A Survey on Machine Learning Approaches for Software Bug Localization

Emre Doğan
Bilkent University
Department of Computer Engineering
emre.dogan@bilkent.edu.tr

Hamdi Alperen Çetin
Bilkent University
Department of Computer Engineering
alperen.cetin@bilkent.edu.tr

ABSTRACT
A software bug[1] is an error, failure or fault in a computer program that causes the program to produce invalid output or to crash. The term of *"bug localization"* means identifying the exact locations software faults. Manual techniques are commonly used to pinpoint bugs. These techniques are too expensive in terms of effort and time if they are applied manually. By making use of different machine learning techniques to detect the locations of software bugs, the process of debugging can be eased to avoid the related costs. In this paper, we examine several machine learning-based bug localization approaches from 16 different studies.

Keywords
Software fault localization, machine learning, survey.

1. INTRODUCTION
Testing is a very important and critical step in the software development lifecycle. But no matter how much time and effort are spent in testing a software, there always exist some bugs during or after the development process of the product. For this reason, detecting and even localizing these defects is very important to save a lot of effort and time. For this debugging process to be successful, it should be considered to remove as many defects in the program as possible without introducing new bugs at the same time [14].

There are several studies done in the literature with different approaches to the bug localization. Several surveys on fault localization have been published up to now [10,14,16]. All these surveys make a comprehensive review on the literature, and they include a subsection for machine learning-based approaches for bug localization, but not in a detailed manner. It is not surprising as most of studies with machine learning approach have been published within the last few years. In this survey, our motivation is to bring all the bug localization studies with different machine learning based approaches and make a comprehensive analysis on them. While these studies share some similar goals, the methods employed are quite different from each other. We aim to cluster these studies with respect to their similarities and differences. The rest of the survey is organized in the following order: in Section 2, a brief background on software bug localization is given, and then the proposed models employing machine learning are grouped with respect to several metrics in Section 3. Finally, conclusion and future work related to bug localization are discussed in Section 4.

[1] We use "bugs," "faults" and "software defects" interchangeably. We also use "program," "application" and "software" interchangeably. In addition, "locating a bug" and "localizing a fault" have the same meaning, and "a statement is covered by a test case" and "a statement is executed by a test case" are used interchangeably.[14]

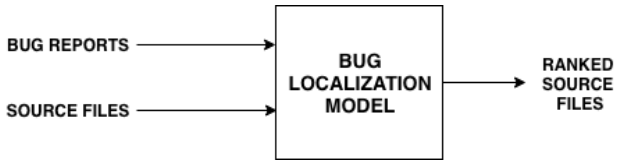

Figure 1. Simple Bug Localization System Architecture.

2. BACKGROUND
Software quality is an important concern in industry. Any software has at least some minor bugs that lead to serious costs at every stage of its lifecycle. To avoid these costs, it is very crucial to find and fix bugs as soon as possible. Software developers use code review and classical debugging methods to find bugs. Locating bugs is considered to be the most time consuming and challenging activity in this context [13]. Therefore, it would considerably ease the job of software developers if it was possible to detect and localize bugs in a software project automatically or semi-automatically. For this purpose, it is observed that machine learning might be a very supportive tool to automate the process of bug localization. The process of bug localization and auto-debugging is considered to be one of the ten most challenging problems for the next years [12]. Earlier studies on the literature focuses on more manual methods [10]:

- *Program Logging:* Statements like print are used to monitor variables, flags, etc.
- *Breakpoints:* They are used to stop the code when execution reaches a determined point.
- *Program Slicing:* It is a technique to abstract a program into a reduced form by deleting irrelevant parts.

These methods do not help developers to find and fix bugs in an automated manner and cannot reduce the cost of time and effort for debugging process.

Over the last two decades, machine learning has become a more popular and powerful tool for several classification and detection problems, including bug localization. In this survey, we will examine papers related to the machine learning based bug localization methods.

