Applying Deep Learning Based Automatic Bug Triager to Industrial Projects

Sun-Ro Lee
Min-Jae Heo
Chan-Gun Lee*
Chung-Ang University
Seoul, Republic of Korea
ssunno,elfel,cglee@cau.ac.kr

Milhan Kim
Gaeul Jeong
Software Engineering R&D Laboratory
LG Electronics
Seoul, Republic of Korea
milhan.kim,gaeul.jeong@lge.com

ABSTRACT

Finding the appropriate developer for a bug report, so called 'Bug Triage', is one of the bottlenecks in the bug resolution process. To address this problem, many approaches have proposed various automatic bug triage techniques in recent studies. We argue that most previous studies focused on open source projects only and did not consider deep learning techniques. In this paper, we propose to use Convolutional Neural Network and word embedding to build an automatic bug triager. The results of the experiments applied to both industrial and open source projects reveal benefits of the automatic approach and suggest co-operation of human and automatic triagers. Our experience in integrating and operating the proposed system in an industrial development environment is also reported.

CCS CONCEPTS

• Software and its engineering → Software maintenance tools;

KEYWORDS

automatic bug triage, convolutional neural network, text classification, industrial project

ACM Reference format:
Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying Deep Learning Based Automatic Bug Triager to Industrial Projects. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 6 pages.
https://doi.org/10.1145/3106237.3117776

1 INTRODUCTION

The software development process has to face the software bugs inevitably. While software bugs need to be resolved for the quality of the software, managing them in a timely manner comes with some cost. To handle them systematically, many organizations often adopt issue tracking systems such as Mozilla BugZilla [6] and Atlassian JIRA [3]. Even though the issue tracking systems are found helpful in managing bugs efficiently, still, many tasks for resolving bugs demand considerable amount of resources. Finding the appropriate developer for an issue, so called 'Bug Triage', is one of such tasks. It becomes even harder if the project involves a large number of developers. Bug triage has been one of the important factors contributing to high cost of resolving bugs [2]. Especially in commercial software projects, it is considered critical since delays in resolving bugs may result in reschedules of release date and losses of business opportunities.

To solve this problem, automatic bug triage techniques have been proposed in recent studies [2, 7]. In particular, many related studies have proposed approaches based on machine learning and they build classifiers by analyzing patterns of previously resolved bug reports. Upon the arrival of a new report, they use the classifiers to recommend developers suitable to fix the given bug. The success of such techniques typically relies on the classification performance of machine learning and natural language processing techniques.

In this study, we report our experience of applying an automatic bug triager based on a deep learning to industrial projects. Specifically, our bug triager uses Convolutional Neural Network (CNN) for machine learning and Word2Vec [11] for word embedding. Those decisions are made due to the following reasons.

Firstly, it is frequently reported that deep learning tends to perform better than traditional machine learning in many problems such as computer vision, automatic speech recognition and natural language processing. Moreover, a deep learning algorithm itself is a good feature extractor, hence, it does not require sophisticated feature extraction techniques [10]. This advantage is very appealing because bug reports tend to have complicated sub-structures and contain versatile types of information. For example, a bug report may contain source code snippet, stack traces, and log data, which may require techniques for advanced feature selections to conventional machine learners. Our preliminary experiments with Naïve Bayes, Support Vector Machine(SVM), C.48, and CNN using a few open-source projects confirmed superior performance of CNN over those traditional machine learning algorithms; hence, we have decided to build a CNN-based bug triager.

Secondly, we transform each bug report into a set of vector representations using the word embedding technique. The considered industrial projects include bug reports written in multiple languages due to its international collaboration nature. Without

*Corresponding author

proper handling of this characteristics, machine learners may experience difficulties in recognizing identical concepts expressed in different languages. During the construction of a word embedding model from the industrial projects, the words with similar semantics in different languages tend to be mapped to similar vector representations.

The main contributions of this paper are summarized as follows:

- We propose an automatic bug triage system based on deep learning (CNN) and word embedding (Word2Vec) techniques. We have seen deep learning is applied to a few problems in software engineering; however, our study is the first attempt to apply it to the bug triage problem to the best of our knowledge.
- The performance of the proposed system is compared with that of human triager to measure the benefit in terms of both accuracy and time overhead. Based on the issue tracking logs for the industrial projects, we simulate the situations where the proposed system is applied and validate the effectiveness of the proposed system.
- We present the architecture of the bug triage system and how to integrate it with our development framework. We report our maintenance experience in industrial environment, too.

