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 embed vectors of reals 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 code snippet. In this approach, it is possible to numerically express the code snippets and the relationship between them as vectors of reals. 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. The TF-IDF value increases when a certain word frequently appears 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.

Table 1 Result of using TF-IDF method

Before	After
ext| get| nojoumal	get| nojoumal
ext| put| nojoumal	put| nojoumal
ext| chek| start	chek| start
ext| start| sb	start| sb
ext| stop	stop
ext| start| reserved	start| reserved
ext| abort| handle	abort| handle
ext| get| create| access	get| create| access
ext| handle| dirty| metadata	handle| dirty| metadata
ext| handle| dirty| super	handle| dirty| super

2. Related research

The CMU-SEI group[4] has implemented a C parser for code2vec. They paid attention to syntactic differences between parsers of C and Java, such as function declarations which exist in C but not in Java. They used Clang and LLVM[5], and tried to

[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

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 multiple different files, the program performs the processing for each file individually. First, it extracts a function definition, which consists of a function name, arguments, return values, and a 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. Furthermore, numerical values are often inappropriate as general words, so we delete them. We do this for all prepared C language programming code files.

We calculate the number of occurences of documents containing a specific word (DF) and consider this as an entire dictionary. We also make a dictionary of the frequency of appearance of each word (TF) for each file. Based on these dictionaries, we calculate the TF-IDF value.

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i \qquad tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \qquad idf_i = \log \frac{|D|}{|d : d \ni t_i|}$$

with

- $n_{i,j}$: number of occurrences of the word t_i in the document d_j.
- $\sum_k nk$, j: sum of occurrences of all words in document d_j.
- $|D|$: total number of documents in the corpus.
- $|d : d \ni t_i|$: number of documents where the term t_i appears.

After calculating all the words' TF-IDF, we obtain 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 between the results of CMU-SEI and our function name estimation. We compare the top 50 C language repositories on GitHub as learning data. According to our result, it is found that CMU-SEI had better accuracy. The reason for this is that CMU-SEI considers these differences.

1. CMU-SEI limited the maximum number of leaves in an AST to 32, but we did not do this. The more leaf nodes there are, the more complex it becomes, so we should limit it.
2. We think that the AST that we create is different from the AST that CMU-SEI creates. It may be related to accuracy.
3. CMU-SEI rearranged the training data randomly and divided it into training data, test data, and validation data, but we sorted the data in alphabetical order. Since similar names are in the same group, the accuracy may have dropped.

Table 2 Comparison between CMU-SEI and proposed method[6] [7]

	CMU-SEI	proposed method
precision	0.1455	0.0381
recall	0.1239	0.052
F1	0.1338	0.044

Precision - Precision is the proportion of true positives among the positive predictions.
Recall - Recall is the ratio of true positive.
F1 score - F1 Score is the harmony mean of Precision and Recall.

We believe that CMU-SEI presents better accuracy due to its' implementation which considers the above factors. As such, we will apply TF-IDF to the research of CMU-SEI in an attempt to improve our 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 work includes applying TF-IDF to the research of CMU-SEI, and evaluating the application of our proposed method to other identifiers such as variable names and function names with the goal being the estimation of the function from the function name. Besides, since the accuracy may change depending on the threshold for the TF-IDF method, it is also significant to consider where the threshold to be set is. We will set various thresholds to find the best one to improve accuracy.

References

[1] Lian, Z.: Exploration of the Working Principle and Application of Word2vec, *Sci-Tech Information Development & Economy*, Vol. 2, pp. 145–148 (2015).
[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, pp. 1–29 (2019).
[3] Zhang, W., Yoshida, T. and Tang, X.: A comparative study of TF* IDF, LSI and multi-words for text classification, *Expert Systems with Applications*, Vol. 38, No. 3, pp. 2758–2765 (2011).
[4] Software Engineering Institute: code2vec-c, Carnegie Mellon University (online), available from ⟨https://github.com/cmu-sei/code2vec-c⟩ (accessed 2019).
[5] Balogh, G. D., Mudalige, G. R., Reguly, I. Z., Antao, S. and Bertolli, C.: OP2-Clang: A source-to-source translator using Clang/LLVM LibTooling, *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, IEEE, pp. 59–70 (2018).
[6] Flach, P. and Kull, M.: Precision-recall-gain curves: PR analysis done right, *Advances in neural information processing systems*, pp. 838–846 (2015).
[7] Sasaki, Y.: The truth of the F-measure, *Teach Tutor mater*, Vol. 1, No. 5, pp. 1–5 (2007).