Are NLP Metrics Suitable for Evaluating Generated Code?

Riku Takaichi[1(✉)], Yoshiki Higo[1], Shinsuke Matsumoto[1], Shinji Kusumoto[1], Toshiyuki Kurabayashi[2], Hiroyuki Kirinuki[2], and Haruto Tanno[2]

[1] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka, Japan
r-takaic@ist.osaka-u.ac.jp
[2] Nippon Telegraph and Telephone Corporation, Minato, Tokyo, Japan

Abstract. Code generation is a technique that generates program source code without human intervention. There has been much research on automated methods for writing code, such as code generation. However, many techniques are still in their infancy and often generate syntactically incorrect code. Therefore, automated metrics used in natural language processing (NLP) are occasionally used to evaluate existing techniques in code generation. At present, it is unclear which metrics in NLP are more suitable than others for evaluating generated codes. In this study, we clarify which NLP metrics are applicable to syntactically incorrect code and suitable for the evaluation of techniques that automatically generate codes. Our results show that METEOR is the best of the automated metrics compared in this study.

Keywords: Automated metric · Code generation · Deep learning

1 Introduction

Code generation is a technique that generates program source code without human intervention. It significantly changes the software process and is known as a promising way to reduce the burden of programming on developers [14]. In recent years, there has been much research on automated methods for writing code, such as code generation [1,4,14]. In these studies, automated metrics (hereinafter, referred to it simply as "metrics") are used to evaluate generated code, and several metrics for code evaluation have already been proposed [13,15]. These metrics use abstract syntax trees or program dependency graphs, assuming that code is syntactically correct. However, research on code generation is still in its infancy, and it is not uncommon for syntactically incorrect code to be generated. For example, 7.0 % of code generated by SNM [14] and 90 % of code by Coarse-to-Fine [4], which are recently proposed code generation models, are syntactically incorrect. Therefore, metrics for code that assume that generated code is syntactically correct may not be usable.

In some cases, metrics used in natural language processing (NLP) are used to evaluate the code generation techniques in place of metrics for code [12]. For

D. Taibi et al. (Eds.): PROFES 2022, LNCS 13709, pp. 531–537, 2022.
https://doi.org/10.1007/978-3-031-21388-5_38

example, BLEU is frequently used to evaluate the quality of generated code. However, this metric has limitations when used in code evaluation [5,13]. It is thus still unclear which metrics are suitable for evaluating code without assuming the syntactic correctness of the code.

In this study, we clarify which metrics are suitable for evaluating code, and in particular, can be applied to syntactically incorrect code. More specifically, we focus on the code generation task by deep learning, and clarify which metrics are suitable for evaluating code that are automatically generated from requirements written in natural language.

Currently, it is difficult to generate complete code from requirements described using natural language [7]. When using code generation, human modification of the generated code is required to make the code meet the requirements. Therefore, it is desirable that the generated code be easy to modify into code that satisfies the requirements. The suitability of metrics for generated code can be evaluated by the ease with which the generated code can be modified into code that satisfies the requirements.

In this study, we measure the ease of modifying various examples of generated code into code that satisfies given requirements. The results suggest that METEOR is the best metric that correlates with the ease of modifying generated code and is thus the most suitable for evaluating code created via code generation [3].

2 Research Questions

RQ1: Which metrics can be used to evaluate the ease of modifying generated code in terms of modification time?

The ease of modifying generated code can be evaluated in terms of the time a developer take to modify it. To evaluate the ease of modification of generated code in terms of modification time, we investigate which metrics can evaluate the ease of modification. More specifically, we examine the correlation between the time it takes a developer to modify generated code into code that satisfies their requirements and the evaluation values of metrics.

RQ2: Which metrics can be used to evaluate the ease of modifying generated code in terms of the size of changes to the code needed to modify it?

The ease of modification of generated code can also be evaluated by the amount of modification of the code by a developer. When evaluating the ease of modification of generated code by the amount of modification, we investigate which metrics can evaluate the ease of modification. As in RQ1, we examine the correlation between the amount of modification and the evaluation values of metrics, where the strongest correlation between these is considered indicative of ease of modification. In this study, the amount of modification is defined as the number of tokens to modify the generated code.

