Tracking Down Misguiding Terms for Locating Bugs in Deep Learning-Based Software (Student Abstract)

Youngkyoung Kim,[1] **Misoo Kim,** [2] **Eunseok Lee** [3]

[1] Department of Electrical and Computer Engineering, Sungkyunkwan University
[2] Institute of Software Convergence, Sungkyunkwan University
[3] College of Computing and Informatics, Sungkyunkwan University
agnes66@skku.edu, misoo12@skku.edu, leees@skku.edu

Abstract

Bugs in source files (SFs) may cause software malfunction, inconveniencing users and even leading to catastrophic accidents. Therefore, the bugs in SFs should be found and fixed quickly. However, from hundreds of candidate SFs, finding buggy SFs is tedious and time consuming. To lessen the burden on developers, deep learning-based bug localization (DLBL) tools can be utilized. Text terms in bug reports and SFs play an important role. However, some terms provide incorrect information and degrade bug localization performance. Therefore, those terms are defined here as "misguiding terms," and an explainable-artificial-intelligence-based identification method is proposed. The effectiveness of the proposed method for DLBL was investigated. When misguiding terms were removed, the mean average precision of the bug localization model improved by 33% on average.

Introduction

Software is used in products in various fields such as medical, aerospace, and transportation, and its role is getting larger. Defects in software not only inconvenience users but can also lead to serious, possibly deadly, consequences. For large-scale software used in industry, numerous bugs are reported as a bug report (BR), and there are many source files (SFs) to be searched. As a result, considerabe time is spent resolving bugs, and there may be delays when bugs that urgently need to be fixed.

The deep learning-based bug localization (DLBL) technique automates the bug localization process, making it possible to identify the bug location quickly for bug resolution (Xiao et al. 2019). The core of this technology is to take the text token of the BR and the text token of the SF as input and calculate the relevance score of SF to BR, which is used to rank SFs. Since word vectors are representative data applied to this technique, it is necessary to remove unnecessary words for search in advance. The problem is that, among text tokens that have undergone preprocessing such as stop word removal, there are misguiding terms (MGTs) that have a great influence on the DLBL model by conveying the incorrect meaning.

Some experienced developers can establish rules for selecting MGTs based on empirical knowledge. However, since this method is individual-dependent, bias may occur, and unidentified terms may exist. Therefore, we propose automatically identifying MGTs for DLBL.

Motivation

In preprocessing, stop-words that have little or no meaning, such as articles, are removed to improve computational efficiency and localization performance. However, in addition to these words, others convey incorrect information and worsen the DLBL performance.

First, the domain-specific meaning of the term can not be fully embedded in the vector representation of the term used for DLBL. For example, one can consider the terms used in the experimental dataset in this study, such as the term "eye", where the "eye" function is used to create a sparse matrix. The similarity between "eye" and "sparse" is only 0.13, while the similarity with "dense" is even higher, at 0.23. This situation means that, when there is an "eye" in the BR, the SFs having "dense" are found as buggy files, even though the buggy file contains "sparse". If such incorrect embedded words significantly impact the DLBL model, the bug localization model makes incorrect predictions.

Second, words unrelated to defects are frequently overused. For example, the data utility of PyTorch (torch.utils.data) is a package that is frequently used to call various modules, such as Dataset and DataLoader. In data-driven software, the word "data" can appear in any process, and it does not provide adequate information in all situations. In the case of deep learning-based software, which is popular data-driven software, if a DataLoader appears in the attached example code of a BR, which explains an error in the training process, it will lead to an incorrect localization result because the model concentrates on the word "DataLoader". Similarly, the word "load" can be used in various situations when loading a model, loading data, or loading a container. Eliminating such MGTs that are likely to provide irrelevant information enables the bug localization model to focus on more relevant information

Proposed Method

To find project-specific MGTs, historical textual data of a software project, DLBL model, and explainable-AI (XAI) are used to extract MGTs. The aim is to remove the terms that rank the fake buggy SF (FBSF) higher than the real buggy SF (RBSF) while preserving the terms that can make the RBSF rank higher than the FBSF.

Dataset and DLBL Model Acquisition. First, past BR and SF pairs are collected. When the SF is fixed for the BR, then this pair is labeled *Positive*. Otherwise, the pair is labeled *Negative*. Pair data are split into a DLBL-acquisition dataset (training dataset) and an impacting-word-identification dataset (validation dataset). The DLBL model is trained with the DLBL-acquisition dataset. Impacting words are identified with the trained DLBL model and impacting-word-identification dataset.

