Improving Code Completion in Pharo Using N-gram Language Models

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

ABSTRACT

In this paper I 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 the suggested completions.

1 INTRODUCTION

Code completion is one of the most essential features in any IDE; it is one of the first things a developer notices and it is something that can "make or break" the flow of productivity when coding. The accuracy and the speed with which the completions are suggested are paramount in helping make the completion as effective as possible. That is why researchers and software engineers are constantly trying to find ways to enhance completion engines in IDEs.

When it comes to code completion in the Pharo IDE[2], the situation is no different: over the past couple of years a lot of work has already been done to improve the completion engine there. However, I believe that it can only benefit from additionally improving the sorting strategies by using machine learning techniques.

2 CONTEXT

The current implementation of code completion in the Pharo IDE 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. We can then give contextually suitable completions, such as suggesting a class name for a global node, method names for a method node, and so on. However, once we get the list of suggestions, there is no efficient way to sort them, and so the desired completions can appear at the bottom of the list. This requires the developer to scroll down or type in more symbols and can be very unproductive.

In this work I intend to study how code completion in Pharo can be improved using 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 [3], and, if successfully adapted to Pharo, I believe it can help enhance the quality of code completion.

[1] https://pharo.org/

[2] Pharo is an object-oriented dynamically typed programming language inspired by Smalltalk, and the Pharo IDE is an interactive IDE intended for developing in Pharo.

<Programming'20> Companion, March 23–26, 2020, Porto, Portugal

ACM ISBN 978-1-4503-7507-8/20/03…$15.00
https://doi.org/10.1145/3397537.3398483

3 RELATED WORKS

There are many approaches to handling the process of completing code. The classic strategies used to rely on language-specific pattern matching, i.e. completing based on the prefix the developer has typed in. According to Bruch et al. [1], code completion engines up until 10 years ago would also mainly rely on type information, with no contextual analysis. In the paper, the authors countered this approach by implementing intelligent code completion systems, which learned from source code examples and gave much more contextually relevant results. In the latest years statistical and machine learning models have also gained popularity. In machine learning approaches, source code is often treated as regular text, and the contextual analysis is disregarded. IntelliSense[3] is an example of a code completion tool that uses semantic analysis, whereas TabNine[4] is an example of a tool using deep learning.

This work has been inspired by the paper "On The Naturalness of Software" [3]. There, as well as in [2, 4–6], 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. [3] 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 PROPOSED APPROACH

4.1 Implementation

What we want to do in Pharo is slightly different. Unlike the implementations of [2–6], 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:

[3] https://code.visualstudio.com/docs/editor/intellisense
[4] https://www.tabnine.com/

- 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

4.2 Evaluation

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 *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion), March 23–26, 2020, Porto, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3397537.3398483

REFERENCES

[1] Marcel Bruch, Martin Monperrus, and Mira Mezini. 2009. Learning from examples to improve code completion systems. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*. 213–222.
[2] 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.
[3] 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.
[4] 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).
[5] Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *Acm Sigplan Notices*, Vol. 49. ACM, 419–428.
[6] 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.