3] proposed developer recommendations using text classification and machine-learning techniques. Jeong et al. [4] analyzed the bug tossing history using Markov chains and recommended appropriate developers. However, past studies required verification of models using various projects and data, and their developer recommendation performance must also be improved.

To resolve this problem, this study utilizes the convolutional long short-term memory neural networks (CNN-LSTM) algorithm [5] and the feature selection algorithm [6] to recommend appropriate developers. It classifies bug reports based on developers and the feature selection algorithm is applied to the classified bug reports to extract their features. The extracted features are then applied to the convolutional neural network (CNN) algorithm, and its output is input into the long short-term memory (LSTM) algorithm. Finally, the result of the LSTM algorithm identifies the recommended developer.

To evaluate the model of the proposed method, we compared that the proposed method provided better recommendations in open source projects (Google Chromium [7], Mozilla Core [8], Mozilla Firefox [8]).

In this paper, we conducted research using the following research questions. RQ1: Is the model proposed by developer recommendation appropriate? RQ2: Does the model proposed by the developer recommendation perform better than the baseline? These questions first check whether we recommend developers well and compare baseline to verify the developer recommendation performance. In addition, it is compared whether there is a significant difference through statistical verification [9,10].

The contribution of this paper is as follows.

- A feature selection algorithm for each developer was applied to extract their characteristics. When extracting features for each developer, the words in the top K were adjusted, and the optimal K was identified.
- The extracted results were applied to the CNN-LSTM algorithm, and the developer was recommended. The extracted features were input into the CNN algorithm, the output of which were then input into the LSTM. Finally, the result of the LSTM indicated the recommended developers. It has also been shown that CNN-LSTM models make better predictions than individual CNN or LSTM models.
- The proposed algorithm learned the model by extracting features for each developer from bug reports. There is a difference in confirming that the performance of extracting characteristics for each developer has been improved more than simply learning bug reports.
- The performances of the baseline and developer recommendation algorithms were compared for Google Chrome, Mozilla Core, and Mozilla Firefox open source projects. The proposed model showed approximately 13% performance improvement over DeepTriage.

We provide a brief of the summary as follows. In open-source projects, about 50% of the bugs are reassigned after they are assigned to developers. Therefore, the cost of software maintenance increases and good quality software is not provided quickly. In order to resolve this problem, this paper recommended an appropriate developer by extracting characteristics for each developer. In details, the features are extracted by selecting features for each developer. The extracted features are input to the CNN-LSTM algorithm to train the model and recommend appropriate developers. To compare the performance of the proposed model, open source projects (Google Chrome, Mozilla Core, Mozilla Firefox) were compared with the baseline (DeepTriage). This result showed an approximate 13% performance improvement over DeepTriage. Thus, if bug reports are analyzed based on the developer and appropriate developers are recommended, bugs can be accurately corrected, and the software quality can be improved.

The structure of this paper is as follows. The introduction is shown in Section 1, and related work is shown in Section 2. Section 3 introduces background knowledge, and Sections 4 and 5 introduce the proposed approach and experiments. The discussion of the experiment results is shown in Section 6, and the conclusion is shown in Section 7.

2. Related Work

Bhattacharya et al. [11] applied title, description, keywords, product, component, and developer activity information to the naive Bayes algorithm. The data used were obtained from Eclipse (306,297) and Mozilla (549,962), all of which showed approximately 77% performance in Rank#5.

Tamrawi et al. [12] used title and description information and extracted features to create a model. The data used were obtained from Eclipse (69,829), and their method showed 68.00% performance in Rank#5.

Anvik et al. [13] applied title and description information to naive Bayes, EM, support vector machine (SVM), C4.5, and K-nearest neighbor algorithms. The data used were obtained from Eclipse (7233) and Firefox (7596), and appropriate developer recommendations were made.

Xuan et al. [14] applied title and description information to naive Bayes and SVM algorithms and extracted developer priority information to create a model. The data

used were obtained from Eclipse (49,762) and Mozilla (30,609), and their model exhibited performances of 53.10% and 56.98% in Rank#5, respectively.

Shokripour et al. [15] applied bug location prediction and developer expertise using title and description information, extracted weighted unigram noun terms, and created a model for recommending developers. The data used were obtained from JDT (85) and Firefox (80), and their model exhibited performances of 89.41% and 44.46% in Rank#5, respectively.

Wang et al. [16] used title and description information, applied an active developer cache, and created a recommendation model. The data used were obtained from Eclipse (177,937) and Mozilla (69,195), and their model exhibited performances of 84.45% and 55.56% in Rank#5, respectively.

Yang et al. [1] proposed a new method that uses linear discriminant analysis (LDA). Features, such as natural language, severity, components, priorities, and products, were used. They created a set of developers who contribute to a bug report with the same combination of topics and functions, as well as graded and ranked scores using bug allocations, annotations, developer commitments, and attachments.

Nguyen et al. [17] used bug fixes and the severity level of historical bug reports as inputs for a log-normal regression model to predict the time required to solve bugs with topics inferred from LDA. Based on the average bug resolution time of each developer, appropriate developers are recommended who could resolve bugs the fastest according to the most important topic in the open bug report.

Zhao et al. [18] combined a vector space model with a subject model for classification data. A term frequency-inverse document frequency vectorizer was used to create a vector space model. LDA was used to create subject models, and machine learning, SVM, and neural network algorithms were used for classification tasks. SVMs have been demonstrated to perform better than neural networks.

Naguib et al. [19] proposed an approach using developer activities that included the modification, review, and developer assignment of LDA and bug reports. In addition to the title and description of bug reports, it uses system components to categorize bug reports as items. Additionally, a list containing activities for each developer can be created and used in conjunction with their connection to the topic to recommend suitable developers.

Xia et al. [20] proposed a multi-functional topic model that models the topic distribution for the feature combination of bug reports to extend the underlying topic modeling algorithm LDA. The functional combination of bug reports considers products and components as additional observed variables. This approach recommends developers based on feature combinations and preference scores for topics.

Alenezi et al. [21] extracted categorical features and metadata for feature extraction of bug reports with text properties. They calculated the gain ratio to determine the essential features that provided a normalized scale for each feature that contributed to the classification. Using text and categorical data together is better than using only the test data because using only one categorical feature lowered the classification performance.

Mani et al. [22] proposed a bidirectional RNN-based technique for automating bug classification. They used word2vec embedding techniques and attention mechanisms to learn the syntax and semantic features of long word sequences to vectorize the text. This method is superior to conventional machine-learning methods.

Guo et al. [23] proposed a CNN-based method that included developer activities. They used word2vec embeddings for vectorization, created real world scenarios using segmentation verification, and validated their methods using large datasets obtained from open source projects.

Zhang et al. [24] used a submitter-bug-commenter heterogeneous network to improve the quality of the developer recommendations. Using this network, submitters and commenters can obtain propagated topic information from bugs, which can adjust the subject distribution obtained by the LDA model.

Yadav et al. [25] proposed a technique that uses bug classification expertise to rank developers. They reduced the length of the bug transmission and constructed profiles of

the developers. Developer expertise scores were generated by leveraging indexed metrics, priority-weighted fixed problems, and bug resolution times. Component-based, cosine, and Jaccard similarities were determined to calculate professionalism scores. This method recommends an appropriate list of developers based on their professional scores.

