Bringing Structure to Naturalness: On the Naturalness of ASTs

Profir-Petru Pârţachi
profir@nii.ac.jp
National Institute of Informatics
Tokyo, Japan

Mahito Sugiyama
mahito@nii.ac.jp
National Institute of Informatics/SOKENDAI
Tokyo/Kanagawa, Japan

ABSTRACT

Source code comes in different shapes and forms. Previous research has already shown code to be more predictable than natural language at the token level: source code can be natural. More recently, the structure of code — either as graphs or trees — has been successfully used to improve the state-of-the-art on numerous tasks: code suggestion, code summarisation, method naming *etc.* This body of work implicitly assumes that structured representations of code are similarly statistically predictable, *i.e.* natural. We consider that this view should be made explicit and propose directly studying the Structured Naturalness Hypothesis. Beyond just naming existing research that assumes this hypothesis and formulating it, we also provide evidence for tree representations: TreeLSTM models over ASTs for some languages, such as Ruby, are competitive with *n*-gram models while handling the syntax token issue highlighted by previous research 'for free'. For other languages, such as Java or Python, we find tree models to perform worse, suggesting that downstream task improvement is uncorrelated to the language modelling task. Further, we show how one may use naturalness signals for near state-of-the-art results on just-in-time defect prediction without manual feature engineering work.

KEYWORDS

Naturalness, Structure, AST, Self-Cross-Entropy

ACM Reference Format:
Profir-Petru Pârţachi and Mahito Sugiyama. 2024. Bringing Structure to Naturalness: On the Naturalness of ASTs. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643517

1 INTRODUCTION

Source code is highly repetitive; despite the flexibility that programming languages offer, developers tend to converge to similar constructions which lend to its statistical predictability [3]. In the seminal work on source-code *naturalness*, Hindle *et al.* [3] show that this predictability, which represents naturalness, lends itself well to statistical language models. This property of source-code has been successfully exploited for many tasks such as code synthesis [1], code completion [2], defect prediction [10] *etc.*

While it seems intuitive that naturalness should extend from the unstructured token domain into structures, which we refer to as *structured naturalness*, either via parsing, such as abstract syntax trees (ASTs), or by static or dynamic analysis, such as control-flow graphs, data-flow graphs, call graphs *etc.*, this assumption has not been studied directly. Further, it is an open question if such tasks help with the language modelling task or specifically the downstream tasks for which such views are employed.

We propose that the structured naturalness hypothesis should be the primary subject of study rather than an unstated assumption as the current body of literature would have it. By better knowing the nature of the statistical regularities induced in structured views of code from the underlying naturalness of the source code itself, we can better tune statistical approaches that seek to use it either as an aid or directly as a signal.

2 METHODOLOGY

In this study, we focus on TreeLSTMs [7, 9] as an avatar for a tree model: a language model that incorporates syntax structure directly. Before this work, tree models were not actively and directly researched from a "naturalness" perspective.

To analyse tree-model-based entropy, we estimate the probability of an output token given a bilateral context and the associated partial AST. We consider this probability to be an analogue for the masked token training typical of many language models, for example in the pre-training of transformers. We estimate this probability using a standard Child-Sum TreeLSTM [9].

To estimate self-cross-entropy for the naturalness experiment, we borrow the cross-fold-validation methodology of Hindle *et al.* [3]. Unlike Hindle *et al.*, we restrict ourselves to fewer folds due to the computational cost of neural models: we split the data into five groups, train on four of these and evaluate on the held-out set. While Hindle *et al.* employ an *n*-gram model and can explore the impact of *n* on naturalness, we do not have a direct analogue. Therefore we use context length as a proxy for this parameter.

As we aim to study the self-cross-entropy of a reasonable model, we use the hyper-parameter defaults from NLP literature [7]: embedding size — 150, LSTM hidden layer — 75, feed-forward hidden layer — 25, batch size — 8 (64 for JavaScript and Ruby), learning rate — 0.025, weight decay — 0.0001, using an ADAM optimiser. To generate a training corpus, we start from CodeSearchNet [4], which we pre-process using ANTLRv4 [1] and their example grammars[2]. To obtain an analogue for *n*-gram length, we filter examples by context length for the following lengths: $\{20, 25, 30, 40, 50, 60\}$.

In the second experiment, we focus on indirect, but applicable evidence for the hypothesis. Previously, Ray *et al.* [6] have shown a correlation between higher entropy and defect-proneness. This

[1]https://github.com/antlr/antlr4
[2]https://github.com/antlr/grammars-v4

Table 1: Self-cross-entropy of TreeLSTM for different limits of the sequence length across languages.

Language	20	25	30	{40, 50, 60}
go	12.35	12.13	12.11	12.09
Java	13.66	13.17	13.13	13.13
JavaScript	-	12.28	11.53	11.48
Python	-	-	14.04	13.97
Ruby	9.04	8.60	8.46	8.45

Table 2: The self-cross-entropy of an *n*-gram model and TreeLSTM model on ANTLRv4 ASTs.

