Improving Code Completion in Pharo Using N-gram Language Models

Myroslava Romaniuk
Ukrainian Catholic University
romaniuk@ucu.edu.ua

ABSTRACT

In this paper we present applying statistical language models to improve code completion in Pharo [1]. In particular, the goal is to use n-gram models for sorting the completion candidates and in such a way increase the relevancy of untyped suggestions.

1 INTRODUCTION

Code completion is one of the essential features of any IDE that greatly improves the developer experience and productivity. Thus, it is important to find a way to make it as effective as possible. Specifically, in the case of Pharo, a dynamically-typed Smalltalk-based programming language and IDE, in the last year and a half a lot of work has been carried out to improve code completion. However, there are more potential improvements from which developers can only benefit, such as improving the relevance of suggestions and hence reducing time and effort the developer spends to go through the list of completions.

2 CONTEXT

The current implementation of code completion in Pharo is based on abstract syntax tree (AST) information of source code, where we are able to determine the structure of the code and infer the type information where possible. Once we have a list of semantically suitable completion candidates, they are sorted alphabetically. However, this approach ignores the potential of suggesting completion results based on code history, which is something that has been successfully adopted by other IDEs. In this work I intend to study how code completion in Pharo can be improved using machine learning (ML) and statistical language models. In particular, the idea is to use the N-gram model, as it has been documented to be used for such a task [2], and, if successfully adapted to Pharo, might help enhance the quality of code completion.

There are many approaches to handling the process of completing code. The classic strategies use lexical [2] (completing based on the prefix the user is typing) or semantic (use code structure analysis information to tailor completions) models. In the latest years statistical and ML models have also gained popularity. For the most part when using ML, source code analysis is also often disregarded, and the results of simply training the model on the code base and inferring all the possible code dependencies are used instead.

[1]https://pharo.org/
[2]https://code.visualstudio.com/docs/editor/intellisense

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 RELATED WORKS

My work was inspired by the paper "On The Naturalness of Software" [2]. There, as well as in [1, 3–5], the structural information is not taken into consideration and the problem of improving code completion is reduced to a natural language processing problem, where predicting the next token is treated as predicting the next word in a sentence.

The n-gram language model approach proposed by Hindle et al. [2] is based on capturing statistical regularity at the n-gram level by taking n-1 previous tokens that are already entered into the text buffer, and attempting to guess the next token. Using this model, it is possible to estimate the most probable sequences of tokens and suggest the most relevant code completions to developers.

4 OUR APPROACH

What we want to do in Pharo is slightly different. Unlike the implementations of [1–5], where code completion is based solely on the results of statistical or machine learning models, the code completion in Pharo works by making use of AST information and giving contextually relevant results. However, as Pharo is a dynamically-typed language, type information is not always available, and the suggested results, after being sorted alphabetically, require more effort to go through and manually narrow down. This is where we think a sorting strategy based on an n-gram model will work better. In this case tokens will be suggested according to their probability of appearing in the already typed sequence, which will allow us to propose more accurate suggestions even without knowing the exact type of the token.

The estimated contributions of this work are the following:

- improve code completion in Pharo by sorting candidate completions with an n-gram language model
- build a tool based on a trained n-gram model that would propose completion fast enough to be used in an IDE

5 FUTURE WORK

We would like to have a methodology to numerically evaluate the results of code completion produced by different completion strategies. In particular, to have a robust estimation of the improvement of various n-gram models compared to the alphabetical completion sorting and among each other.

6 CONCLUSION

In this paper we proposed applying n-gram language models when sorting completion candidates in Pharo. Having an AST-based completion engine, we are not always able to infer type information from the nodes, and hope to compensate for it and improve the relevance of suggestions by implementing a fast n-gram based sorting functionality to be used in the Pharo IDE.

ACM Reference Format:
Myroslava Romaniuk. 2020. Improving Code Completion in Pharo Using N-gram Language Models. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

REFERENCES

[1] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 763–773.
[2] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 837–847.
[3] Jian Li, Yue Wang, Michael R Lyu, and Irwin King. 2017. Code completion with neural attention and pointer networks. arXiv preprint arXiv:1711.09573 (2017).
[4] Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In Acm Sigplan Notices, Vol. 49. ACM, 419–428.
[5] Zhaopeng Tu, Zhendong Su, and Premkumar Devanbu. 2014. On the localness of software. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 269–280.