2 RELATED WORK

2.1 Automatic Bug Triage

Several works have been done to build automatic bug triagers using machine learning algorithms. Anvik et al. [2] proposed a bug triager using SVM. Ahsan et al. [1] proposed a way to improve the performance of the machine learning classifiers by preprocessing bug reports using TF-IDF and Latent Semantic Indexing(LSI). Jeong et al. [7] proposed a tossing graph model that analyzes the bug reallocation history among developers based on the Markov chain. The tossing graphs represent developer relationships that deal with associative developers or similar topics, and can improve the performance of machine learning classifiers.

However, most former studies focused on open source projects only but rarely attempted in industrial environments. We shall summarize some representatives of those work in the industrial context in the following. Jonsson et al. [9] proposed a bug triager with a machine learning ensemble and measured performance on five industrial projects. To improve the performance of automatic bug triage, they ensemble known machine learning algorithms such as Naïve Bayes, SVM, and Decision Trees. Dedík et al. [4] compared various combinations of machine learning algorithms and natural language processing techniques to build automatic bug triagers for both open source projects and industrial projects. They found that SVM, TF-IDF, and stop words removal were the best combination.

2.2 Text Mining using Deep Learning

Deep learning has become popular thanks to its superb performance on image processing applications. On the other hand, deep learning also rendered itself effective in text mining such as text categorization, machine translation, and semantic analysis. One of recent study proposed to analyze sentiments in sentences using CNN [5]. They performed learning of word-level embeddings using the Word2Vec to capture syntactic and semantic information in the sentences. Johnson et al. [8] proposed a model combining Long Short-Term Memory (LSTM) and CNN. Since LSTM can learn sequence information, it can construct a learning model reflecting not only the semantics of words but also their appearance order.

3 CNN BUG TRIAGER

3.1 Linguistic Challenges and Preprocessing

This section introduces a preprocessing step required to train bug reports effectively in the proposed model. We describe challenges found in our industrial projects and suggests a preprocessing method to address them.

The first challenge is multi-lingual bug reports. Multinational companies based on non-English speaking countries often use their native language and English together in their work. It is common that documents in such companies are mixed with different languages. For example, LG Electronics, a South Korea based global company with multiple R&D sites all over the world, have repositories of bug reports written in multiple languages. In the bug reports, there may be words written in different languages but have the same or similar meaning which we should consider as synonyms. Unless properly handled, it can degrade the performance of learning due to increase of dependent features in the training data.

The second challenge is use of jargons. Industrial projects often use jargons referring to the elements of a particular product or process steps. Beware that meaning of these jargons are usually different from the original. But they are important terms in the project. Thus, the bug triager should recognize and handle them properly.

To address these two challenges, we adopt Word2Vec [11] to convert each word to a vector form. During the configuration of Word2Vec using the company's bug reports, each word appearing in the document can be represented as a vector representation conveying its semantics. Due to this characteristics, the similar words with different forms can be treated similarly. Also the jargons can be handled properly because Word2Vec deduces latent meaning of word(s) from the training data.

3.2 CNN Model for Assigning Bug Reports

This section introduces the training process of our CNN based bug triager which will be referred to as CNN triager in what follows. Figure 1 shows the structure of the proposed deep learning model. Before learning, Word2Vec converts the each words into vectors in order to organize bug reports into training data. The training data is in the form of a matrix and each row corresponds to the vector representation of each word contained in the bug report. The length of the column is equal to the length of a vector, which is generated by Word2Vec.