Kumari et al. [26] solved the bug classification problem using a bug dependency-based mathematical model. Bug dependence is caused by design flaws, improper coding, misunderstandings between users and developers, and coding errors. The entropy is calculated from the summary and explanatory notes of the bug report, and the developer's mission is determined by the entropy.

Almhana et al. [27] proposed an automated bug classification method that extracts files to be examined for each bug report and considers the dependencies between them. It uses multipurpose searches based on dependencies on other reports and priorities to rank the bug reports of programmers. This method reduced the positioning time of bugs by more than 30%, compared to conventional bug prioritization techniques.

Jahanshahi et al. [28] proposed a dependency-recognition bug classification method, which is different from the dependency-based method. It allocated bugs appropriately using NLP and integer programming. Their method used integrated textual information, dependencies between bugs, and costs associated with each bug. This technology improves the time required to solve bugs and reduces the number of delayed bugs. However, they assumed that each developer could work on only one report at a time.

Alenezi et al. [29] created a model for assigning developers to new bug reports using the naive Bayes classifier. They used five term selection methods in feature selectors, log odds ratio, chi-square, mutual information, and term frequency, to extract discriminative terms when extracting features to learn predictive models.

Wang et al. [30] proposed RCNNs that learn the features of bug reports and recommend developers. This algorithm uses CNN to extract information in the text and RNN to extract sequence information in the text. However, in this study, the model is differentiated from this study by learning the model using feature extraction for each developer.

Xi et al. [31] proposed a model using text content, metadata, and tossing sequence information of the bug report. We propose ITRIGY to learn the features of textual content and the tossing sequence and incorporate the features.

3. Background Knowledge

Bugs are an inevitable part of software development, and developers must read bug reports and perform debugging to correct them. To this end, a project manager reads the bug report and recommends an appropriate developer. The developer recommendation process for open source projects is shown in Figure 1.

When a user discovers a bug or a functional issue while using a program (a), a bug report is created (b). Once a user submits a bug report, it is stored (c) in the corresponding software bug repository. The project manager reads the bug report passively (d) and allocates the appropriate developer to solve the bug (e). However, the developer reallocation in open source projects (Eclipse [32] and Mozilla [8]) occurs for approximately 50% of the bugs. Additionally, the bug report can be written by a general user, developer, quality assurance officer, etc. Therefore, the subjective method of expressing a bug may be different owing to differences in their understanding of the bug. Project managers who read bug reports can also assign developers based on their subjective judgments.

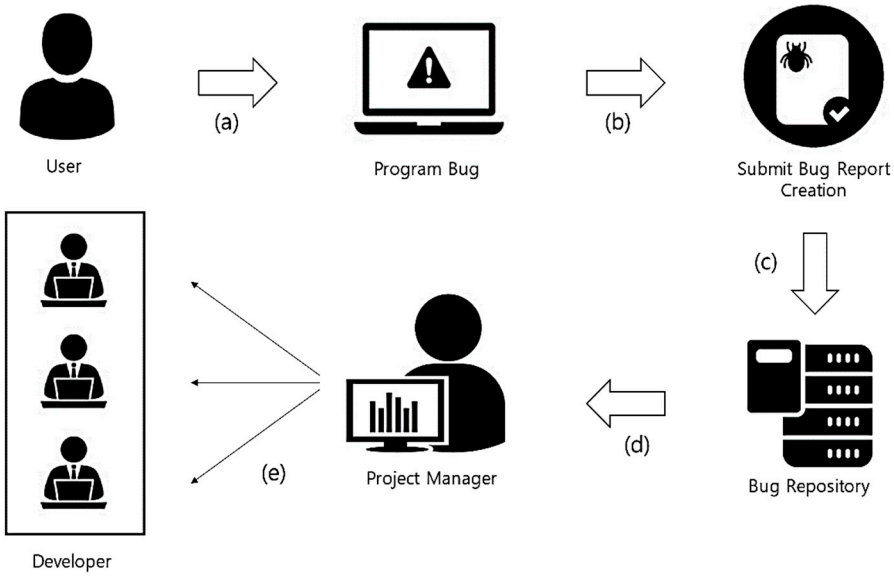

Figure 1. Developer Recommendation in Open Source Projects.

Through an automated bug developer recommendation method, these problems can be resolved, and software quality can be improved. This study describes bug reports and bug tracking management systems to understand developer recommendations.

3.1. Bug Report

In general, users who discover bugs and functional improvements freely express their findings in the text format. An example of a Mozilla Firefox bug report is presented in Figure 2. This bug report is a Mozilla Firefox #82301 [33] and was submitted on 22 May 2001. The bug reporter is *mpt*, the bug allocation is *mak*, and the developer is *magicxinzhang*. The status of the bug report is *RESOLVED* or *FIXED*, and the bug has been corrected and is *CLOSED*.

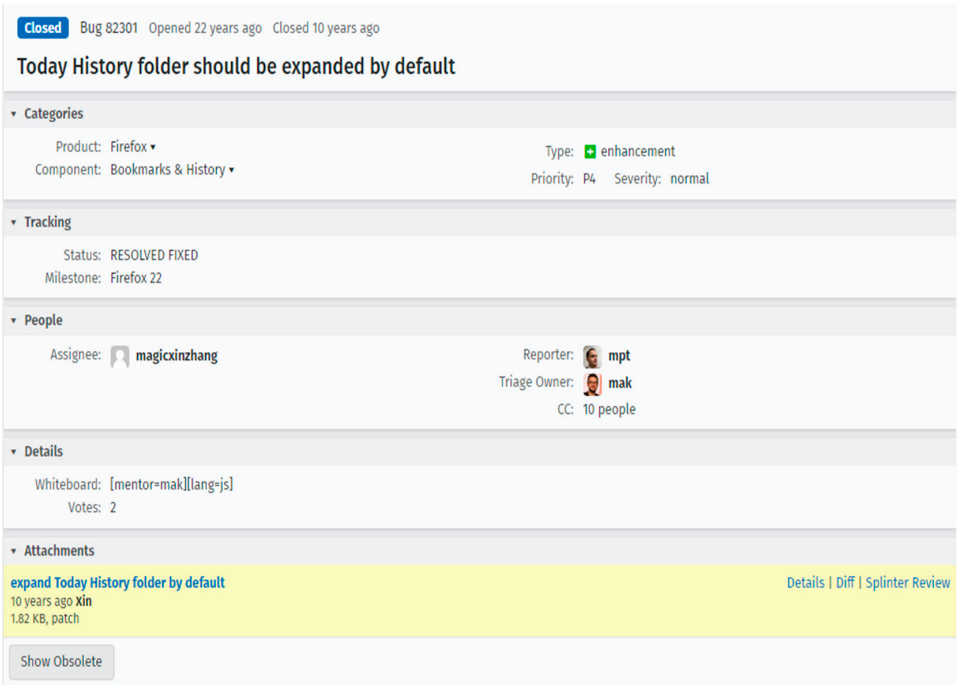

Figure 2. An example of Mozilla Firefox bug report.

The report for Mozilla Firefox was submitted on 13 March 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*.