Real and Fake Buggy SF Identification. Based on the prediction of the trained DLBL model, FBSFs and RBSFs are identified. The RBSFs are a set of SFs that were actually modified to fix bugs and predicted as a buggy file. FBSFs are SFs that are not modified to fix bugs but are predicted as buggy files with a high score and ranked higher than real buggy files.

Impacting Word Recognition. With two document sets (FBSFs and RBSFs), first, identify the impacting words that caused the DLBL model to decide the label as Positive (buggy). Explainable techniques, such as LIME (Ribeiro, Singh, and Guestrin 2016), can explain the prediction of the classifier. LIME observes the impact of the word in a document by masking the words based on the normal distribution of the data, keeping the data from getting too far from the original data. Therefore, LIME is utilized to identify the impacting words. From each file, the impacting words and their scores are collected. Next, an average score of each word is calculated. A single word has two representative scores for FBSFs and RBSFs. If the word appears only in RBSFs, it remains a true positive impacting word. Words appearing only in the FBSF are false-positive words and are classified as MGTs. If words appear in both the BRSF and RBSF, those with higher average impacting scores in the FBSF than in the RBSF are classified as MGTs. The terms with higher average scores in the RBSF are not chosen because this may lower the rank of RBSF more than that of the FBSF.

MGT Removal. For a new BR, the previously obtained DLBL model can be reused without any new training. The DLBL model calculates the relevancy of the BR and each SF with the MGT removed. Real bug files are ranked higher because terms that cause the DLBL model to predict fake bug files as buggy files are excluded from the input, enabling the model to make better predictions.

Preliminary Experiment

Setting. An experiment was conducted on two relatively large open-source projects, MXNet and PyTorch-Lightning, from a recent bug benchmark (Kim, Kim, and Eunseok 2021). The effectiveness of the proposed method was demonstrated by comparing the performance of the basic DLBL model with and without MGTs. For a basic DLBL model, a simple convolutional neural network based text classification model proposed by Kim (Zhang and Wallace 2015) was used, which is not only effective for general natural language analysis but also the basis of various DLBL models (Xiao et al. 2019). The performance of the DLBL model was evaluated with widely used metrics for bug localization, mean reciprocal rank (MRR), and mean average precision (MAP). MRR and MAP represent the effort of the developer to locate the first buggy SF and all buggy SFs, respectively.

Project	#MGT	Best #MGT	Type	MRR	MAP
MXNet	180	160	w/ MGT	0.25	0.20
			w/o MGT	**0.32**	**0.25**
PyTorch-Lightning	294	60	w MGT	0.41	0.25
			w/o MGT	**0.54**	**0.36**

Table 1: Preliminary Results

Results. Since too much information can be discarded if too many words are removed, the number of MGTs (#MGT) was adjusted in increments of 10 to get the best #MGT. The numbers of collected MGTs for MXNet and PyTorch-Lightning was 180 and 294. Terms such as “data” and “load” were identified as MGTs as described in the Motivation section. Table 1 shows the performance of bug localization with the best #MGT of each project. When MGTs were removed, the DLBL model could locate buggy SFs more accurately for both projects. For MXNet, MRR and MAP improved by 25.5% and 25.3% respectively. For PyTorch-Lightning, MRR and MAP improved by 31.4% and 40.8%, respectively. The proposed method was effective for improving the ranking of the most suspicious buggy SF and improving the overall ranking of the buggy SF. The result of the preliminary experiment implies the existence of MGTs, suggesting that MGT removal can significantly improve the DLBL performance. In the future, it is planned to advance the technique for the selection of the best #MGT.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT)(2019R1A2C2006411, 2021R1A6A3A01086997)

References

Kim, M.; Kim, Y.; and Eunseok, L. 2021. Denchmark: A Bug Benchmark of Deep Learning-related Software. In *Proceedings of the 18th International Conference on Mining Software Repositories*, 540–544.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. ” Why should i trust you?” Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144.

Xiao, Y.; Keung, J.; Bennin, K. E.; and Mi, Q. 2019. Improving bug localization with word embedding and enhanced convolutional neural networks. *Information and Software Technology*, 105: 17–29.

Zhang, Y.; and Wallace, B. 2015. A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.