Proceedings

2023 30th Asia-Pacific Software Engineering Conference

APSEC 2023

4 - 7 December 2023
Seoul, Korea

CPS
Conference Publishing Services

Detecting Common Weakness Enumeration through training the Core Building Blocks of Similar Languages based on the CodeBERT Model

Chansol Park
Software Engineering Laboratory
Hongik University
Sejong, Korea
c2193102@g.hongik.ac.kr

R. Young Chul Kim
Software Engineering Laboratory
Hongik University
Sejong, Korea
bob@hongik.ac.kr

***Abstract*—The traditional code analysis approach to measure code quality, such as bad smells, coupling, cohesion, complexity, and common weakness enumeration, was a rule-based mechanism. In current, analyzing code through AI is being studied by many researchers. AI code analysis approach trains code patterns to AI model through labeled code datasets. The problem is that AI models require plenty of training datasets for better performance, but there are not enough datasets to train. To solve this problem, we suggest training an AI model with core building blocks of a target language and similar languages together. We can solve two problems the lack of datasets and the low performance of the AI model. Finally, we will compare the performance of the two models. One is a model trained with a dataset containing only one target language, and the other is a model trained with a dataset containing similar languages.**

Keywords—Software Engineering, Static Analysis, Common Weakness Enumeration, Artificial Intelligent Quality

I. Introduction

Currently, some static analyses are used to measure source code quality. We manually define code patterns for code quality, such as Bad Smell, Coupling, Cohesion, Complexity, and Common Weakness Enumeration (CWE), and measures code quality through such defined rules. Now, many researchers are adapting software engineering with Artificial Intelligence(AI) approach. Artificial Intelligence and Software Engineering Research can be divided into AI for SE or SE for AI. AI-based approaches for code quality measurements are also part of such research. On AI-based static analysis, we train patterns of code quality to AI models. The trained AI model measures code quality. At this time, for the AI model to learn the pattern of code well, many high-quality datasets are absolutely needed. But still, we need more with the number of datasets labeled with code quality. To solve this problem, we propose training an AI model with core building blocks of a target language and similar languages together. Our approach may solve the lack of datasets. It may also improve the performance of the AI-based code quality measurement.

II. Related Works

A. Rule-Based Code Visualization

Code visualization technique expresses code design and quality analyzed from the source code. Rule-based code visualization studies have been studied in various ways [1, 2, 3, 4, 5, 6]. The difficulty of rule-based code quality extraction is that we must manually define the rules for all code quality. Therefore, if there are many patterns of code to be detected, such as CWE, it takes a long time to define the rule. Also, sometimes defining rules can be impossible too complex.

B. Common Weakness Enumeration Detection Based On the CodeBERT Model[7]

The CodeBERT is a programming language model pre-learned with Python, JAVA, JavaScript, Go, Ruby, and PHP. This study trains CodeBERT Model with PMD to generate a learning dataset. The PMD detects vulnerabilities in the code line. The detected vulnerability is converted into a CWE element and labeled on the code line. Subsequently, only certain CWE elements are extracted from the dataset and tokenized to train models that detect specific CWE elements. The model is transfer-learned through the generated token sequence. This trained model can identify whether a specific CWE element exists in the code line of the JAVA language.

Model could detect the code line contained the vulnerability well, and still did not determine whether the vulnerability was harmful or not. To solve this problem, we try to train with code building blocks rather than code lines. However, we can't find enough dataset which is labeled with the core building blocks and also it was very difficult to create dataset ourselves.

C. CVEfixes Dataset

CVEfixes dataset is a dataset that extracted CVE and CWE vulnerability information from 1754 open source data and extracts and organizes source code information that has been resolved through 5495 vulnerability correction commitments[8]. In the CVEfixes dataset, vulnerable code in function units can be extracted. Improved functions can also be extracted due to modifying the vulnerable parts.

III. Training Core Building Blocks of Simmilar Languages based on the CodeBERT Model

Due to the lack of a JAVA dataset for training, we train the CodeBERT Model with core building blocks of a target language and similar languages together. Although some languages are different, they may have similar structures and vocabulary. We compare example functions written in JAVA with example functions written in different languages that the model used for pre-learning with Jaccard similarity between the vocabulary used in the token sequences. Table 1 shows the signatures of example functions and result of tokenized signatures. Table 2 shows the types and numbers of vocabulary used in example functions written in each language and the Jaccard similarity with JAVA example functions. The Core Building Blocks of JavaScript are the most similar to JAVA, and Ruby was the most distant.