Firstly, in the convolution layer, the feature is extracted by applying the convolution filter c^j to the training data where j represents a window size. Each filter extracts a feature from j words at a time. When the length of training data is n, the following feature map $\mathbb{F}$ is obtained from c^j.

$$\mathbb{F}^j = \{f_1, f_2, \ldots, f_{n-j+1}\}$$

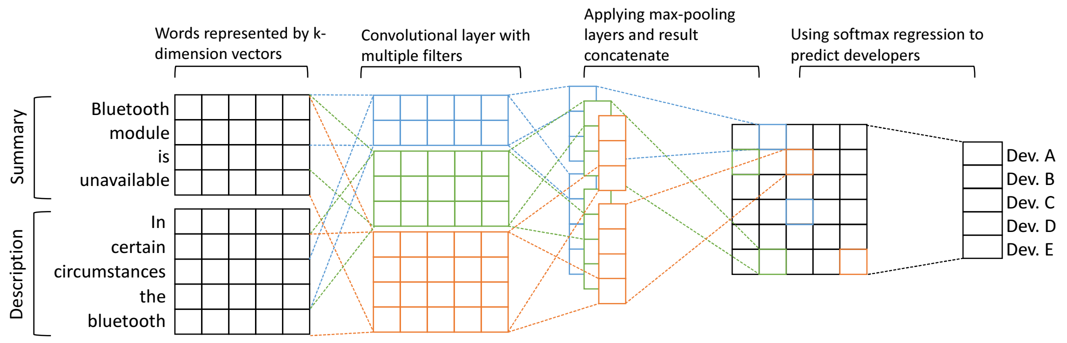
Words represented by k-dimension vectors
Convolutional layer with multiple filters
Applying max-pooling layers and result concatenate
Using softmax regression to predict developers
Summary
Bluetooth
module
is
unavailable
Description
In
certain
circumstances
the
bluetooth
Dev. A
Dev. B
Dev. C
Dev. D
Dev. E

Figure 1: Architecture CNN model for bug assignment

In $\mathbb{F}^j$, f_l means l-th feature extracted using c^j. For all $\mathbb{F}$, a non-linear function Rectified Linear Unit(ReLU) is applied for back propagation. ReLU is a technique applied to logistic regression of features in neural networks. In the extracted $\mathbb{F}$, the subsampling process *max-pooling* is applied, which is a simple operation to select the value of *max* in $\mathbb{F}$. This procedure is performed in each convolution filter. To obtain various features, we apply multiple filters and use different filter sizes to obtain diverse $\mathbb{F}$ from the training data. When the number of filters is h, *pooled* $\mathbb{F}$ is obtained by *max-pooling*.

$$pooled\ \mathbb{F} = \{max(\mathbb{F}_1^j), max(\mathbb{F}_2^j), \ldots, max(\mathbb{F}_h^j)\}$$

We apply this process to two fields, Summary and Description, of the bug report, respectively; then, we obtain two *pooled* $\mathbb{F}$ and concatenate them. Finally, through the fully-connected layer and softmax regression, we calculate the probability that each developer is appropriate for the given bug report.

In the proposed CNN model, *dropout* and *l2_regularization* are applied to prevent overfitting. The overfitting happens when a neural network is customized too much to the training data. In such a case, the neural network may exhibit poor performance on new data. *dropout* is a technique for 'forgetting' some of the information learned by the neural network. The *l2_regularization* is used for preventing the model from distortion by imposing penalties to outliers.

4 EVALUATION

In this section, we evaluate the performance of the implemented CNN triager. Specifically, we aim to answer the following research questions:

- RQ1: How well does CNN triager predict developers?
- RQ2: How much is the benefit of automating bug triage?

4.1 Data Collection

For the experiments we collected bug reports of which states are RESOLVED from four industrial projects (anonymized into A, B, C, and D) and three open source projects. The project information such as the number of developers, the number of reports, the collection period is shown in Table 1. All the bug reports from the industrial projects were written by Quality Assurance (QA) team members. We regard the developers who fixed more than 10 bug reports as "active". In the literature, many approaches [1, 2, 4] reported the effects of filtering non-active developers. All the industrial projects have adopted JIRA to manage bug reports and recorded each event of tossing a bug report along with detailed information such as its assignee and timestamp.