3.2. Bug Tracking System

For software quality management, a bug tracking system [34] must be implemented, and various bug tracking systems exist. In these systems, bugs can be distinguished using various types of information, and their history management is also possible. A bug tracking system includes several functions. Bug tracking is a function that can be verified by writing the name of the developer who compiles a list of bugs and fixes them. Issue tracking is a function similar to bug tracking, which creates a list of the functions to be added or tasks to be performed and puts the name on the list of developers they want to develop. A ticket is a function that allows them to identify the extent to which a function has been developed. A wiki is a type of function used to write shared documents on the web, which can be modified by someone else. If a bug tracking system is not employed, bugs must be managed manually and software quality management costs may increase. The Eclipse and Mozilla open source projects use Bugzilla's bug tracking system, whereas large-scale systems primarily use bug tracking systems.

4. Proposed Approach

In this study, we recommend developers to use the CNN-LSTM and feature selection algorithms. The overall schematic of the proposed method is shown in Figure 3.

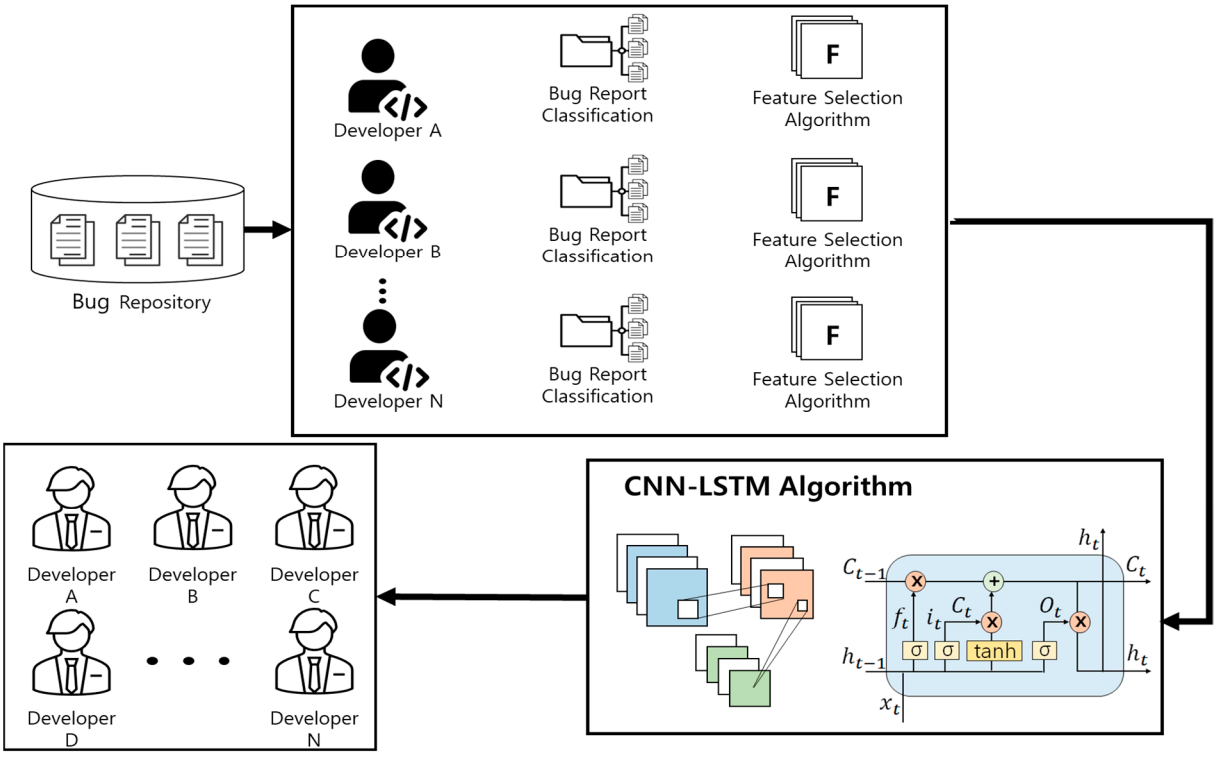

Figure 3. Schematic of the proposed method.

First, bug reports are extracted from the bug repository and classified based on the developer. Feature selection algorithms are applied to the classified bug reports, and developer-specific features are extracted. The extracted features are then input into the CNN algorithm, and its output is input to the LSTM algorithm. Finally, the LSTM output returns the recommended developer.

4.1. Preprocessing

The bug report is composed of words expressed in sentences. Preprocessing is performed during natural language processing (NLP) [35] to process the sentences. First, word tokenization, lemmatization, and stop word removal are performed on the bug reports. Using this method, sentences can be classified into words and circular extractions of root words and stop words can be removed. For example, for the sentence *"what steps will reproduce the problem1 open or reload the bookmark manager2 notice that a folder is selected in the lefthand view3 click organize and try to select add folder or add page add folder add page are greyed out if you explicitly click on a folder they then become active this applies not just do the default bookmark folder but also when reloading a page with a specific folder id in the url eg click on some other folder then hit r eload that folder is still selected but you cant use organize"*, when the preprocessing process is performed on this sentence, "step", "select", "grey", "apply", "reload", and "select" are extracted, and "steps", "selected", "greyed", "applies", "reloading" and "selected" are extracted as "step", "select", "grey", "apply", "reload", and "select", respectively.

4.2. Feature Selection Algorithm

Feature selection [6], which is the process of selecting features to construct a model, is a crucial technology in both deep and machine learning. An overall schematic of the feature selection algorithm is shown in Figure 4.

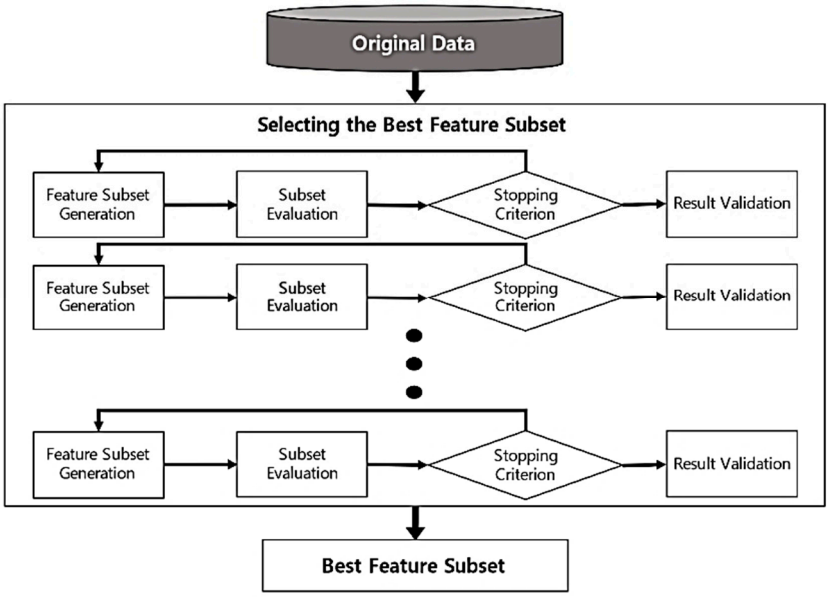

Figure 4. Overall schematic of the feature selection algorithm.

