Bug Report Classification using LSTM and CNN for More Accurate Software Defect Locating

Anonymous Author(s)

ABSTRACT

The PI proposes a two-year project to develop an interactive and practical system that locates software defects automatically. This system will alleviate developers' effort in bug finding and improve productivity. In preliminary work, the PI developed a ranking model that ranks all the source code files for a given bug report. A source file at a higher position in the ranked list is more likely to contain defects than a source file at a lower position. The proposed system is an extension of the PI's preliminary work. In this project, the PI propose to 1) introduce a pre-filtering technique to filter out low-quality bug reports; 2) use artificial neural network to measure semantic relationship between bug reports and source code files; 3) implement an interactive 3-D VR user interface on multiple platforms for the system. The first two objectives aim at making the system be more accurate. The third objective aims at making the system be more usable. The PI propose to recruit two student assistants including one graduate and one undergraduate to help develop the system and perform experimental evaluation. The system will be published and will be used in the software engineering (SE) courses at California Sate University San Marcos to help students learn SE concepts in a lively manner.

This project will result in an interactive and practical system that can be used by the public. To increase system accuracy, this project proposes using artificial neural networks (ANNs) to classify bug reports into either "predictable" or "unpredictable", where "unpredictable" refers to low-quality reports that do not contain sufficient information for locating the bug. The system works on "predictable" reports and keeps silent on "unpredictable" reports. Besides, this project proposes using ANNs to measure the semantic similarity between a bug report and a source file. This semantic similarity is used as a new feature that complements other lexical-similarity features. To make the system be usable, this project proposes to develop an interactive user interface on multiple platforms. Furthermore, this project proposes to develop a supplemental and selectable 3-D VR visualization module that helps developers understand the code change history more efficiently.

This project will publish a practical defect positioning system that can be used in both academic and industry. The research outcome will provide the basis for the PI's future work. The methodology and techniques used in this project will contribute to the software engineering (SE) community in similar work. The funding to this project will help the PI build an SE lab and found an SE research group at California State University San Marcos (CSUSM). The research lab will provide students with the necessary computing resources for doing SE research projects. The research group will expose students to new SE techniques and research projects. The funding to this project will also help provide paid positions and new opportunities to our students that have strong desire in doing research projects.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; • **Software and its engineering → Software testing and debugging**; • **Information systems →** *Multilingual and cross-lingual retrieval*;

KEYWORDS

Recurrent neural network, long short-term memory, convolutional neural network, bug localization, bug report

ACM Reference Format:
Anonymous Author(s). 2018. Bug Report Classification using LSTM and CNN for More Accurate Software Defect Locating. In *Proceedings of The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software development is complex. During the software development and maintenance process, to ensure product quality, one of the most common tasks of a development team is bug fixing.

A software *bug* or *defect* is a coding mistake that may cause an unintended or unexpected behavior of the software component [**?**]. Upon discovering an abnormal behavior of the software project, a developer or a user will report it in a document, called a *bug report* or *issue report*. A bug report provides information that could help in fixing a bug, with the overall aim of improving the software quality. A large number of bug reports could be opened during the development life-cycle of a software product. For example, 3,352 bug reports of the Eclipse Platform product were submitted in 2016 alone[1]. In a software team, bug reports are extensively used by both managers and developers in decision making during their daily development activities [**?**].

When a new bug report is received, before being solved, it will be screened and prioritized. This screening process is called bug report triage, which aims at ensuring the bug report quality. During the triage steps, the development team will check whether the bug report contains sufficient information for developers to fix, whether it is a duplicate report, what is its priority, and who should be the appropriate developer to solve it. After the triage, the report will be assigned to a developer to fix. Then the fix will be tested usually by another developer. If the fix is verified, the report will

[1]https://bugs.eclipse.org/bugs/

be closed. Otherwise, it will be assigned to someone again. This process represents a common defect cycle or bug-fixing cycle.