Table 1: Summary of the projects used for the experiment

Project	Period	Reports	Dev.(Active)
A	2015-08-15~2016-12-31	1,257	65(18)
B	2015-08-13~2016-12-31	6,256	165(76)
C	2014-10-13~2016-12-31	14,583	82(67)
D	2015-03-13~2016-12-31	3,152	22(12)
JDT	2013-10-20~2016-10-20	1,465	70(15)
Platform	2013-08-01~2016-07-31	4,825	225(55)
Firefox	2013-08-01~2016-07-31	13,667	848(142)

4.2 Performance Validation(RQ1)

4.2.1 Evaluation Measure. We evaluate the performance of the proposed model by measuring the top-K accuracy. In this evaluation method, performance is measured according to the degree to which the recommendations from the implemented CNN triager includes the correct developer. Notice that all information in the recommendation list as well as the first one is useful. In case the first candidate is not a correct answer, then we can re-assign the bug report to the next probable developers shown in the list. Since the CNN triager computes the appropriateness score for each developer, we sort the results and generate ranked list of recommended developers.

The top-K accuracy is calculated as:

$$top-K\ accuracy = \frac{1}{|Q|}\sum_{i=1}^{Q} assess(a_i, \mathbb{R}_i^K) \tag{1}$$

$$assess(a_i, \mathbb{R}_i^k) = \begin{cases} 1 & \text{if } a_i \in \mathbb{R}_i^k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In Equation (1), Q indicates the set of bug reports. a_i represents the correct developer, i.e. ground-truth, for the i-th bug report. $\mathbb{R}_i^k$ means the ranked list of which size is k recommended by the CNN triager for the ith bug report. In Equation (2), $assess(a_i, \mathbb{R}_i^k)$ returns 1 when $\mathbb{R}_i^k$ covers a_i and 0 otherwise.

Table 2: Performance of CNN triager and human triager

Project	Human Triagers			CNN with All Developers					CNN with Active Developers				
	Acc_1	Acc_2	Acc_3	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5
A	41.9%	72.8%	84.2%	61.2%	70.1%	75.3%	77.6%	79.2%	73.1%	85.8%	90.2%	92.2%	94.5%
B	25.9%	64.0%	82.1%	57.8%	68.5%	73.5%	76.6%	78.9%	58.1%	69.3%	74.3%	78.0%	80.7%
C	35.4%	71.9%	87.9%	85.2%	90.2%	92.5%	93.6%	94.6%	85.6%	90.1%	92.2%	93.3%	94.2%
D	78.8%	93.5%	98.9%	71.2%	86.5%	91.6%	94.1%	94.5%	72.9%	87.1%	93.0%	95.0%	96.6%
JDT	N/A	N/A	N/A	46.6%	61.1%	68.1%	71.7%	74.3%	51.6%	68.5%	76.9%	82.6%	86.1%
Platform	N/A	N/A	N/A	36.1%	45.8%	50.5%	53.7%	56.7%	42.7%	52.8%	58.3%	63.0%	65.8%
Firefox	N/A	N/A	N/A	27.1%	36.7%	42.8%	47.1%	50.5%	30.5%	42.0%	48.4%	53.5%	57.1%

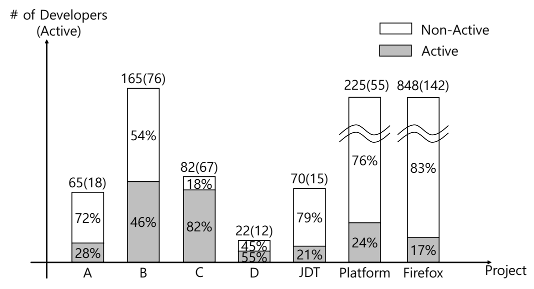

Figure 2: Percentage of active developers in each project

4.2.2 Experimental results of top-K accuracy. Table 2 shows the top-K accuracy evaluation result of the CNN triager for each project. All the experiments were performed with 10-fold cross validations. It also presents the performance of human triagers, which is calculated from the issue tracking logs for industrial projects. "CNN with Active Developers" indicates the cases where the candidates are restricted to the active developers only. Figure 2 shows the percentage of active developers in each project.

When the candidates span all the developers, all the industrial projects with the CNN triager achieved higher classification accuracy than open source projects. The industrial projects show up to 82.83% and 35.83% higher performance than open source in top-1 and top-3 accuracy, respectively.