First, learning is performed for a feature extraction model to extract the features of the text. Grid search and document frequency are used to reduce the vector size of the models to extract features at a high rate. Generally, when the volume of documents increases, the computational speed of feature extraction decreases. In this study, we used the grid search and document frequency during feature extraction. During feature extraction, a subset of features is generated to calculate the association between features. The subset is then evaluated, and the feature selection model is repeatedly validated. Using a feature selection ensemble, the best combination is determined, and the features of the bug report are extracted. We provide the details of developer-based feature selection example as shown in Figure 5 [22].

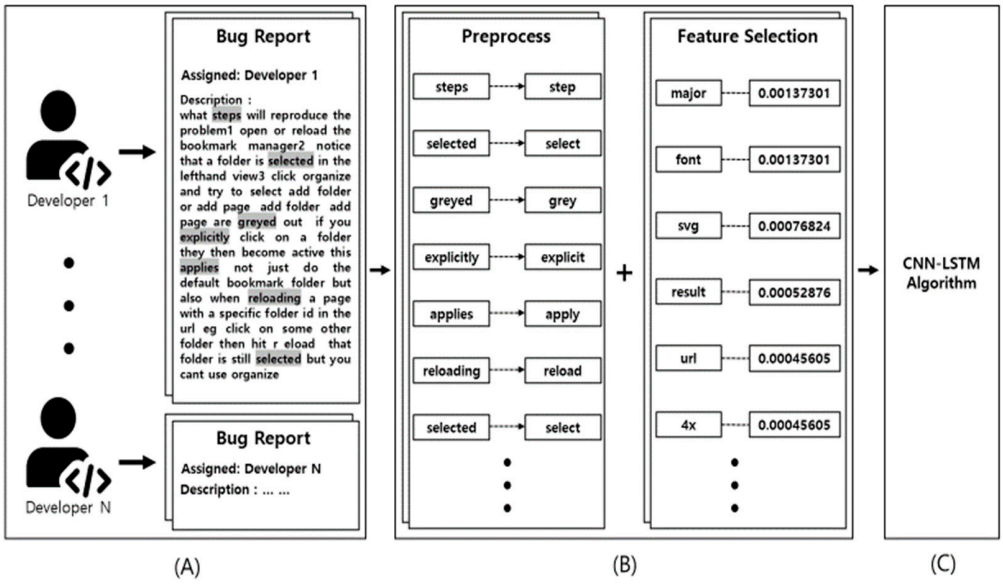

Figure 5. An example of developer-based feature selection.

In this paper, we get connectivity by classifying bug reports from developers (A). Then, we perform the preprocessing, including tokenization of sentences, removal of stop words, and extraction of root words. In detail, the tokenization is that the sentence is divided into words, and in the stop word removal, unnecessary words are removed to reduce the dimension of the word dictionary. In extraction of root words, the word is converted into a root of word. We put bug reports for each developer as input to the feature selection algorithm. In the feature selection algorithm, top k refers to the features with the top k scores with high scores (B). For example, assuming k = 3, feature extraction for each developer is performed up to the feature with the third highest score. The extracted features are combined with developer-based bug reports and put as input to the CNN algorithm. Finally, the developer is recommended by putting the output of the CNN algorithm as the input of the LSTM algorithm (C).

4.3. CNN-LSTM Algorithm

The features of the developer-specific bug report were extracted and applied to the CNN-LSTM algorithm. First, the feature is input into the CNN algorithm, and its output is input into the LSTM algorithm. Finally, the output of the LSTM algorithm is the recommended developer. An overall schematic of the CNN-LSTM algorithm is shown in Figure 6.

The CNN is a deep learning algorithm that is used for speech recognition [36] and NLP [36]. The algorithm can learn directly from data, analyze data, find patterns, and learn from neural network models that include convolution preprocessing. Convolution refers to the inversion and transition of one of the two functions, and the integration of the result multiplied by the other function. CNN [5] may comprise multiple layers. Filters, which are also called kernels, consist of square matrices. In the convolution layer, the size of the feature map becomes smaller than that of the input data, owing to the actions of the filter and stride. Convolution padding prevents the shrinking of the output data of a layer. Padding refers to filling the input data with a specific value, using a specific pixel on the outer surface, and, in this study, it is usually filled with a padded value of 0. The pooling layer receives the output data from the convolution layer as input and is used to reduce the size of the output data or emphasize specific data.

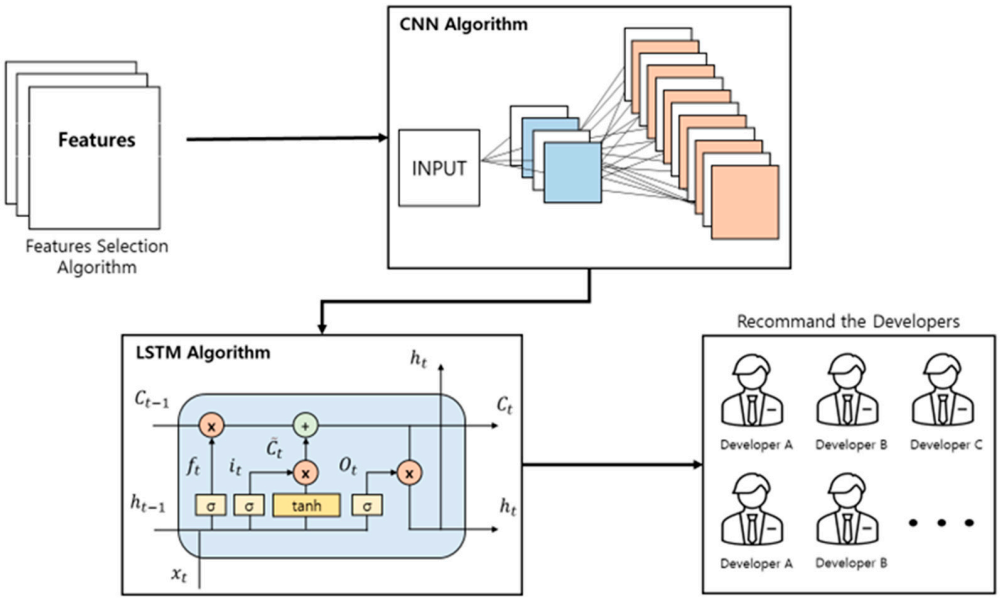

Figure 6. Overall schematic diagram of the CNN-LSTM algorithm.

LSTM [5] is a type of recurrent neural network (RNN) that compensates for the RNN shortcomings of referring only to the previous information and is called a short-term memory that also remembers previous data. LSTM adds an input gate, a forget gate, and an output gate to the memory of the hidden layer and determines and stores the data to be remembered. Its result is the recommended developer.

5. Experimental Result

The following steps were performed to conduct the experiment in this study. First, bug reports were collected and stored in a bug repository. Preprocessing was then performed on the stored bug reports. The bug reports were then classified based on the developer and the classified bug reports were applied to the feature selection algorithm. The applications of the bug report contents were extracted by dividing it into developers using the CNN-LSTM algorithm. Finally, the appropriate developers were recommended.

5.1. Our Dataset

We used this dataset [22] in this study, which is shown in Table 1. Google Chrome, Mozilla Firefox, and Mozilla Core open source projects were used, and the total number of bugs was approximately 860,000. The dataset used in this paper has more than 1000 developers (up to 1944). Therefore, the model is learned and evaluated by applying the following four conditions (M = 0, 5, 10, 20) according to the minimum number of bug reports assigned to each developer. For example, 5 means that the minimum number of bug reports assigned to each developer is 5 or more times, and 10 means that the minimum number of bug reports is assigned to each developer is 10 or more times.