A developer who is assigned with a bug report usually needs to reproduce the abnormal behavior [?] and perform code reviews [?] in order to find the cause. However, the diversity and uneven quality of bug reports can make this process nontrivial. Essential information is often missing from a bug report [?]. To locate the bug, developers not only need to analyze the bug report using their domain knowledge, but also collect information from peer developers and users to narrow their search. Employing such a manual process in order to find and understand the cause of a bug can be tedious and time-consuming [?].

Bacchelli and Bird [?] surveyed 165 managers and 873 programmers, and reported that finding defects require a high-level understanding of the code and familiarity with the relevant source code files. In the survey, 798 respondents answered that it takes time to review unfamiliar files. While the number of source files in a project is usually large, the number of files that contain the bug is usually very small. For example, the Eclipse JDT bug 424772 was fixed in a patch (commit) with four changed files, while the source code package checked out from this commit contains 10,544 Java files. Therefore, we believe that an automatic approach that ranked the source files with respect to their relevance for the bug report could speed up the bug finding process by narrowing the search to a smaller number of possibly unfamiliar files.

If the bug report is construed as a query and the source code files in the software repository are viewed as a collection of documents, then the problem of finding source files that are relevant to a given bug report can be modeled as a standard task in information retrieval (IR) [?]. In recent years, researchers [? ? ? ? ? ? ?] have proposed various IR models to recommend source code files for bug reports automatically. These models use one or more *features*. Each *feature* is a type of information that measure the relationship between the bug report and the source code file. However, the weight parameters of these models are not learned automatically. Therefore, the feature combinations of these model are sub-optimal. The model scalability is also limited.

To address this issue, in preliminary work, the PI introduced a learning-to-rank model to combine different *features* automatically [? ?]. The problem of locating software defects is approached as a ranking problem, in which the source code files (documents) are ranked with respect to their *relevance* to a given bug report (query). In this context, *relevance* is equated with the likelihood that a particular source code file contains the cause of the defect described in the bug report. The ranking function is defined as a weighted combination of *features*, where the *features* are automatically trained on previously solved bug reports using a learning-to-rank technique. We performed extensive evaluations on six large open-source Java projects, from which the results showed that the learning-to-rank model outperformed other state-of-the-art approached [? ?].

Matching bug reports with the relevant source code files is complicated by the lexical gap between natural languages used in the bug reports and the technical terms used in the source code files. To bridge this language mismatch, the PI employed word embeddings, which are vector representations of words, to measure the semantic similarity between bug reports and source code files [?]. These semantic similarities are used as additional *features* in the ranking function. Results of evaluations on four large open-source Java projects showed that adding these new word-embedding-based semantic features significantly improved the ranking performance [? ?].

1.0.1 A.1 Objectives. The results of the PI's preliminary work are promising and motivate the PI to continue this work to address the following aspects.

(1) **Bug report quality classification:** Bug report quality varies. Some bug reports do not contain sufficient information for developers to locate the defect [?]. Running a ranking system on low-quality reports may obtain misleading results. Therefore, the PI propose to include in the very beginning a pre-filtering step, in which a bug report is classified as either "predictable" or "unpredictable". After filtering out the "unpredictable" reports, the ranking system works only on the "predictable" reports and keeps silent otherwise. The purpose of doing so is to make the ranking results be more trustworthy. This project will use deep neural networks such as Convolutional Neural Network (CNN) [?] and Recurrent Neural Network (RNN) [?] for bug report classification.
(2) **Document-level similarity:** When calculating the word-embedding-based semantic features, our preliminary work used the Mihalcea's [?] similarity function, in which the similarity between a word w in one document and a bag of words in another document T is computed as the maximum similarity between w and any word w' in T: $sim(w, T) = \max_{w' \in T} sim(w, w')$. However, the similarities between w and other words in T are ignored, which cause information loss. As such, the PI propose to explore alternative methods for aggregating word-level similarities into a document-level similarity. More specifically, this project will use deep neural networks e.g. CNN and RNN to automatically learn feature vectors from bug reports and source code files respectively. Then a new semantic feature can be computed as the cosine similarity between the feature vectors of the bug report and a source file. This new feature can be used alone or as an complementary feature to improve the system performance.
(3) **User interface:** Despite many theories and algorithms were proposed recently to locate software defects automatically, practical systems with friendly user interfaces are rare, not to mention user studies that evaluate the effectiveness and usefulness of these approaches for developers in real work. Given the promising results of the preliminary work, which provides a self-learning ranking model on the back-end side, the PI propose to develop an interactive and user-friendly user interface on the front-end side. This project will develop a practical system that can be used in the software engineering course at California State University San Marcos (CSUSM) to help students better understand software engineering concepts.