When the candidates are restricted to active developers only, all the industrial projects achieved higher accuracy than open source projects except the project B, which showed lower performance than the JDT in top-3 and top-5 accuracy. It should be pointed out that the developer group in the project B is much larger than that in JDT.

Notice the performance difference within each project when the candidates span all the developers versus when restricted to active developers only. The top-1 accuracy of the projects A and JDT improved by 19.44% and 10.73%, respectively. Also, top-1 accuracy of the projects Platform and Firefox improved by 18.28% and 12.55%, respectively. However, for the projects B, C, and D, the performance improvements are less than 3%.

We found that the latter three projects showing smaller improvements had a higher percentage of active developers than the others. It can be observed that filtering of non-active developers helps to boost the accuracy better in case percentage of active developers are low. In addition, the industrial projects tend to have higher percentages of active developers than the open source projects, which can be understood by their characteristics.

Now let us compare the top-1 accuracy of the proposed approach with the human triager in industrial projects. The CNN triager performs better than the human triager except the project D, where the number of candidate developers are smaller than the others as shown in Figure 2. It is observed that the human triager is good at picking a correct developer in a relatively small group, however, the automatic triager can do better in larger groups.

4.3 Effectiveness Validation(RQ2)

In this section, we measure the benefits of automating bug triage tasks through adopting the CNN triager. More specifically, the benefit is captured by the savings in the triage overhead of time due to the automatic bug triaging.

The baseline is set to the case of human triager and we can calculate its triage overhead from the logs of our issue tracking system. We could derive the tossing counts and the time duration between the tossings for each bug report whose status is RESOLVED.

For the automatic case, we estimate the triage overhead by considering the logs and top-K accuracy of the CNN triager.

In this experiment we define that the triage overhead for a bug report is a summation of all the tossing costs until its resolution. The k-th tossing cost t^k is composed of *review cost* and *forward cost* as illustrated in Figure 3.

The review cost indicates the time duration spent by the assigned developer to check whether the assignment is correct or not. The developer needs to review the assigned bug report and decide to accept it or not. If the assignment was not correct, then the developer will forward the bug report to another developer or the triager to re-assign. The time duration until the next tossing happens is represented by forward cost.

Figure 4 shows the human triager's median of k-th tossing cost in industrial projects. The reason why we use a median not an average is that we found that some bug reports were managed externally from the issue tracking system. Those bug reports contain incorrect tossing timestamps due to batched clean-ups. They increase the average significantly because their assignment periods are typically more than a few dozen days, which cannot be true in our company. Notice that the tossing cost t^0 does not include review cost. In our scheme, each bug report is initially delivered to the triager and he does not need to check whether the assignment was correct.

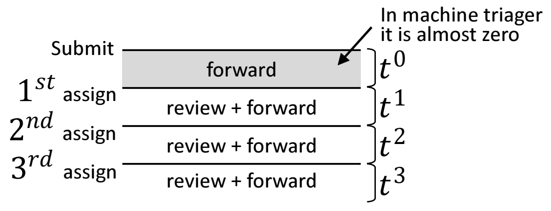

Figure 3: Estimation of triage overhead

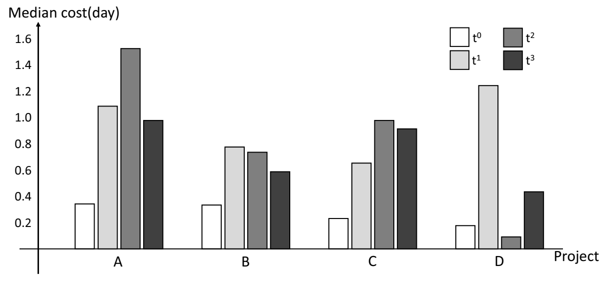

Figure 4: Median cost of k-th tossing

Table 3: Expected triage overhead

Project	Human Triager	CNN(5)	CNN(4)	CNN(3)	CNN(2)	CNN(1)
A	1x	1.2x	1.07x	0.93x	0.80x	0.67x
B	1x	1.2x	1.08x	0.95x	0.82x	0.71x
C	1x	0.4x	0.37x	0.33x	0.29x	0.26x
D	1x	1.53x	1.40x	1.29x	1.14x	0.97x