Table 1. Our dataset.

	Google Chromium	Mozilla Firefox	Mozilla Core
# of Reports	383,104	314,388	162,307

The equipment used in the experiment included the Intel Core i9-12900 CPU, 64 GB of RAM, and NVIDIA GeForce RTX 3090 GPU. In the model, the hyperparameter is *following embedding = 200, conv1 = 128, max_pooling = 2, lstm = 512, learning_rate* $= 1 \times 10^{-4}$. We provide the model of summary as follows.

Input_1 (Input_Layer)
Conv1d_1 (Conv1D)
Max_pooling1d_1 (MaxPooling1D)
lstm_1 (LSTM)
dense_1 (Dense)
flatten_1 (Flatten)
activation_1 (Activation)
repeat_vector_1 (RepeatVector)
permute_1 (Permute)
multiply_1 (Multiply)
lamda_2 (Lambda)
concatenate_1 (Concatenate)
batch_normaliztion_1 (BatchNor)
dropout_1 (Dropout)

5.2. Evaluation Metrics

The following was used to evaluate the performance of the proposed model and baseline. If the bug report recommended 320,000 developers, and the developer recommended by the model would be accurate, it would provide a total of 383,104. In this case, the accuracy is 83.53%. In this study, the model was evaluated using precision [1], recall [1], F-measure [1], and accuracy [22], and these evaluation metrics are mainly used in machine learning and data mining.

$$Dev\_Acc = \frac{Dev\_TP + Dev\_TN}{Dev\_TP + Dev\_FN + Dev\_FP + Dev\_TN}$$

$$Dev\_Pre = \frac{Dev\_TP}{Dev\_TP + Dev\_FP}$$

$$Dev\_Rec = \frac{Dev\_TP}{Dev\_TP + Dev\_FN}$$

$$Dev\_F = \frac{2 \times Dev\_Pre \times Dev\_Rec}{Dev\_Pre + Dev\_Rec}$$

We note that Dev_Acc represents the accuracy of the developer recommendation. Dev_Pre represents the rate at which the prediction of true is actually true. Dev_Rec represents that is predicted to be true among the actual true. Dev_F represents the true measure of how well the predicted result predicted in relation to the correct answer. Moreover, Dev_TP means that the rate at which the prediction of true is actually true. Dev_TN means that the rate at which the prediction of false is true. Dev_FP means that the rate at which the prediction of true is false. Dev_FN means that the rate at which the prediction of false is actually false.

This paper evaluated developer performance based on Top K and compared it with baseline based on Top 1, Top 5, Top 10, and Top 20. For example, in the case of Top 1, one developer was recommended, and Top 10 recommended 10 developers.

5.3. Baseline

An experiment was conducted by comparing the proposed model with a baseline to evaluate its efficiency. The baseline used in this study was DeepTriage [22] and related products were disclosed, thereby allowing for various comparative analyses. DeepTriage [22] learns features using the deep bidirectional recurrent neural network with attention (DBRNN-A) algorithms and recommends appropriate developers.

5.4. Research Questions

To compare the performance of the proposed model and baseline, the following research questions were considered.

RQ1: Is the developer recommendation performance of the proposed model appropriate?

This question verifies the performance before making comparisons with the baseline. It was confirmed that the developer recommendations are appropriate.

RQ2: Does the proposed model perform better than the baseline?

This question confirms whether the proposed model performs better than the baseline and compares whether there exist statistically significant differences between them. For statistical verification, the Wilcoxon test [9] and *t*-test [10] were performed, and it was observed that there was a significant difference between the proposed method and baseline.

5.5. Results

5.5.1. Result of Our Approach

The performance of our model is shown in Figure 7. We note that the X-axis represents the evaluation metrics for open source projects (Google Chrome, Mozilla Core, and Mozilla Firefox) and the Y-axis represents a score value. Overall, the accuracy is more than approximately 52.4%, and appears to recommend developers well. In this experiment, the best prediction was made when K = 1, and this value was used in the feature selection algorithm for comparison with the baseline.

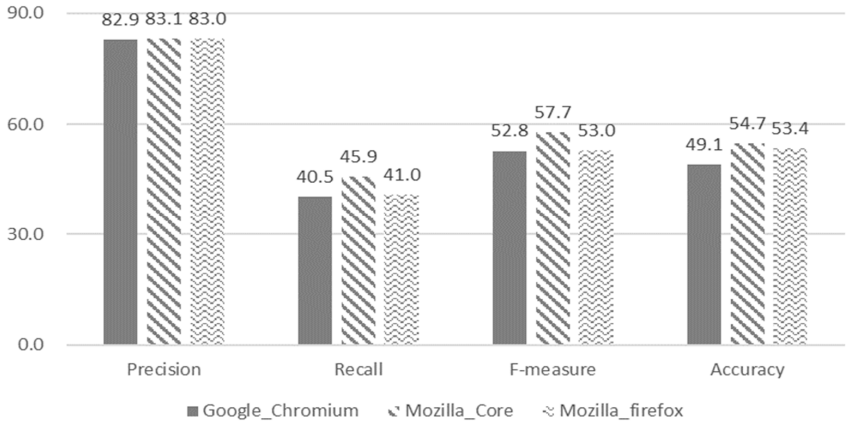

Figure 7. Performance of our approach.

Performance comparisons between evaluation metrics for each project are shown in Figures 8–11. The X-axis representing the parameter (M = 0, 5, 10, 20) is the minimum number of bug reports assigned to each developer and the Y-axis represents a performance. We note that the scores of our model are 82.6% (precision), 42% (recall), 54% (F-Measure), and 52% (accuracy) in Figures 8–11, respectively.

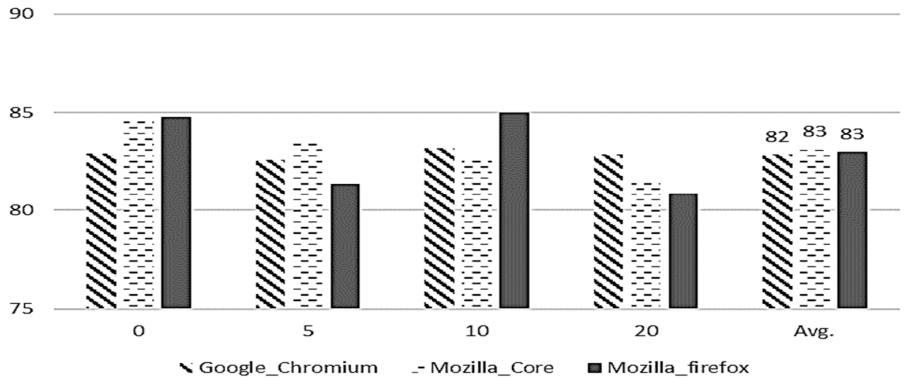

Figure 8. Precision result between for each project.