For all the studies that we examined, there are two types of architectures dealing with bug localization. One of them takes bug reports and source files as input and gives ranked source files as output. The representation of this architecture is given in Figure 1. A bug report is a document containing deterministic information about the issue itself such as bug ID, bug time, bug description. And, the source files are nothing but the files containing source codes of the software project. Ranked Source Files means a list

Study	ML Approach			Input	Output	Case Study
	NN	DNN	Other			
[1,2,5]		Yes		Bug reports, source files, project info	Ranked list of source files	Yes
[6,7]	Yes			Statement coverage info of executable tests	Ranked list of statements	Yes
[3]		Yes		Change (i.e. git commit)	Label as clean or buggy	Yes
[4]		Yes		Statement coverage info of executable tests	Ranked list of statements	Yes
[8]			Markov Network	Source code, a set of passed/failed tests	Ranked list of statements	Yes
[9]			Decision Tree	Statements, a set of passed/failed tests	Ranked list of statements	Yes

Table 1. Comparison of the studies on bug localization.

NN: Neural network, DNN: Deep neural Network

Study	Explicit Time Evaluation	Accuracy Evaluation Methods						
		EXAM	SCORES	MRR	MAP	Top-k	Cost Effectiveness	F-Measure
[1, 2]	Yes			Yes	Yes	Yes		
[3]	Yes						Yes	Yes
[4,6,7,9]		Yes						
[5]				Yes	Yes			
[8]			Yes					

Table 2. Evaluation methods used in papers.

MRR, Top-k, MAP and F-Measure are well known metrics.

EXAM: Percentage of executable statements that have to be examined until the first buggy statement is reached.

SCORES: Percentage of actual buggy statement.

Cost effectiveness: the percentage of buggy changes found when reviewing a specific percentage of the lines of code.

including the source files causing the related bug. The other type of architecture takes coverage of executable tests for statements and gives suspiciousness as output which can be used to generate ranked statements.

3. MACHINE LEARNING-BASED APPROACHES FOR BUG LOCALIZATION

In our survey, we investigated nine different papers and compared the proposed methods with respect to different metrics. Table 1 shows detailed information about these metrics.

First, all the papers are grouped by their methodology. 7 of them are empowered by neural networks. Other 2 papers focus on bug localization with decision trees and Markov logic networks. Then, we cluster the papers by their input/output configuration. The methods using similar input and output configurations are grouped together. Eventually, we group the proposed methods by accuracy evaluation metrics. Most of the approaches share some metrics but some of them use a completely different metric to evaluate accuracy. For example, in [3], a metric named *cost effectiveness* is defined which is not used by another approach. Also, in [1,2,3] time discussion is made clearly but the others do not mention it in detail. Table 2 shows evaluation methods used in the studies.

The categorizing process is done by a top-down approach. The following subsections (3.1 and 3.2) will compare the studies by their methods. Within these subsections, I/O Configuration and evaluation metric-based comparisons are completed.

3.1 Neural Network Approaches

7 papers employ neural network approaches to localize bugs. 5 of them specifically use different types of deep neural networks. Lam et al. and Xiao et al. use bug reports, source files and project metadata as input in order to rank source files according to their suspiciousness value as output [1,2,5]. Lam et al. use deep neural networks to relevance between bug reports and source code to find relevancy features. They use another deep neural network to combine these features and the ones coming from an information retrieval model and project metadata. Their study benefits from the fact that deep neural networks are capable of capturing high-level discriminative information. Xiao et al. follow another way and use an approach combining convolutional neural network and cascade forest. Wong et al. follow a different method to localize bugs, and they use statement coverage of executable tests to rank statements according to suspiciousness level as output [6,7]. They study on two different neural networks such as radial bases function neural network and back propagation neural network. Even though these studies are relatively old, they are quite important as they are among the first studies using neural network in bug localization. Zheng et al. propose an approach which uses same input and output with the previous studies using deep neural network instead of neural network [4]. Also, Yang et al. propose another approach for just-in time defect prediction which performs deep belief networks to find whether the change (i.e. commit) is clean or buggy.

3.2 Other ML Approaches

Before deep learning became as popular as it is today, different machine learning approaches were employed for bug localization. Besides them, some old-fashioned machine learning methods are still very useful as they are understandable by human. We investigate two examples employing this kind of machine learning method.

One approach is proposed by Briand et al. employing the C4.5 decision tree algorithm and Tarantula [15], a technique that utilizes correlation-based heuristics, and gives the output of the ranked list of statements in terms of suspiciousness [9].

Another approach is given by Zhang et al. to solve the problem by Markov logic networks [8]. The input& output configuration is very similar to Tarantula, only replacing statements with source code. What makes this method special is that the authors used a very simple and unique architecture.

