TUNA: TUning Naturalness-based Analysis

Matthieu Jimenez
University of Luxembourg
matthieu.jimenez@uni.lu

Maxime Cordy
University of Namur, Belgium
University of Luxembourg
maxime.cordy@unamur.be

Yves Le Traon
University of Luxembourg
yves.letraon@uni.lu

Mike Papadakis
University of Luxembourg
mike.papadakis@gmail.com

Since the seminal work of Hindle et al. [1] on software naturalness, natural language processing techniques were applied successfully to facilitate a number of software engineering tasks, such as code completion [1], bug prediction [2], etc. The key idea behind these techniques is to capture regularities of existing code into a language model, with the aim of inferring characteristics of previously unseen code based on how regular it looks according to the model. As an example, feeding a model with buggy code makes it capable of detecting new bugs, that is, the code containing the bugs will appear *natural* to the model. As for code completion, the language model can suggest the most-likely token to follow a given incomplete piece of code.

The most common types of language model are the n-gram models. They consist of a set of conditional probabilities of the form $P(t_k|t_{k-n+1}\dots t_{k-1})$, each of which records the probability that some token t_k follows a preceding sequence $t_{k-n+1}\dots t_{k-1}$. In the case of source code analysis, tokens are fragments of a given representation of the code, such as a lexical token or a node of the code's abstract syntax tree. To build an n-gram model for a given source code, one must (1) *tokenize* this code to obtain sequences of tokens; (2) parameterize the model (e.g. set the size n of the n-grams, specify the unknown threshold, choose a smoothing method); (3) train the model, that is, make it compute the conditional probabilities based on what it observes in the source code. Then, the model can be used to compute the *naturalness* of some new code, typically as a cross-entropy value obtained from the computed probabilities. A lower cross-entropy means that the new code is more natural, and thus more similar to the code used to train the model. By computing naturalness values, one can then perform valuable source code analyses (see [3] for an expanded view of what can be achieved).

However, in our related ICSME '18 paper [4], we have shown that the conclusions of a study can drastically change with respect to how the code is tokenized and how the used n-gram model is parameterized. These choices are thus of utmost importance, and one must carefully make them. To show this and allow the community to benefit from our work, we have developed TUNA (TUning Naturalness-based Analysis), a Java software artifact to perform naturalness-based analyses of source code. More precisely, TUNA provides multiple functionalities through the interaction of dedicated modules.

First, TUNA's module can retrieve Java source code contained in a public GitHub repository. As such, the module `tuna-gitUtils` includes a class `GitClonePull`, where one can specify a source repository and a destination folder before cloning or updating this repository.

Second, TUNA can tokenize Java source code based on multiple representations (e.g., as UTF8 tokens, as programming language grammar's lexical units, as sequences of nodes of the abstract syntax tree visited in depth-first or breadth-first order). To achieve this, the module `tuna-tokenizer` provide a factory named `JavaFileTokenizerFactory`, which provides methods to instantiate any tokenizer mentioned in our work [4]. For grammar-based or AST-based tokenization, it relies on JAVAPARSER [5] to parse the source code.

Third, one can parameterize n-gram models, train them based on tokenized source code, and compute the cross-entropy of one or more source files. To this end, the module `tuna-modelling` provides an interface `NgramModel` and an implementation of it, based on Kylm [6] and named `NgramModelKylmImpl`. Following the interface segregation principle, alternative implementations can easily be added in the future.

Finally, the module `experiment` contains the code needed to replicate our experiments reported in [4].

To the best of our knowledge, TUNA is the first open-source[1], end-to-end toolchain to carry out source code analyses based on naturalness. We continue to make it evolve as we perform additional studies. As such, other modules exist; we presented only the ones relevant for the purpose of our related paper [4].

REFERENCES

[1] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proceedings of ICSE '12*. Piscataway, NJ, USA: IEEE Press, 2012, pp. 837–847.

[2] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the "naturalness" of buggy code," in *Proceedings of ICSE '16*. New York, NY, USA: ACM, 2016, pp. 428–439.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *CoRR*, vol. abs/1709.06182, 2017.

[4] M. Jimenez, M. Cordy, Y. L. Traon, and M. Papadakis, "On the impact of tokenizer and parameters on n-gram based code analysis," in *Proceedings of ICSME 18*, 2018.

[5] J. Parser. (2017) Java parser github. [Online]. Available: https://github.com/javaparser/javaparser

[6] G. Neubig. (2017) Kyoto language modeling toolkit. [Online]. Available: https://github.com/neubig/kylm

[1] One can download it at https://github.com/electricalwind/tuna