2640-0715/23/$31.00 ©2023 IEEE
DOI 10.1109/APSEC60848.2023.00088

TABLE I. The Token Sequence of Function Signature.

Token Sequence of Function Signature	
JAVA	int sampleFunction(int dan){
	[0, 2544, 7728, 47802, 1640, 2544, 20435, 48512]
JavaScript	function sampleFunction(dan){
	[0, 35435, 7728, 47802, 1640, 14856, 43, 25522]
PHP	function sampleFunction($dan): int{
	[0, 35435, 7728, 47802, 1640, 68, 14856, 3256, 6979, 45152]
Python	def sampleFunction(dan):
	[0, 9232, 7728, 47802, 1640, 14856, 3256]
Ruby	def sampleFunction(dan)
	[0, 9232, 224, 1215, 42891, 1640, 14856, 43]
Go	func sampleFunction(dan int) int{
	[0, 48901, 7728, 47802, 1640, 14856, 6979, 43, 6979, 25522]

TABLE II. The Comparison of Jaccard similarities between vocabularies used in example functions.

Language	Vocabulary	Number of Vocabulary	Jaccard Similarity
JAVA	{0, 48512, 898, 131, 45056, 49789, 134, 2, 1, 150, 28696, 1437, 671, 939, 7728, 47802, 321, 6979, 35524, 50118, 49230, 5457, 20435, 1493, 45152, 1640, 361, 111, 2544, 112, 114, 1009, 24303, 118, 8061}	35	.
JavaScript	{0, 45056, 898, 48512, 2, 1, 134, 14856, 150, 28696, 1437, 671, 43, 939, 7728, 25522, 47802, 321, 35524, 50118, 49230, 5457, 20435, 42964, 1493, 45152, 1640, 361, 35435, 111, 112, 1009, 114, 24303, 118, 8061}	36	0.775
PHP	{0, 45056, 48512, 131, 2, 49789, 134, 1, 14856, 44688, 43155, 150, 28696, 1437, 671, 7728, 3256, 47802, 321, 6979, 68, 35524, 50118, 49230, 5457, 1493, 45152, 1640, 361, 35435, 111, 112, 1009, 114, 24303, 118, 8061}	37	0.756
Python	{0, 1, 898, 2, 134, 14856, 49419, 9232, 150, 28696, 1437, 671, 35, 939, 7728, 50, 3256, 47802, 321, 50118, 49230, 5457, 20435, 1493, 4832, 1640, 361, 111, 112, 1009, 114, 8061}	32	0.595
Ruby	{0, 1, 898, 2, 134, 14856, 42891, 49419, 9232, 150, 28696, 1437, 671, 43, 939, 50, 1215, 321, 50118, 49230, 5457, 20435, 1493, 224, 1640, 361, 109, 111, 112, 1009, 114, 1397, 253, 8061}	34	0.500
Go	{0, 45056, 898, 15747, 2, 48901, 134, 1, 14856, 13, 28696, 1437, 671, 43, 939, 7728, 25522, 47802, 321, 6979, 35524, 50118, 49230, 5457, 20435, 42964, 1493, 45152, 1640, 361, 111, 112, 1009, 114, 24303, 8061}	36	0.690

IV. Case Study

As a case study we train three models. The first model was trained using only the JAVA dataset. The second model was trained with the JAVA, JavaScript, and PHP datasets, which are Jaccard similarities with JAVA greater than 0.7. The third model was trained using a dataset of all six languages. We compare the performance of each of the three models with CWE test cases in Juliet Java 1.3[9] after training.

TABLE III. The Statistics of the Dataset used for Transfer Learning of the CodeBERT model.

Training Data	Number of Functions
Only JAVA Dataset	3,026
Languages with greater similarity than 0.7(JAVA, JavaScript, PHP) Dataset	255,190
All Language Dataset	266,048

TABLE IV. The Test results of Three models (accuracy).

