code2vec for C: The Acquisition Method of Distributed Representation of the C Language with The TF-IDF Method

Kotori Hieda[2,a)] Kenji Hisazumi[1,b)] Hirofumi Yagawa[3] Akira Fukuda[1,c)]

Abstract: Code2vec is a method for obtaining a distributed representation of program code. It obtains the embedding vector of program code through machine learning and predicts the label such as ‘method body’ representing the functionality of the code snippets. Thus, it is possible to obtain a distributed representation of the code snippet whose meaning is taken into account. In embedded system development, the non-object-oriented programming language C is often used, however code2vec is intended for object-oriented programming languages such as Java and C#. Therefore, to apply code2vec to the C language, there are some problems that must be solved: labelling is difficult since the function name differs from that of the object-oriented language, and we need to develop a method of feature amount extraction from C language. In the following paper, we propose a method for extracting feature values from the C language programs and making use of the TF-IDF method for decomposing a given function name into both module-specific names and general operation names, as found in object-oriented languages.

Keywords: code2vec, TF-IDF, C language, code snippet, function name estimation

1. Introduction

In natural language processing, methods for obtaining distributed representations that take into account their meanings such as word2vec[1] have been proposed and applied in various ways. Current software development is supported in various ways using similar methods in program code, however there is room for improvement in both methods and applications.

Code2vec[2] has been proposed as a method for obtaining the distributed representation of a given program’s code. In this approach, it is possible to numerically express the code snippets and the relationship between them as real vectors. The main application of the distributed representations obtained by code2vec is to estimate the method name from the method body. Code2vec is designed for object-oriented languages such as Java and C#.

C language is often used in embedded systems, however, as it is not an object-oriented language, it is not possible to directly apply C language to code2vec. In particular, when estimating the function name, estimation accuracy decreases because the function name itself contains both the module-specific name and general operation name.

In this study, we apply TF-IDF (Term Frequency Inverse Document Frequency)[3] to the function name to remove module-specific names based on the TF-IDF value as shown in Table 1.

Table 1 Result of using TF-IDF method

BEFORE	AFTER
ext|get|nojournal	get|nojournal
ext|put|nojournal	put|nojournal
ext|journal|check|start	check|start
ext|journal|start|sb	start|sb
ext|journal|stop	stop
ext|journal|start|reserved	start|reserved
ext|journal|abort|handle	abort|handle
ext|journal|get|write|access	get|write|access
ext|forget	forget
ext|journal|get|create|access	get|create|access
ext|handle|dirty|metadata	handle|dirty|metadata
ext|handle|dirty|super	handle|dirty|super

The TF-IDF value increases when a certain word appears frequently in a specific document, whilst only appearing infrequently in other documents. Since the module-specific name is unique within a document, its TF-IDF value is high. Therefore, we set a threshold for the TF-IDF value and delete the one with the highest value to leave only the operation name. By learning only operation names as function names, we aim to improve the accuracy of function name estimation.

2. Related research

The CMU-SEI group[4] implemented a C parser for code2vec. They paid attention to syntactic differences between parsers of C and Java, such as function declarations in C are not in Java. They

[1] Faculty of Information Science and Electrical Engineering, Kyushu University
[2] Graduate School of Information Science and Electrical Engineering, Kyushu University
[3] Fujitsu Kyushu Network Technologies Limited
[a)] hieda@f.ait.kyushu-u.ac.jp
[b)] nel@slrc.kyushu-u.ac.jp
[c)] fukuda@f.ait.kyushu-u.ac.jp

used Clang and LLVM[5], and tried to apply code2vec to C language by abstracting the functions. In this case, the accuracy of function name estimation may not improve because the module name and operation name are not separated.

3. Analytical method

In this section, we describe the analysis procedure of C language program code. Since a C language program is made up of a plurality of files, it performs the processing for each file. First, it extracts a function definition. A function definition consists of a function name, arguments, return value, and function body.

The function name is usually a compound word, so we extract only the general terms. Since camel case and underscore are used to represent word breaks, we extract words by separating them. Besides, numerical values are often inappropriate as general words, so we delete them. We do them for all prepared C language programming code files.

We calculate all occurrences number of documents about a word (DF) and consider this as an entire dictionary. We also make a dictionary about the frequency of appearance of each word (TF) for each file. Based on these dictionaries, we calculate the TF-IDF value. We calculate the TF-IDF value of a word X in document A as (frequency of word X in document A) / (total frequency of all words in document A) * log (total number of documents) / (number of documents containing word X). After calculating all the words' TF-IDF, it makes a function name with the deletion of words below the threshold.

Next, we extract features from the function body and analyze code2vec following other language's implementation method. We parse the function body, convert it to an abstract syntax tree, and extract all terminal symbols in the tree. We consider all combinations of the extracted terminal symbols and extract a sequence of non-terminal symbols and terminal symbols connecting terminal symbols as a path.

Code2vec learns the function name consisting of only common words and the feature value extracted from the function body.

4. Result

Table 2 shows a comparison of the results of CMU-SEI and our function name estimation. We compare the top 50 repositories about C language on GitHub as learning data. According to this, it found that CMU-SEI had better accuracy. The reason for this is that CMU-SEI considers the difference between C and Java in 6 major elements, and implements it considering the differences.

1. The C parser uses a simpler maximum leaf node metric beyond which it will skip a function.

2. The C parser has a parameter that can be set that determines whether to tag or discard function declarations when they are encountered.

3. The C parser does not numbers child nodes and includes this information in the generated code paths. 4. The C parser uses a sha256 hash code.

5. The C parser is more permissive and will only filter out symbols, some of which will break file formats.

6. The C parser collects all bags of path contexts into a single unique list, shuffles the list, and then samples the datasets (test, val, train) from the shuffled list.

Table 2 Comparison between CMU-SEI and Kyushu University

	CMU-SEI	proposed
Precision	0.1455	0.0381
recall	0.1239	0.0520
F1	0.1338	0.0440

I think that CMU-SEI had better accuracy because of the implementation considering these factors. So we will try applying TF-IDF to the research of CMU-SEI to given better results.

5. Conclusion

In this paper, we showed how to extract features using LLVM and Clang to apply code2vec to C language. In tasks that estimate function names from function bodies, identifiers such as C language function names are often composed of compound words consisting of module-specific names and general operation names so we argued that this could be an obstacle. To solve this problem, we proposed a method for classifying function names in C language into module-specific names and operation names using TF-IDF. As a result, we found that we can extract only general operation names from function names even in C language. Future challenges include applying TF-IDF to the research of CMU-SEI, the evaluation by applying the proposed method and applying to other identifiers such as variable names and function names. We hope we can finally infer the function from the function name.

References

[1] Rong, X.: word2vec parameter learning explained, *arXiv preprint arXiv:1411.2738* (2014).

[2] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: code2vec: Learning distributed representations of code, *Proceedings of the ACM on Programming Languages*, Vol. 3, No. POPL, p. 40 (2019).

[3] Ramos, J. et al.: Using tf-idf to determine word relevance in document queries, *Proceedings of the first instructional conference on machine learning*, Vol. 242, Piscataway, NJ, pp. 133–142 (2003).

[4] cmu sei: code2vec-c, Software Engineering Institute (online), available from ⟨https://github.com/cmu-sei/code2vec-c⟩ (accessed 2019).

[5] Lattner, C. et al.: clang: a C language family frontend for LLVM, *Website* (2007).