The expected triage overhead of the CNN triager $E(C)$ is derived as follows:

$$E(C) \leq \sum \hat{t}^k \cdot f^k \qquad (3)$$

where f^k is the probability of failing to pick a correct assignee in the k-th tossing. $\hat{t}^k$ is an approximation of the k-th tossing cost t^k for the CNN triager. In our model, t^k for the CNN triager can be computed by deducting the forward cost from the k-th tossing cost for the human triager. Unfortunately, the issue tracking logs do not allow us to separate the k-th tossing cost into review cost and forward cost. Hence, we use approximation $\hat{t}^k$ which is a median of all the tossing costs in the issue tracking logs for the project where $k > 0$. $\hat{t}^0$ is set to zero because CNN triager can decide an assignee within a few seconds. Beware that this approximation produces an upper bound to the CNN triager because each tossing cost t^k in the logs includes forward cost as well as review cost; in fact, forward cost should be almost zero for the CNN triager. Hence, $E(C)$ is less than or equal to the upper bound.

Table 3 shows the simulation results of the proposed technique. By setting the original triage overheads to 100%, each trial shows the changed triage overhead. Each column named with CNN(k) indicates the situation where top-k recommendations from the CNN triager were used for the assignments before the k-th tossing, and the remaining tossings are handled by the human triager. For example, CNN(2) indicates the case where the first and the second

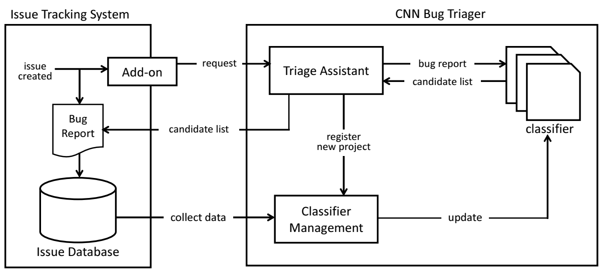

Figure 5: Architecture of CNN triager and its integration

recommendations from the CNN triager were used for the first and the second assignments, respectively. Notice that the second assignment happens only when the first assignment turns out incorrect. The human triager takes care of the rest. The reason for this construction is two-fold. One is that top-k accuracy for the CNN triager does not reach 100% where $k \leq 5$ for any projects in our experiment. In addition, we find that almost no bug reports experienced more than six tossings in issue tracking logs.

Hence, we constructed the experimental cases where the CNN and human triagers work together by adjusting the maximum number of automatic tossings for each report. For the remaining steps where the human triager is adopted, we used the top-K accuracy and the triage overhead of the human triager approximated from the issue tracking logs. We believe that this also reflects the realistic scenarios where humans intervene when necessary.

It turns out that CNN(1) is the best where we utilize the top-1 recommendation only from the CNN triager. This is due to the fact that top-1 accuracy of the CNN triager is higher than or comparable to top-1 accuracy of human triager. In addition, the triage overhead is zero for the first assignment. It was also observed that the assignment accuracy of human triager tends to increase faster than that of the CNN-triager typically after a few tossings.

Notice that the CNN triager becomes more valuable when the developer group is large, which is represented by the project B in our experiment. In contrast, the human triager alone can be sufficient when the group is relatively small such as the project D.

5 AUTOMATIC BUG TRIAGE IN INDUSTRIAL ENVIRONMENT

In this section, we present how the CNN triager is integrated with our issue tracking system.

5.1 Using CNN Triager as Assistant

As we have presented in Section 4, the proposed approach can aid the human triager effectively in our industrial projects. Hence, we decided to use the CNN triager as an assistant for human triager in industrial development environment. Figure 5 shows the architecture of a bug fixer recommendation system implemented in LG Electronics' issue tracking system. When a new bug report is created, the CNN triager computes five most appropriate candidates using its pre-trained CNN model and appends the ranked recommendation list as additional information of the bug report. Later, the human triager utilizes this information as a hint during the bug assignment.

5.2 Maintenance for CNN Triager

Without re-training of the learned model with the latest bug resolutions in a timely manner, the model is doomed to become obsolete. An obsolete CNN triager may produce incorrect recommendations for new bug reports. The daily average numbers of bug reports for the projects A, B, C, and D were 5.90, 37.17, 35.59, and 19.51, respectively. The daily maximums were 59, 189, 412, and 70, respectively.