3 Background

3.1 Code Generation

Code generation is a method by which source code is written automatically. It can be classified in terms of the following elements:

- Input, for example, requirements written in natural language [7], DSL [10], or input/output examples [9].
- Approach, for example, translation-based [7] or search-based [11].

This study focuses on translation-based code generation using deep learning, which takes requirements written in natural language as input.

3.2 Edit Distance

The edit distance is the minimum number of edits (insertions, deletions, or substitutions) required to make one sequence X equivalent to another sequence Y.

The normalized edit distance (NED) between X and Y is computed as

$$\mathrm{NED}(X,Y) = \frac{\mathrm{EditDistance}(X,Y)}{\max\left(\mathrm{length}(X), \mathrm{length}(Y)\right)}$$

where $\mathrm{EditDistance}(X,Y)$ is the edit distance between the sequence X and Y. Here, $\mathrm{length}(S)$ refers to the length of the sequence S. The value of normalized edit distance is between 0 and 1. In this study, the edit distance is calculated by considering the code as a sequence of tokens.

3.3 Metrics

Metrics are used for the automated evaluation of the quality of translation results. Ideally, automated evaluations should correlate highly with human evaluation because metrics are meant to be a feasible alternative to human evaluation. The metrics used in this study are as follows:

BLEU [8] is an metric for evaluating the quality of natural language machine translation results. It is calculated using the n-gram of two sequences.

STS [13] is calculated using the edit distance.

ROUGE-L [6] is calculated using the length of the longest common subsequence.

METEOR [3] is an metric for evaluating the quality of automated translation results in the field of NLP [2]. In this study, a code was regarded as English text because a code is usually written using English words.

These metrics are between 0 and 1. A higher value means a higher evaluation. BLEU, STS, and ROUGE-L were selected because there are studies that used them to evaluate code [12,13]. METEOR was selected because it is designed to address BLEU's weaknesses [2].

4 Experiment

We conducted an experiment to measure the ease of modifying code created with a code generation model to code satisfying given requirements. The ease of modification we measured involves either the modification time or the modification amount. The higher these, the lower the ease of modification. The amount of modification is measured by the normalized edit distance between generated code and modified code. We also examine the correlation between the ease of modification and evaluations of the generated code using metrics. The stronger the negative correlation, the more suitable the automated evaluation value is for evaluating code generated from requirements described in natural language.

4.1 Code Generation Model

We created a code generation model using a deep neural network for NLP available on GitHub[1]. The code generation model was trained on the dataset ReCa [7] comprising requirement text, correct code, and test cases used in programming contests. The dataset includes 5,149 requirements and 16,673 Python code. The code generation model was trained using 300 data entries for testing, 200 for validation, and the remainder for training.

The input of the model is text that has been preprocessed with lowercasing, lemmatization, and removing stopwords. The original text before preprocessing is requirement text written in English. The output of the model is tokens of Python code. It can be automatically transformed into actual Python code. The generated code may not satisfy the requirement described in the input text. The correct code satisfies the requirement and passes the test cases.

4.2 Measuring Ease of Modification

We conducted an experiment with human subjects to measure the ease of modification of generated code. In this experiment, 10 data entries were randomly sampled from the 300 test data. The sampled data have an average of 53.4 test cases per requirement. The subjects were 11 people, one associate professor and ten students. Each subject had a different skill level in Python. A cheat sheet with the code that might be needed when modifying the generated code was supplied for the subjects who were less skilled. Each subject experiments with the 10 sampled data. The experimental steps are as follows:

STEP-1 [Understanding Requirements]. Subjects receive the requirements text and test cases. They understand the requirements by reading the text.

STEP-2 [Modifying Generated Code]. Subjects receive the generated code. They modify the generated code to satisfy the requirements given in STEP-1.

STEP-3 [Testing]. Subjects check whether the code modified in STEP-2 passes all the test cases given in STEP-1. If it passes, STEP-3 is completed. Otherwise, they return to STEP-2 to modify the generated code once more.

[1] https://github.com/nazim1021/neural-machine-translation-using-gan.

We count the seconds from STEP-2 to the end of STEP-3 and took this value to be the time developers took to modify the generated code. In the above steps, the modified generated code is called "modified code". We cannot obtain both the modification time and the modified code if the subject cannot successfully modify the generated code so that the requirements are met.