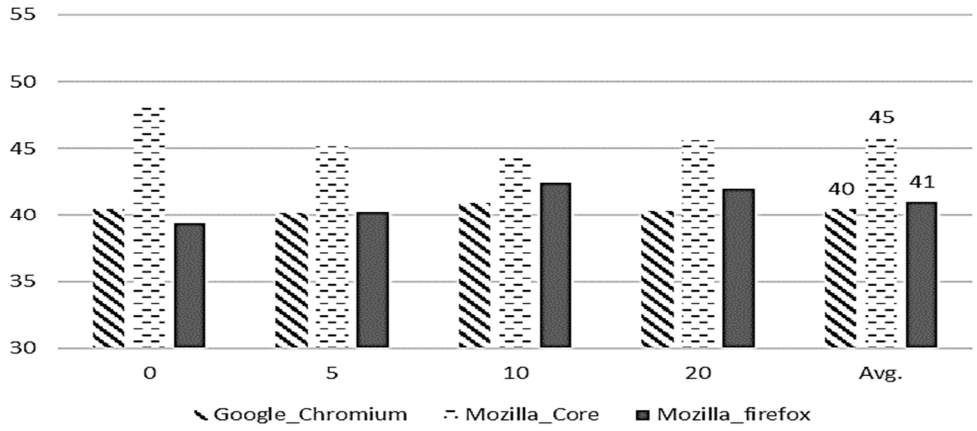

Figure 9. Recall result between for each project.

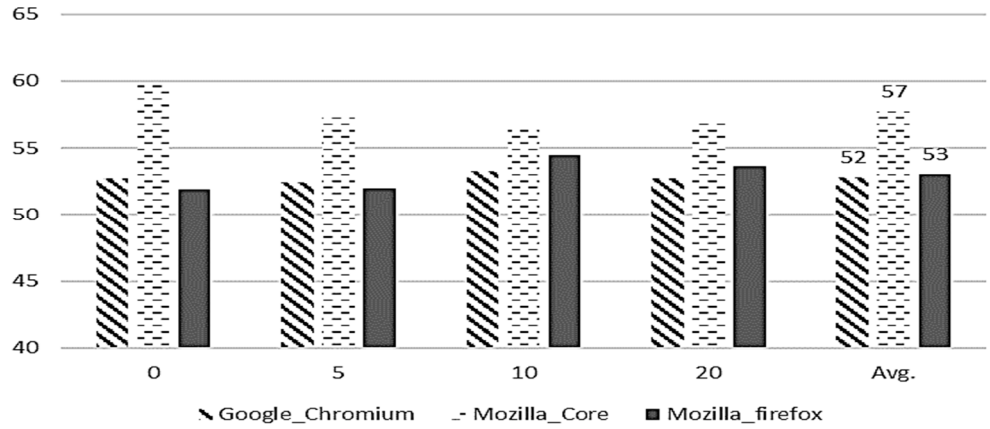

Figure 10. F-measure result between for each project.

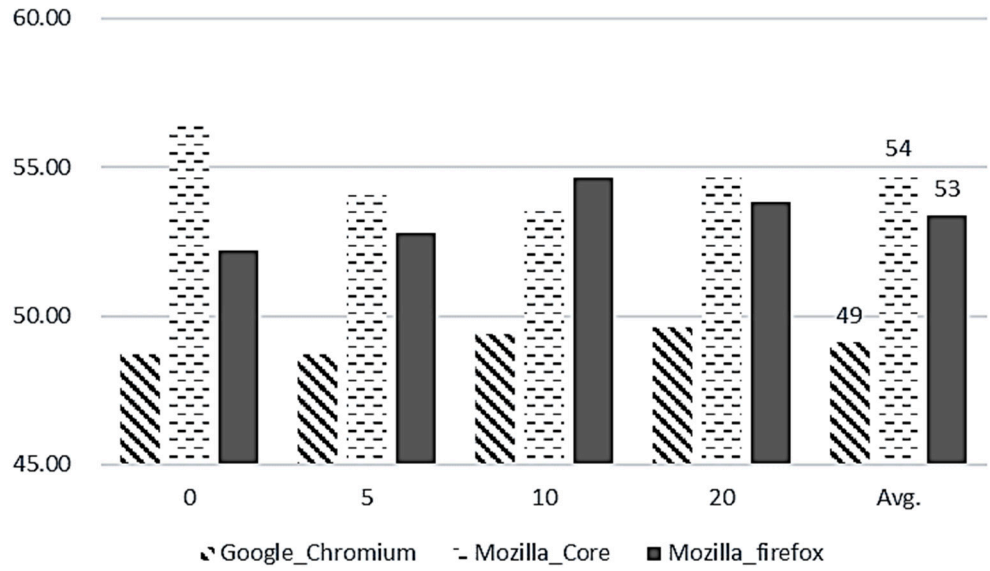

Figure 11. Accuracy result between for each project.

Additionally, we compared our model with non-algorithm, and the results are shown in Figure 12. The X-axis represents the projects (Google Chromium, Mozilla Core, Mozilla Firefox), and the Y-axes indicates the accuracies. The figure indicates that the proposed method is better at recommending developers than the non-algorithm. We note that the proposed model has improved, which is about 43% more performance than non-algorithm.

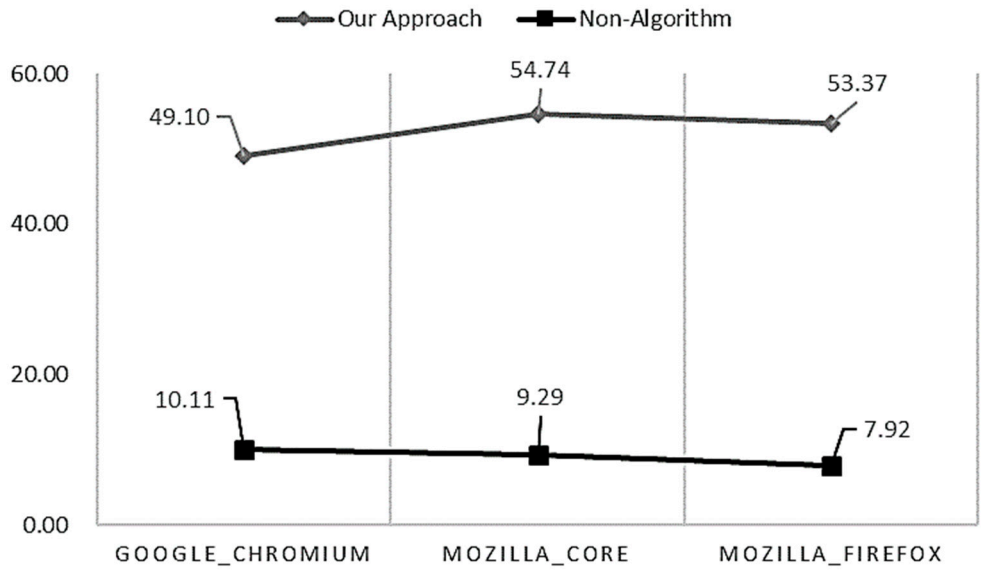

Figure 12. Performance comparison of our approach and non-algorithm.

5.5.2. Comparison Results

The performance comparison between the proposed model and the baseline is shown in Figure 13. The X-axes indicate the baseline for the open source projects (Google Chromium, Mozilla Core, Mozilla Firefox), and the Y-axes indicate the accuracies. The figure indicates that the proposed method is better at recommending developers than the baseline. We note that the proposed model has improved, which is about 13% better performance than with DeepTriage. Moreover, they indicate that the difference in the performance of the model proposed through statistical verification is significant, compared with the baseline.