1.1 B. Preliminary Work

The preliminary work of this proposal produced the following results, which provide solid support for the proposed study.

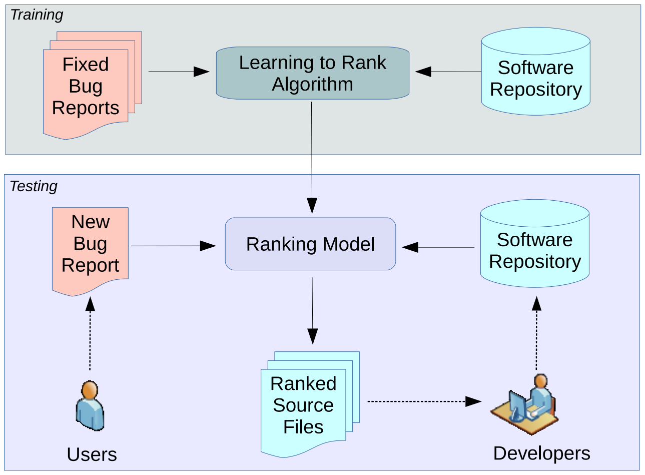

Figure 1: System architecture for *training* and *testing*.

<u>A learning-to-rank system</u> [? ?] was designed to rank all the source code files for a given bug report. Figure 1 shows the high-level architecture of this system, in which the **ranking model** is trained to compute a matching score for any bug report r and source code file s combination. The scoring function $f(r,s)$, which is shown in Equation 1, is defined as a weighted sum of k features, where each feature $\phi_i(r,s)$ refers to a measurement of the relation between the source file s and the received bug report r:

$$f(r,s) = \mathbf{w}^T \Phi(r,s) = \sum_{i=1}^{k} w_i * \phi_i(r,s) \tag{1}$$

Given an arbitrary bug report r as input at test time, the model computes the score $f(r,s)$ for each source file s in the software project and uses this value to rank all the files in descending order. The user is then presented with a ranked list of files, with the expectation that files appearing higher in the list are more likely to be relevant for the bug report i.e., more likely to contain the cause of the bug.

The model parameters w_i are trained on previously solved bug reports using a learning-to-rank algorithm [?]. In this learning framework, the optimization procedure tries to find a set of parameters for which the scoring function ranks the files that are known to be relevant for a bug report at the top of the list for that bug report.

During feature engineering, we designed different types of features including features measuring the text similarity between bug reports and code in different granularity, features measuring the text similarity between bug reports and API references, features measuring the code change history, features about similar fix, and features measuring code complexity based on code-dependency graph [?]. The calculation of text similarity is based on the classic Vector Space Model (VSM) technique, which is widely use for measuring lexical similarity between documents. Later, to bridge the lexical gaps between bug reports and code, we used word embeddings, which are vector representations of words, to create two additional features for measuring their relations [?].