Language	*n*-gram [5]	TreeLSTM (ANTLRv4 AST)
Java	8.0	13.1
JavaScript	7.0	11.5
Python	9.5	14.0
Ruby	9.0	8.5

suggests a potential signal for our experimental setup. We study if exploiting such a signal can provide a model for JIT Defect Prediction. We show that a simple pipeline can get near SOTA results without manual feature engineering by using AST regularities. We borrow the dataset construction from Sohn *et al.* [8] and evaluate on Apache Commons Lang/Math with the extra step of pre-processing changed methods into ASTs using ANTLRv4. For classification, we employ two Random Forest Classifiers trained over an AST embedding to obtain a buggy and clean score both used for the final classification.

3 SELF-CROSS ENTROPY OF AST TREES

Table 1 shows the self-cross-entropy for different languages and sequence length limits. While lacking in the granularity of data points compared to *n*-gram results, we still observe a shift towards lower entropy as we increase the context size. Languages tend to show this transition around $k = 30$. While we lack data points for $k > 30$ for Java and $k > 40$ for Go and Python, as the transition happens at or before $k = 30$, we expect the final data point to be representative of these languages, hence grouped in the table. Compared to English under an *n*-gram model, whose self-cross-entropy is known to be around 12 bits, two languages show more predictability/lower self-entropy by being well under that, Go shows a borderline result by being a smidgen above 12 bits, while Java and Python show less predictability. This hints that the hypothesis should be checked before a model is employed with a particular structure as using an incorrect structural inductive bias/prior can misguide a model.

Focusing on JavaScript, Java, Ruby, and Python which overlap with languages studied by Rahman *et al.* [5] (which are shown in Table 2), we find mixed results in terms of predictability; however, in line with the results of Rahman *et al.* after they account for SyntaxTokens. We observe that Ruby is marginally better at 8.5 bits *vs* 9 bits when viewed as a tree, while JavaScript suffers in predictability at 11.5 bits when viewed as a tree *vs* 7 bits when viewed as an *n*-gram. For Java, we find that 13.1 *vs* 8 bits, while Python 14.0 *vs* 9.5 bits, hinting that the ASTs obtained from ANTLRv4 are a bad fit for the language model task explored.

4 JUST-IN-TIME DEFECT PREDICTION RESULTS

The best reported F1 by Sohn *et al.* on Apache Commons Lang is 0.199 while our approach reports 0.157. While balanced accuracy is 0.816 for us and 0.596 for Sohn *et al.* On Apache Commons Math, we find that the F1 is 0.401 for Sohn *et al.* while we obtain 0.194, a worse result when considered directly; accuracy, meanwhile, is 0.769 for us *vs* 0.690 for Sohn *et al.*

These results show that our approach guided by intuition from structured naturalness enables a competitive and simple approach to just-in-time defect prediction; *crucially, we avoid manually constructing a feature space for the problem and instead exploit statistical properties of structures from the populations of interest.* The design space, however, is still open for investigation, either from a better understanding of structured naturalness or combining more traditional features employed in defect prediction literature.

5 CONCLUSION

In this paper, we propose to study the structured naturalness hypothesis directly and consider AST trees. We find that on the language modelling task, for most languages, the self-cross-entropy is higher than a similar *n*-gram model; however, for downstream applications, such as JIT Defect Prediction, there is sufficient signal to reach near SOTA performance.

Acknowledgements: This work was supported by JST, CREST Grant Number JPMJCR22D3, Japan, and JSPS KAKENHI Grant Number JP21H03503.

REFERENCES

[1] Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei. 2015. Bimodal Modelling of Source Code and Natural Language. In *Proc. of ICML (JMLR Workshop and Conference Proceedings, Vol. 37)*, Francis R. Bach and David M. Blei (Eds.). JMLR.org, 2123–2132.

[2] Pavol Bielik, Veselin Raychev, and Martin T. Vechev. 2016. PHOG: Probabilistic Model for Code. In *Proc. of ICML (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 2933–2942.

[3] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 837–847.

[4] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *ArXiv preprint* abs/1909.09436 (2019).

[5] Musfiqur Rahman, Dharani Palani, and Peter C Rigby. 2019. Natural software revisited. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 37–48.

[6] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the "naturalness" of buggy code. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 428–439.

[7] Yusuke Shido, Yasuaki Kobayashi, Akihiro Yamamoto, Atsushi Miyamoto, and Tadayuki Matsumura. 2019. Automatic source code summarization with extended tree-lstm. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[8] Jeongju Sohn, Yasutaka Kamei, Shane McIntosh, and Shin Yoo. 2021. Leveraging Fault Localisation to Enhance Defect Prediction. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 284–294.

[9] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proc. of ACL*. Association for Computational Linguistics, Beijing, China, 1556–1566.

[10] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 297–308.