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

【Related research】

The CMU-SEI group 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, 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.

【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 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, 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 determine all possible combinations of terminal symbols, and the paths (the sequence of non-terminal symbols contained between terminal symbols), are considered as features.*

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

【Result】

Table2 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 RESULTS, it IS 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.

WE BELIEVE that CMU-SEI PRESENTS better accuracy because 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.

【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.