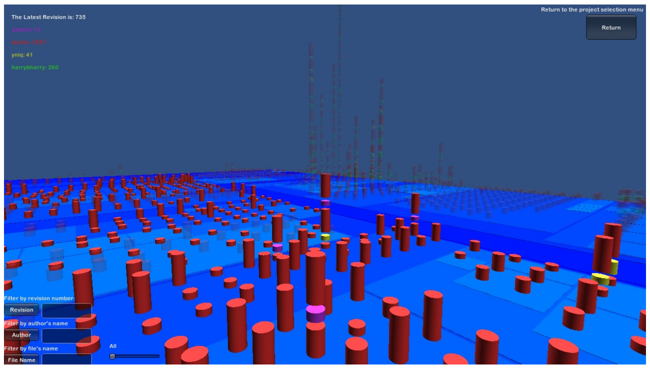
The Latest Revision is: 735
Return to the project selection menu
Return
Filter by revision number
Revision
Filter by author's name
Author
Filter by file's name
File Name
All

Figure 2: Screenshot of a 3-D software visualization tool

<u>A 3-D software visualization tool</u> [?] was developed and was used for visualizing software project's *Subversion* (SVN) repository. Figure 2 shows a visualization view. Blue districts refer to folders. A stack of cylinders refers to a source code file. A single small cylinder refers to a revision of a source file. Different colors refer to different developers that contributed to the revisions.

This visualization tool is too old to be used in the proposed study because 1) it works on a very old version control system SVN, which is seldom used now; 2) it visualized all the files of a project in one view, which causes poor poor scalability and can hardly work on large projects. However, its visualization scheme and the 3-D city metaphor will be used in the proposed study.

1.2 C. Proposed Study

The main goal of the proposed project is to develop an interactive and accurate software defect positioning system that can be used by developers to locate software defects efficiently. This practical system is a continuing work of the PI's preliminary work. To achieve the project goal, this project will pursue several objectives as discussed in Section A.1. More specifically, the PI propose to do the following work.

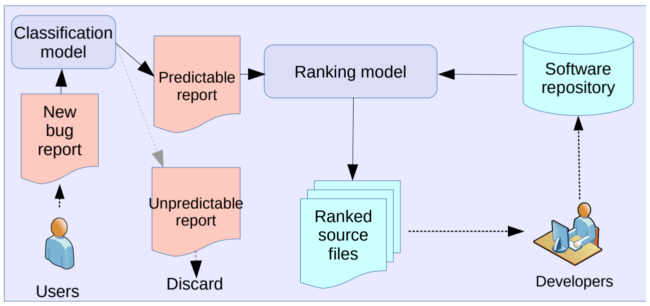

Figure 3: Ranking after pre-filtering.

1.2.1 C.1 Including a pre-filtering step to filter out "unpredictable" bug reports. Inspired by a similar work from Kim et al. [?], the PI propose to include a pre-filtering step in the ranking system. The purpose is to further increase the ranking accuracy by filtering out "unpredictable" bug reports that do not contain sufficient information for locating the bug. As shown in Figure 3, a new received bug report will first be labeled by a classification model as either "predictable" or "unpredictable". Then the ranking system will rank all the source code files for a "predictable" report in order to locate the defect. But it will keep silent for "unpredictable" reports as it has no confidence in working correctly for this types of reports.

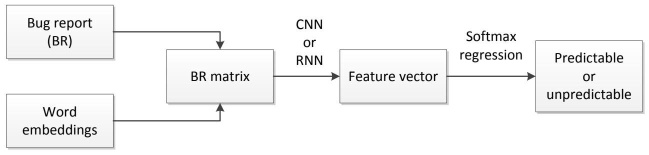

Figure 4: Bug report classification flowchart.

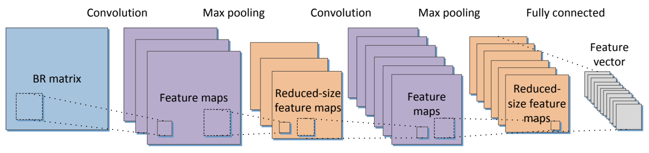

Figure 5: An example of a CNN model.