As shown in Figure 5, our implementation includes the module called "Classifier Management", which is responsible for routine re-training of the CNN model.

6 DISCUSSION

From the experimental results, the industrial projects showed better performance than open source projects in almost all cases. In addition, four industrial projects showed variations in performance. We discuss a few aspects that may have influenced the performance.

Bug report quality We suspect high-quality bug reports contributed to better performance in industrial project cases. In open source projects, bug reports are written by end-users, contributors, committers, etc. Because of this diversity, the bug reports in open source projects are more prone to lower quality and more duplicates. In contrast, the bug reports from the industrial projects were written by QA team members and systematically inspected; hence, the quality of bug reports were well-maintained.

Size of developer pool In general, the bug triage with less number of developers is easier because the number of required classification becomes smaller. From this perspective, Project B and Firefox were the most difficult cases, respectively in each category. Note that developers can easily join and leave from open source projects. Thus, open source projects tend to have larger number of developers than industrial projects as shown in Table 1. We also suspect that the developer pool of industrial projects is more stable than that of open source projects. It would be harder for the CNN triager to infer correct developers in the unstable pool.

Data imbalance The performance of machine learning can be degraded due to data imbalance in the training data, which can be partly captured by the ratio of active developers. In open source projects, it is relatively difficult to trace the status of participating individuals; hence, there is a tendency that bug fix requests are concentrated on main committers responsible for the project. As shown in Figure 2, active developers in open source projects such as JDT, Platform, and Firefox account for 21%, 24%, and 17%, respectively, which are lower compared to industrial projects. It is possible that the bug reports fixed by non-active developers did not have enough instances to train the CNN triager properly.

In our collected data, the industrial projects had overall advantages such as better bug report quality, smaller developer pool, and less data imbalance. Among the industrial projects, Project C showed the best performance due to its characteristics of the highest ratio of active developers and moderate size of developer pool. Project D was the second due to its smallest pool size and moderate ratio of active developers. Although the ratio of active developers of Project B was higher than that of Project A, its largest pool size resulted in the worst performance. It is also interesting to see the comparatively strong effect of filtering in Project A and all the open source projects, which have low ratio of active developers.

7 CONCLUSION

In this paper, we presented an automatic bug triage system based on deep learning and word embedding techniques. Our experiments included both industrial and open source projects, which was rarely attempted in previous studies. For the chosen industrial projects, we had to address the challenges of multi-languages and jargons in the bug reports.

The experiments showed that the CNN triager achieved higher accuracy in industrial projects compared to open source projects. We also observed that proper co-operations of CNN and human triagers produce the best result in terms of triage overhead reduction.

We also reported the integration of the CNN triager with the industrial development framework and shared our experience in its operation. By integrating the CNN triager with JIRA issue tracking system, we could be able to provide recommendations from the CNN triager as hints for the bug triage task.

Currently, we are running a pilot system for four industrial projects while using the CNN triager as an assistant. We are collecting the feed-backs from the human triagers and the accuracy of the recommendations is being measured. In the future, we expect to extend this system to other industrial projects in our company and report the results and findings.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea grant (NRF-2017R1D1A1B03029552) and the LG Electronics.

REFERENCES

[1] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. 2009. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*. IEEE, 216–221.

[2] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. ACM, 361–370.

[3] Atlassian. 2017. Atlassian JIRA Project Management Software. (2017). https://jira.atlassian.com/

[4] Václav Dedík and Bruno Rossi. 2016. Automated bug triaging in an industrial context. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*. IEEE, 363–367.

[5] Cícero Nogueira Dos Santos and Maira Gatti. 2014. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts.. In *COLING*. 69–78.

[6] Mozilla Foundation. 2017. Bugzilla@Mozilla. (2017). https://bugzilla.mozilla.org

[7] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 111–120.

[8] Rie Johnson and Tong Zhang. 2016. Supervised and semi-supervised text categorization using LSTM for region embeddings. *arXiv preprint arXiv:1602.02373* (2016).

[9] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering* 21, 4 (2016), 1533–1578.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.