NAME: MOHAMMADREZA ARDESTANI

DATE: SEP 16TH, 2022

PYTHIA: AI-ASSISTED CODE COMPLETION SYSTEM

ALEXEY SVYATKOVSKIY, YING ZHAO, SHENGYU FU, NEEL SUNDARESAN

PROBLEM(S) ADDRESSED

The paper addresses the low accuracy of the current code completion systems, by 20-percent improvement, as the main problem. In the modeling section, there are several challenges regarding how to capture long dependencies in codes while preventing vanishing gradient problems or how to leverage preprocessing with AST or Embedding technics.

MOTIVATION

Existing code completion systems are solely based on the occurrence frequency of attributes and methods. However, we can reach high precision in code completion systems with abundant publically available data, namely GitHub open source repositories, algorithmic developments, and Neural Network Models. This paper has successfully reached the objective mentioned above.

PROPOSED SOLUTION

Collected non-forked GitHub repository codes are used to generate code snippets that go through a task-designed preprocessing pipeline that AST using CFG. Finally, serialized sequence matrix gets produced from the DFS traverse of their AST representation. As a common practice in sequence processing, the embedding technic is used to transfer sparse representations, one-hot vectors, to dense representations of them, which encodes semantical relations. Multiple variations of Recurrent Neural Networks are implemented. LSTM combined with attention, however, outperforms other architecture by nearly 1 percent. Lastly, the best model is deployed on the web and also integrated into IDEs.

EVALUATION & RESULTS

Regarding space complexity, the best model is less than two hundred MB, which is larger than baselines, but it can be handled on the client side. Although Time Complexity is an important factor for code completion systems, it is not discussed in the paper clearly. Further, for evaluating model prediction performance, they use different variations of accuracy metrics, namely top-k and MMR, which take into account the rank of relevant recommendations and do not only do first matching. The best model achieves 92 percent top-k accuracy outperforming MC and Frequency-based baselines by 20 percent on average over classes.

REFLECTIONS ON LEARNING

the most inspiring part was using AST as an assistant for better sequence preprocessing. We can use a similar way in natural language processing tasks, namely domain-specific text summarization systems.

In addition, I was surprised that we don’t have an existing implemented baseline for Code Completion on account of the high popularity of this research area, and the authors had to implement the baselines on their own.

I learned a new method for handling long sequence back propagation described in the paper. It’s named truncated back propagation through time which enables us to train a sequence model with long inputs without facing vanishing or exploding gradient problems.

Another insightful part of the paper was, reusing the input word embedding matrix in the output classification matrix. I usually use a fully connected layer after the time step in classification tasks but now I can adopt their idea to reuse input embedding and reduce my model size.

Research Ideas or Directions (Graduate Student)

We can also incorporate comments and functions' descriptions instead of only using AST of variables and methods. Numpy library, for example, has the second largest of samples while the model has the least accuracy on Numpy after ws library. While we can use Numpy functions' descriptions to redress the balance of low accuracy. Eventually, this idea of using metadata and function descriptions will lead us to build a system similar to Copilot.

It is said: "We train a word embedding on the corpus of the source code repositories," which indicates that the authors have used the raw source codes for the W2V method. It is worth trying to train the embedding model from serialized AST because more than a position of the word, the dependency between words in source code matters, and AST represents a meaningful dependency between different parts of the code.