The flowchart of the classification model is shown in Figure 4, where BR stands for bug report. The PI propose to use deep neural networks such as CNN and RNN. First, the content including summary and description of a bug report will be converted into a matrix of real numbers by using word embeddings. Next, CNN or RNN is applied on the BR matrix to learn a feature vector, which will serve as the input to a softmax regression model. The output is a label of either "predictable" or "unpredictable". The weight parameters of the softmax regression model is learned by using stochastic gradient descent on a dedicated training set with manually labeled bug reports. To create the training data, we will use the ranking results from the preliminary work [?]. For a bug report in the training set, it will be label as "predictable" if the preliminary result [?] successfully locate the bug within the top-10 ranked list, and it will be labeled as "unpredictable" otherwise.

Figure 5 shows an example of a CNN model [?] with two convolutional layers. The first convolutional layer uses a number of fix-size filters to perform convolution over the BR matrix and obtain the output of the same number of feature maps. Following is a max pooling layer that reduces the size of the feature maps. The output of the first layer serves as the input to the second convolutional layer. Finally, a fully connected layer produces a feature vector for the input BR matrix. It is noteworthy that when using CNN, all the training and testing BR matrices should have the same size. This can be done by extracting a fix size e.g. 100 words from all the bug reports. Reports with less words can be padded with 0. RNN works on input with variable length.

Recently, Mills et al. [?] proposed an approach, which is based on the Lucene implementation of VSM [2], to predict query quality for assisting text retrieval tasks in software engineering. We will perform comparison with this related work and may use techniques from it to improve the proposed system.

[2]https://lucene.apache.org/

1.2.2 *C.2 Introducing a new feature to measure the semantic similarity between bug reports and code.*

In preliminary work, we use Mihalcea's function to aggregate word similarities into a document similarity. However, as discussed in Section A.1, it might cause information loss. Recently, Huo et al. [?] used CNN to to learn feature vectors from bug reports and code for measuring their similarity. But CNN requires fix-length input data and therefore may also cause information loss on code.

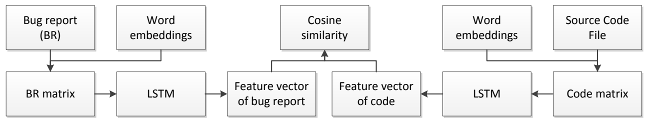

Figure 6: Using LSTM to calculate similarity.

The PI propose to use long short-term memory (LSTM) [?], a special type of RNN that can accommodate bug reports and source code files with variable length. Currently, LSTM networks are the most effective sequence learning models in many applications [?]. We will use the standard LSTM unit defined in [?]. As shown in Figure 6, LSTM will be used to learn feature vectors from bug reports and source code files. The cosine similarity of their vector representations is considered as a measurement of their semantic similarity and will be used as a new feature in the ranking model.

1.2.3 *C.5 Evaluation.*

We will evaluate the software defect positioning system on several large-scale open-source projects that contain a sufficient number (more than 2,000) of source code files and previously fixed bug reports. We will conduct experiments on software projects that are written in Java as well as projects written in C/C++ because these programming languages are widely used in both industry and academic. We will use projects from the Eclipse foundation [3] and Apache foundation [4] because 1) both have many large-scale open-source projects written in Java and C/C++; 2) the source code packages can be easily downloaded from their GIT repositories; 3) their bug reports or issue reports are public accessible. More specifically, the projects used in our preliminary work [?] will be used in this study. Additionally, we will run experiments on more projects such as Apache HTTP Server [5] written in C, Lucene [6] (an information retrieval software library) written in Java, and Hadoop [7] (a software framework for distributed storage and bigdata processing) written in Java. The selection of bug reports for evaluation is based on the same heuristics in [? ?].