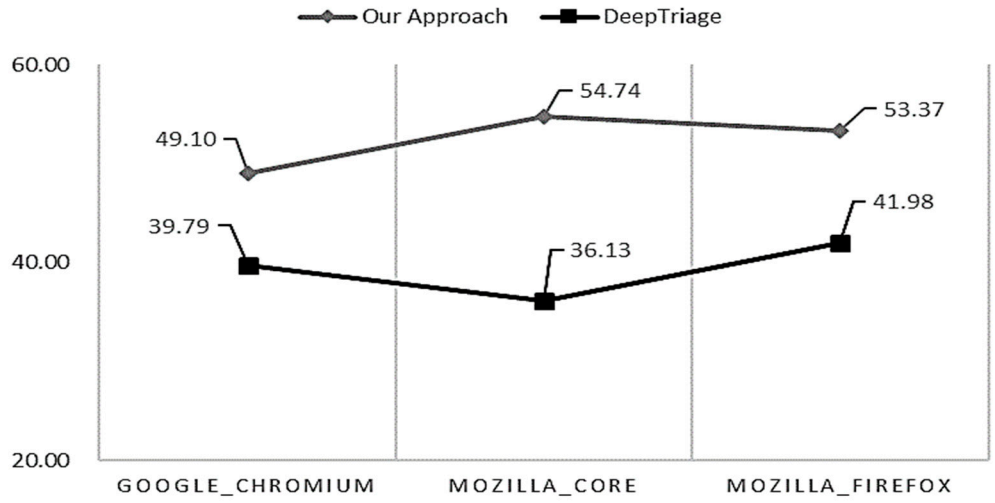

Figure 13. Performance comparison between baselines.

The null hypothesis is set as follows.

$H1_0$: The proposed method is not significantly different from DeepTriage for Google Chromium.

$H2_0$: The proposed method is not significantly different from DeepTriage for Mozilla Core.

$H3_0$: The proposed method is not significantly different from DeepTriage for Mozilla Firefox.

For the set null hypothesis, the alternative hypothesis is set as follows.

$H1_a$: The proposed method differs significantly from DeepTriage for Google Chrome.

$H2_a$: The proposed method differs significantly from DeepTriage for Mozilla Core.

$H3_a$: The proposed method differs significantly from DeepTriage for Mozilla Firefox.

The normal distribution [37] was checked for each hypothesis to proceed with the statistical verification. If the normal distribution was more than 0.05, the *t*-test was performed, and if it was less than 0.05, the Wilcoxon test is performed. The statistical verification results of the proposed model and baseline are listed in Table 2.

Table 2. Statistical verification results.

Hypothesis	*p*-Value	Result
$H1_0$	(Wilcoxon test) 4.88×10^{-2}	$H1_a$: Accept
$H2_0$	(Wilcoxon test) 3.91×10^{-3}	$H2_a$: Accept
$H3_0$	(Wilcoxon test) 2.73×10^{-2}	$H3_a$: Accept

For example, $H1_0$ indicates that our model is not significantly different from DeepTriage for Google Chromium. However, the *p*-value of 4.88×10^{-2} (0.04883) is smaller than 0.05; therefore, we accept $H1_a$. As a result, there is a difference between our model and DeepTriage for Google Chromium. Additionally, we rejected all null hypotheses and accepted all alternative hypotheses.

6. Discussion

6.1. Experiment Results

The proposed model was compared with the baseline called DeepTriage, where it showed better recommendation performance and there was a statistically significant difference between the groups. The proposed method uses the CNN-LSTM and feature selection algorithms, which exhibit the following differences:

- Feature selection for each developer was conducted, and the CNN-LSTM model was learned using the upper K value in the feature selection algorithm. While adjusting the K value, we identified the optimal developer recommendation performance, which was better than that of DeepTriage. K = 1 was set for the proposed method and a developer recommendation was made. If K was 2 or more, the performance gradually decreased, compared to K = 1. When this was checked, it was discovered that the larger the K value, the larger the embedding size of the word, thereby resulting in a gap in the set value and improper learning. In the future, the impact of the K value will be investigated in depth to confirm the association.
- The performance was further improved by applying feature selection to each developer. It is noteworthy that there exist words characterized by each developer and that the distribution of the words mainly exist for certain developers.

6.2. Threats and Validity

In this study, we used Google Chrome, Mozilla Firefox, and Mozilla Core open source projects, and approximately 860,000 data points. However, the entire dataset from Google Chrome, Mozilla Firefox, and Mozilla Core open source projects, and the method proposed using some data does not always perform well. Additionally, the performance for other open source projects must be verified. In the future, this model will be verified using various datasets.

The proposed model adjusts the upper feature words by adjusting the K value in the feature selection algorithm. However, there is no guarantee that the set K value will always perform well. In the future, K values will be adjusted using various datasets. Moreover, the CNN-LSTM algorithm uses the basic hyperparameters of CNN and LSTM, and the model can be improved by identifying the optimal hyperparameters in the future.

The proposed model uses an open source project, and further verification of its applicability to business projects is required. This is because bug management and data structures in open source and business projects are different. We plan to verify this further in the

future. The main purpose of this study was to recommend appropriate developers by learning past bug data. However, recommending new developers that are not included in the learning data was not the goal of this study. In this regard, through extended research, the research will be expanded to a structure that can recommend new developers.

7. Conclusions

During the process of recommending developers for bug correction, a phenomenon occurred wherein the assigned developers could not correct the bug, which were then reallocated to other developers. Through automated developer recommendation algorithms, software quality can be improved, and maintenance costs can be reduced. To address these problems, our method recommends appropriate developers through the CNN-LSTM and feature selection algorithms. First, bug reports for each developer are classified and the characteristics of each developer are extracted. The extracted features are then applied to the CNN-LSTM algorithm. The input of the CNN is a developer-specific feature, and its output is input into the LSTM. Finally, the LSTM output returns the recommended developer list. To evaluate the proposed model, we compared its performance by using DeepTriage as a baseline and demonstrated that the proposed method is better at providing recommendations. In detail, the model has F-measure of 54% and an accuracy of 52% in open source projects. The proposed model showed about 13% performance improvement over DeepTriage. We further verified that the differences were statistically significant. In the future, we plan to expand this research by utilizing automatic parameter adjustments and various other characteristics. In addition, in this paper, it was shown that the extraction of features by developers affects the recommendation of developers. Moreover, the research will be expanded by checking the correlation of the bug report between developers by topic.