4.3 Results

Table 1 lists the Pearson's correlation coefficients between the evaluation value of each metric and the ease of modification (such as the modification time and the modification amount), with p-values. According to the results in Table 1, the correlation between the evaluation and the modification time and modification amount is strongest for METEOR. However, it is only weakly correlated with modification time.

Table 1. Correlation between metrics and the ease of modification

Metric	RQ1: the modification time		RQ2: the amount of modification	
	COR	p-value	COR	p-value
BLEU	−0.181	0.117	−0.392	4.64×10^{-4}
STS	−0.100	0.389	−0.555	1.99×10^{-7}
ROUGE-L	0.011,5	0.921	−0.481	1.08×10^{-5}
METEOR	−0.251	0.028,6	−0.696	3.03×10^{-12}

Answer to RQ1 and RQ2: Among the examined metrics, METEOR is the best metric to evaluate the ease of modifying generated code in terms of the modification time and the modification amount. In addition, BLEU, which is widely used to evaluate generated code, is not a good metric in these context.

5 Conclusion

The purpose of this study was to clarify which NLP metrics can be applied to syntactically incorrect code. We investigated which metrics strongly correlate with the evaluation values obtained in the experiment with subjects. The results of the study showed that METEOR has a relatively strong correlation with both amount of modification and the time required to modify code created by code generation to meet the given requirements. We conclude that METEOR is a better metric for generated code than the frequently used BLEU. However, metrics may not be suitable for evaluating generated code because of its weak correlation with the modification time. In addition, note that there are limitations in applying these results to real projects because the subject experiment in this study was conducted using programming contest data.

For future research, we are going to examine the evaluation values that correlate stronger with the coding time reduced by using the generated code. This is why we plan to compare the time required for subjects to read the requirements and write a program with the time required for them to modify generated code to satisfy the requirements.

Acknowledgements. This research was supported by JSPS KAKENHI, Japan (grant numbers JP20H04166, JP21K18302, JP21K11820, JP21H04877, JP22H03567, and JP22K11985).

References

1. Ahmad, W., Chakraborty, S., Ray, B., Chang, K.W.: Unified pre-training for program understanding and generation. In: Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2021)
2. Banerjee, S., Lavie, A.: METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proceedings of ACL Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (2005)
3. Denkowski, M., Lavie, A.: Meteor universal: language specific translation evaluation for any target language. In: Proceedings of Workshop on Statistical Machine Translation (2014)
4. Dong, L., Lapata, M.: Coarse-to-Fine decoding for neural semantic parsing. In: Proceedings of Annual Meeting of the Association for Computational Linguistics (2018)
5. Karaivanov, S., Raychev, V., Vechev, M.: Phrase-based statistical translation of programming languages. In: Proceedings of ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (2014)
6. Lin, C.Y.: ROUGE: a package for automatic evaluation of summaries. In: Proceedings of ACL Text Summarization Branches Out (2004)
7. Liu, H., Shen, M., Zhu, J., Niu, N., Li, G., Zhang, L.: Deep learning based program generation from requirements text: are we there yet? IEEE Trans. Softw. Eng. **48**(4), 1268–1289 (2022)
8. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of Annual Meeting of the Association for Computational Linguistics (2002)
9. Parisotto, E., Mohamed, A., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis. In: Proceedings of International Conference on Learning Representations (2017)
10. Rabinovich, M., Stern, M., Klein, D.: Abstract syntax networks for code generation and semantic parsing (2017). https://arxiv.org/abs/1704.07535
11. Spector, L.: Autoconstructive evolution: Push, PushGP, and Pushpop. In: Proceedings of Genetic and Evolutionary Computation Conference (2001)
12. Svyatkovskiy, A., Deng, S.K., Fu, S., Sundaresan, N.: Intellicode compose: code generation using transformer. In: Proceedings of ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2020)

13. Tran, N., Tran, H., Nguyen, S., Nguyen, H., Nguyen, T.: Does BLEU score work for code migration? In: Proceedings of IEEE/ACM International Conference on Program Comprehension (2019)
14. Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. In: Proceedings of Annual Meeting of the Association for Computational Linguistics (2017)
15. Zhao, G., Huang, J.: Deepsim: deep learning code functional similarity. In: Proceedings of ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2018)