We will run the system to rank all the source code files for a given bug report and compare the result with the actual fix. The evaluation metrics such as *Mean Average Precision* (MAP) [?] used in our preliminary work [?] will also be used in this study. Additionally, we will use Normalized Discounted Cumulative Gain

[3]https://eclipse.org/
[4]https://www.apache.org/
[5]https://httpd.apache.org/
[6]https://lucene.apache.org/
[7]http://hadoop.apache.org/

(NDCG) [?], which is widely used in evaluating information retrieval models e.g. web search engines, to evaluate our system.

The PI is aware that user study is an effective way to evaluate the effectiveness of the proposed system in developers' real work. However, this is not the main goal of the proposed study. So the PI leave this to future work after the system is published. This study will also provide the basis for our future research on evaluating the effectiveness and usability of the proposed system in assisting teaching software engineering course at CSUSM.

1.3 D. Work Plan

The PI plan to hire one graduate student enrolled in the Master of Science Program in Computer Science (CS) and one undergraduate student enrolled in the Bachelor of Science in Computer Science program at CSUSM to help develop the system. The PI takes the overall responsibility of directing the project and keeps mentoring the students during the development.

In the first year, the graduate student will help implement the bug-report classification model, the LSTM-based semantic similarity feature, and the ranking model running on the server-side. The undergraduate student will implement the client-side programs and the web server that takes charge of the communication between the ranking model and the client-program. Since the ranking model, the database, and the web server work closely, the students will also work together closely during the development.

In the second year, two students will take charge of the maintenance and improvement of the system. Additionally, the graduate student will help run experiments to evaluate the ranking performance of the system on several open-source software projects, analyze the results, and improve the system accordingly. The undergraduate student will help develop a supplemental 3-D VR software visualization module. By the end of this year, the practical system will be published.

Recruitment of students will begin in spring, about three months before they start to work on the project. The PI will broadcast a hiring advertisement on CS-major electronic mailing lists, post a flier on class forums, and make a presentation of the project in the college-wide Frontiers in Science talk. In the coming 2017-2018 academic year, the CS department at CSUSM has a total of 866 undergraduate enrollments and a total of 39 graduate enrollments. Many students have desire to gain hands-on experience by working with faculty members on research projects. The PI will interview interested students. Preference will be given to economically-disadvantaged students, minority students, students making good progress toward their degree, and students who have demonstrated interest in this project.

1.4 E. Broader Impacts

The proposed study will result in a practical software system that alleviates develops' effort in bug finding and improves productivity. The system will be published. Any software developers can use and test it on their own projects. The system will be used in the software engineering courses at CSUSM to help students learn software engineering concepts in a lively manner. This project will expose students to an up-to-date software engineering research topic as well as some cutting-edge techniques including artificial neural networks and virtual reality.

The research outcome of this project will provide the basis for the PI's future research in performing user study and collecting user feedback to evaluate the effectiveness of the system in both academic and industry. The experience learned from this project will be very helpful for PI's research in code recommendation and automatic programming. The methodology and techniques used in this project will contribute to the software engineering research community.

The PI graduated with a Ph.D. degree from Ohio University at May 2016 and joined the CS department at CSUSM as an tenure-track faculty at August 2016. The funding to the proposed study will help the PI obtain important computing resources for building a software engineering (SE) research lab at CSUSM to continue his research. This fund, which supports two student assistants, will also help the PI found an SE research group at CSUSM. The research group will work on adapting new techniques to solve SE tasks, applying up-to-date SE research outcome to assist teaching SE courses, and engaging students in doing SE research projects.

The funding will have a significant impact on the quality of education for students here at CSUSM. The funds requested for student assistants will allow some of our economically-disadvantaged students, may of whom work in retail and on campus dining, to have paid positions during the academic year and summer.

Furthermore, funding of the proposed research program will create new opportunities for our students that have strong desire to participate in research projects. The research opportunities created from the funding of the proposed research will directly contribute to providing talented undergraduates at CSUSM with the research experience necessary to be competitive applicants for top-tier Ph.D. programs.

2 RELATED WORK

This is the related-work section.

3 FUTURE WORK

This is the future-work section.

4 CONCLUSION

This is the conclusion section.