CWE	Model training with JAVA	Model training with JAVA, JavaScript, PHP	Model training with All Languages
CWE-613	0.61702	0.58574	0.44681
CWE-23	0.45626	0.51596	0.51300
CWE-36	0.45686	0.53073	0.54078
CWE-325	0.36170	0.44681	0.45745
CWE-328	0.36170	0.52482	0.46099
CWE-338	0.36170	0.61702	0.38298

Table 3 shows the three Statistics of the dataset used to train the CodeBERT model. Table 4 shows the accuracy of test results through extracted functions from Juliet Java 1.3. Models trained only through the JAVA dataset did not learn well because the data used for the training itself was not sufficient. Models trained through JAVA, JavaScript, and PHP datasets are classified most accurately. Models trained through the datasets from all languages had lower accuracy than models learned in similar languages, even though they trained with the largest data.

V. Conclusion & Future Work

In this paper, we propose a method for learning AI models, including datasets related to the Core Building Blocks of similar languages. Training the model after supplementing the insufficient dataset through similar languages could increase the model's performance. In addition, a language with low similarity hindered the model's performance. Through this, it is possible to solve the problem of dataset shortage when learning artificial intelligence models and improve the model's performance. In the future, after improving the CWE detection accuracy for functions, we will compare performance with existing rule-based CWE detection tools.

Acknowledgment

This research was supported by Korea Creative Content Agency(KOCCA) grant funded by the Ministry of Culture, Sports and Tourism(MCST) in 2023(Project Name: Development of AI-based user interactive multi-modal interactive storytelling 3D scene authoring technology, Project Number: RS-2023-00227917, Contribution Rate: 50%) and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education in 2023(Project Name: The Faultless Codinglization through Automatic Refactoring based on NLP BERT Model, Project Number: 2021R1I1A3050407, Contribution Rate: 50%).

References

[1] G. H. Kang, R. Y. C. Kim, G. S. Yi, Y. S. Kim, Y. B. Park, and H. S. Son, "A Study on Code Static Analysis with Open Source-Based Tool Chainization." KIISE Transactions on Computing Practices, Vol. 21, No. 2, pp. 148-153, 2015.

[2] H. Kwon, and R. Young Chul Kim, "Extracting Use Case Design Mechanisms via Programming based on Reverse Engineering." International Journal of Applied Engineering Research, Vol. 10, No. 90, pp. 503-505, 2015.

[3] B. K. Park, G. H. Kang, H. S. Son, B. K. Jeon, and R. Y. C. Kim, "Code Visualization for Performance Improvement of Java Code for Controlling Smart Traffic System in the Smart City." Applied Sciences, Vol. 10, No. 8, 2020.

[4] S. J. Jung, J. H. Kim, W. Y. Lee, B. K. Park, H. S. Son, and R. Y. C. Kim, "Automatic UML Design Extraction with Software Visualization based on Reverse Engineering." International journal of advanced smart convergence. Vol. 10 No. 3, pp. 89-96, 2021.

[5] W. Y. Lee, and R. Y. C. Kim, "Best Practices on Validation and Extraction of Object oriented Designs with Code Visualization Tool-chain." Journal of Internet Computing and Services, Vol. 23, No. 2, pp. 79-86, 2022.

[6] C. S. Park, W. S. Jang, and R. Y. C. Kim, "Tool Chain Mechanism with Identifying and Collecting High Quality Data for Learning Bad Code based on Code Visualization." 2023 Conference of KISM, Vol 12, No. 1, pp. 52-53, 2023.

[7] C. S. Park, J. H. Kim, S. Y. Moon, and R. Y. C. Kim, "Applied Practice on Identifying Bad Codes through Supervised Learning with Bad Code Patterns.", Proceedings of the 25th Korea Conference on Software Engineering, Vol. 25, No. 1, pp. 119-120, 2023.

[8] G. Bhandari, A. Naseer, and L. Moonen. "CVEfixes: automated collection of vulnerabilities and their fixes from open-source software." Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering. 2021.

[9] NSA Center for Assured Software, 2017, "Juliet Java 1.3," [Online]. Available: https://samate.nist.gov/SARD/test-suites/111.