4. DISCUSSION AND CONCLUSION

In this survey, we examined more than 15 papers related to bug localization and compared 9 of them that are using machine learning. For this purpose, different metrics such as machine learning model, I/O configuration and accuracy evaluation methods are used to group these 9 papers.

One important problem we struggled is the lack of studies on machine learning-based approaches for bug localization. In the last 3 years, the number of studies with machine learning approaches showed a great increase and still going on. We believe that more and more studies with state of art machine learning techniques will be published in the near future. Then, more comprehensive surveys can be published.

The study distribution according to years is analyzed, and it is concluded that with the hype of deep learning in the last years, the number of papers employing deep learning for bug localization are increasing. This tells us that deep learning will take an important place in the future studies of this research area.

To conclude, our aim is to use this survey to provide the software engineering literature with a brief, state-of-the-art research in machine learning applications for software fault localization.

5. REFERENCES

[1] Lam, An & Nguyen, Anh & Nguyen, Hoan & N. Nguyen, Tien. (2015). Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N). 476-481. 10.1109/ASE.2015.73.

[2] Lam, An & Nguyen, Anh & Nguyen, Hoan & N. Nguyen, Tien. (2017). Bug Localization with Combination of Deep Learning and Information Retrieval. 218-229. 10.1109/ICPC.2017.24.

[3] Yang, Xinli & Lo, David & Xia, Xin & Zhang, Yun & Sun, Jianling. (2015). Deep Learning for Just-in-Time Defect Prediction. 17-26. 10.1109/QRS.2015.14.

[4] Zheng, Wei & Hu, Desheng & Wang, Jing. (2016). Fault Localization Analysis Based on Deep Neural Network. Mathematical Problems in Engineering. 2016. 1-11. 10.1155/2016/1820454.

[5] Yan Xiao, Jacky Keung, Qing Mi, and Kwabena E. Bennin. 2018. Bug Localization with Semantic and Structural Features using Convolutional Neural Network and Cascade Forest. In Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018(EASE'18). 101-111. 10.1145/3210459.3210469

[6] Ericwong, W & , Yuqi. (2011). BP NEURAL NETWORK-BASED EFFECTIVE FAULT LOCALIZATION. International Journal of Software Engineering and Knowledge Engineering. 19. 10.1142/S021819400900426X.

[7] Wong, W., Debroy, V., Golden, R., Xu, X. and Thuraisingham, B. (2012). Effective Software Fault Localization Using an RBF Neural Network. IEEE Transactions on Reliability, 61(1), pp.149-169.

[8] Zhang, S., & Zhang, C. (2014). Software bug localization with markov logic. *ICSE Companion*.

[9] Briand, Lionel & Labiche, Yvan & Liu, Xuetao. (2007). Using Machine Learning to Support Debugging with Tarantula. Proceedings - International Symposium on Software Reliability Engineering, ISSRE. 137 - 146. 10.1109/ISSRE.2007.31.

[10] Eric Wong, W & Gao, Ruizhi & Li, Yihao & Abreu, Rui & Wotawa, Franz. (2016). A Survey on Software Fault Localization. IEEE Transactions on Software Engineering. 42. 1-1. 10.1109/TSE.2016.2521368.

[11] Ascari, Luciano César & Araki, Lucilia & Pozo, Aurora & Vergilio, Silvia. (2009). Exploring machine learning techniques for fault localization. 2009 10th Latin American Test Workshop, LATW 2009. 1 - 6. 10.1109/LATW.2009.4813783.

[12] Thomas G. Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton, and Prasad Tadepalli. 2008. Structured machine learning: the next ten years. Mach. Learn. 73, 1 (October 2008), 3-23. 0.1007/s10994-008-5079-1

[13] Eichinger, F., & Böhm, K. (2010). Software-Bug Localization with Graph Mining. *Managing and Mining Graph Data*.

[14] Eric Wong, W & Debroy, Vidroha. (2009). A Survey of Software Fault Localization.

[15] A. Jones, James & Harrold, Mary & Stasko, John. (2002). Visualization of Test Information to Assist Fault Localization. 10.1145/581396.581397.

[16] Pathak, D.P., & Dharavath, S. (2014). A Survey Paper for Bug Localization.