Author Contributions: G.Y. organized and advised the paper, and J.J. wrote the paper and conducted experiments. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: For the dataset in this paper, the following published paper [22] is used.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yang, G.; Zhang, T.; Lee, B. Towards Semi-automatic Bug Triage and Severity Prediction based on Topic Model and Multi-feature of Bug Reports. In Proceedings of the International Computer Software and Applications Conference, Vasteras, Sweden, 21–25 July 2014; pp. 97–106.
2. Hu, H.; Zhang, H.; Xuan, J.; Sun, W. Effective Bug Triage based on Historical Bug-fix Information. In Proceedings of the In-ternational Symposium on Software Reliability Engineering, Naples, Italy, 3–6 November 2014; pp. 122–132.
3. Cubranic, D.; Murphy, G.C. Automatic Bug Triage using Text Categorization. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, Banff, AB, Canada, 20–24 June 2004; pp. 92–97.
4. Jeong, G.; Kim, S.; Zimmermann, T. Improving Bug Triage with Bug Tossing Graphs. In Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Athens, Greece, 19–28 August 2009; pp. 111–120.
5. She, X.; Zhang, D. Text Classification based on Hybrid CNN-LSTM Hybrid Model. In Proceedings of the International Symposium on Computational Intelligence and Design, Hangzhou, China, 8–9 December 2018; Volume 2, pp. 185–189.
6. Ashokkumar, P.; Shankar, S.G.; Srivastava, G.; Maddikunta, P.K.; Gadekallu, T.R. A Two-stage Text Feature Selection Algorithm for Improving Text Classification. ACM Trans. *Asian Low-Resour. Lang. Inf. Process.* **2021**, *20*, 1–19.
7. Google Chromium. Available online: https://bugs.chromium.org/p/chromium/issues/list (accessed on 28 May 2022).
8. Mozilla. Available online: https://bugzilla.mozilla.org/home (accessed on 28 May 2022).
9. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80–83. [CrossRef]
10. The T-Test, Research Methods Knowledge Base. Available online: http://www.socialresearchmethods.net/kb/stat_t.php (accessed on 28 May 2022).

11. Bhattacharya, P.; Neamtiu, I. Fine-grained Incremental Learning and Multi-feature Tossing Graphs to Improve Bug Triaging. In Proceedings of the International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; pp. 1–10.
12. Tamrawi, A.; Nguyen, T.T.; Al-Kofahi, J.; Nguyen, T.N. Fuzzy Set-based Automatic Bug Triaging (NIER track). In Proceedings of the International Conference on Software Engineering, Honolulu, HI, USA, 21–28 May 2011; pp. 884–887.
13. Anvik, J.; Murphy, G.C. Reducing the Effort of Bug Report Triage: Recommenders for Development-oriented Decisions. *ACM Trans. Softw. Eng. Methodol.* **2011**, *20*, 1–35. [CrossRef]
14. Xuan, J.; Jiang, H.; Ren, Z.; Zou, W. Developer Prioritization in Bug Repositories. In Proceedings of the International Con-ference on Software Engineering, Zurich, Switzerland, 2–9 June 2012; pp. 25–35.
15. Shokripour, R.; Anvik, J.; Kasirun, Z.M.; Zamani, S. Why So Complicated? Simple Term Filtering and Weighting for Location-based Bug Report Assignment Recommendation. In Proceedings of the Working Conference on Mining Software Re-positories, San Francisco, CA, USA, 18–19 May 2013; pp. 2–11.
16. Wang, S.; Zhang, W.; Wang, Q. FixerCache: Unsupervised Caching Active Developers for Diverse Bug Triage. In Proceedings of the the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 18–19 September 2014; pp. 1–10.
17. Nguyen, T.T.; Nguyen, A.T.; Nguyen, T.N. Topic-based, Time-aware Bug Assignment. In Proceedings of the ACM SIGSOFT Software Engineering Notes, Hong Kong, China, 16–21 November 2014; pp. 1–4.
18. Zhao, Y.; He, T.; Chen, Z. A Unified Framework for Bug Report Assignment. *J. Softw. Eng. Knowl. Eng.* **2019**, *29*, 607–628. [CrossRef]
19. Naguib, H.; Narayan, N.; Brügge, B.; Helal, D. Bug Report Assignee Recommendation Using Activity Profiles. In Proceedings of the Working Conference on Mining Software Repositories, San Francisco, CA, USA, 18–19 May 2013; pp. 22–30.
20. Xia, X.; Lo, D.; Ding, Y.; Al-Kofahi, J.M.; Nguyen, T.N.; Wang, X. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Trans. Softw. Eng.* **2016**, *43*, 272–297. [CrossRef]
21. Alenezi, M.; Banitaan, S.; Zarour, M. Using Categorical Features in Mining Bug Tracking Systems to Assign Bug Reports. *arXiv* **2018**, arXiv:1804.07803. [CrossRef]
22. Mani, S.; Sankaran, A.; Aralikatte, R. DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, Kolkata, India, 3–5 January 2019; pp. 171–179.
23. Guo, S.; Zhang, X.; Yang, X.; Chen, R.; Guo, C.; Li, H.; Li, T. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. *Neural Process. Lett.* **2020**, *51*, 2589–2606. [CrossRef]
24. Zhang, W.; Cui, Y.; Yoshida, T. En-LDA: A Novel Approach to Automatic Bug Report Assignment with Entropy Optimized Latent Dirichlet Allocation. *Entropy* **2017**, *19*, 173. [CrossRef]
25. Yadav, A.; Singh, S.K.; Suri, J.S. Ranking of Software Developers Based on Expertise Score for Bug Triaging. *Inf. Softw. Technol.* **2019**, *112*, 1–17. [CrossRef]
26. Kumari, M.; Misra, A.; Misra, S.; Sanz, L.F.; Damasevicius, R.; Singh, V.B. Quantitative Quality Evaluation of Software Products by Con- sidering Summary and Comments Entropy of a Reported Bug. *Entropy* **2019**, *21*, 91. [CrossRef]
27. Almhana, R.; Kessentini, M. Considering Dependencies between Bug Reports to Improve Bugs Triage. *Autom. Softw. Eng.* **2021**, *28*, 1–26. [CrossRef]
28. Jahanshahi, H.; Chhabra, K.; Cevik, M.; Baþar, A. DABT: A Dependency-aware Bug Triaging Method. In Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 21–23 June 2021; pp. 221–230.
29. Alenezi, M.; Magel, K.; Banitaan, S. Efficient Bug Triaging Using Text Mining. *J. Softw.* **2013**, *8*, 2185–2191. [CrossRef]
30. Wang, H.; Li, Q. Effective Bug Triage Based on a Hybrid Neural Network. In Proceedings of the 28th Asia-Pacific Software Engineering Conference, Taipei, Taiwan, 6–9 December 2021; pp. 82–91.
31. Xi, S.; Yao, Y.; Xiao, X.; Xu, F.; Lv, J. Bug triaging based on tossing sequence modeling. *J. Comput. Sci. Technol.* **2019**, *34*, 942–956. [CrossRef]
32. Eclipse. Available online: https://bugs.eclipse.org/bugs (accessed on 28 May 2022).
33. Mozilla Bug #82301. Available online: https://bugzilla.mozilla.org/show_bug.cgi?id=82301 (accessed on 28 May 2022).
34. Zimmermann, T.; Premraj, R.; Sillito, J.; Breu, S. Improving Bug Tracking Systems. In Proceedings of the International Con-ference on Software Engineering-Companion Volume, Vancouver, BC, Canada, 16–24 May 2009; pp. 247–250.
35. Wilson, A.A. Natural Language Processing: Machine Learning and Modeling Linguistics in the Domain of Sentiment Analysis. Masters Thesis, University of Nottingham, Nottingham, UK, 2020.
36. Kamath, U.; Liu, J.; Whitaker, J. *Deep Learning for NLP and Speech Recognition*; Springer: Cham, Switzerland, 2019; Volume 84.
37. Shapiro–Wilk Test. WIKIPEDIA. Available online: http://en.wikipedia.org/wiki/Shapiro-Wilk_test (accessed on 28 May 2022).