4-2024

Classifying source code: How far can compressor-based classifiers go?

Zhou YANG
Singapore Management University, zhouyang.2021@phdcs.smu.edu.sg

Classifying Source Code: How Far Can Compressor-based Classifiers Go?

Zhou Yang
Singapore Management University
Singapore
zyang@smu.edu.sg

ABSTRACT

Pre-trained language models of code, which are built upon large-scale datasets, millions of trainable parameters, and high computational resources cost, have achieved phenomenal success. Recently, researchers have proposed a *compressor-based classifier* (Cbc); it trains no parameters but is found to outperform BERT. We conduct the first empirical study to explore *whether this lightweight alternative can accurately classify source code.* Our study is more than applying Cbc to code-related tasks. We first identify an issue that the original implementation overestimates Cbc. After correction, Cbc's performance on defect prediction drops from 80.7% to 63.0%, which is still comparable to CodeBERT (63.7%). We find that hyperparameter settings affect the performance. Besides, results show that Cbc can outperform CodeBERT when the training data is small, making it a good alternative in low-resource settings.

KEYWORDS

Defect Software Prediction, Robustness, Efficient Learning

ACM Reference Format:
Zhou Yang. 2024. Classifying Source Code: How Far Can Compressor-based Classifiers Go? . 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, 3 pages. https://doi.org/10.1145/3639478.3641229

1 INTRODUCTION

As investigated by a recent survey [8], pre-trained language models (e.g., CodeBERT [5]) have demonstrated promising results in a variety of code-related tasks, including defect prediction. What is behind the success is the large-scale datasets, millions of trainable parameters, and high computational resources cost. For example, CodeBERT is a model with 125 million parameters trained on 8.5 million datapoints with 16 NVIDIA Tesla V100 GPUs. Even fine-tuning it on a small dataset (e.g., Devign [19] with 21,854 examples) takes 5 hours on a GTX 2080 GPU. Researchers [15] point out that such a large model is not suitable to be deployed in modern IDEs, encouraging us to explore lightweight alternatives.

Recently, Jiang et al. [9] propose a *compressor-based classifier* (Cbc), which requires no parameter to be trained. Jiang et al. [9] report promising results: Cbc outperforms or is comparable to BERT [3] in a variety of NLP datasets. Cbc is also faster than pre-trained models: we find it takes only 10 mins to finish the evaluation on Devign [19] dataset using a desktop CPU. We conduct the first empirical study to evaluate Cbc on code-related tasks.

In this study, we first point out a potential issue in the Cbc implementation that overestimates the model performance. After correction, the accuracy of Cbc decreases from 80.7% to 63.0% on a defect prediction task, but it is still comparable to CodeBERT (63.7%), even higher than other models like CodeTrans (63.03%) [4] and CoTexT (61.48%) [13]. Second, we further show that the hyperparameter settings affect the performance. We evaluate two strategies to break the tie in the kNN classifier: (1) random selection and (2) decrement k until a tie is broken; the former is found to be more effective via a statistical test. Besides, the accuracy tends to increase when k increases. Third, we find that Cbc can outperform CodeBERT when the training data is small, making it a good alternative in low-resource settings, in terms of both computational resources and training data.

2 BACKGROUND AND RELATED WORK

2.1 Compressor-based Classifier

In a nutshell, (Cbc) consists of three main components: (1) a lossless compressor `gzip` [14], a compressor-based distance metric, and a k-Nearest-Neighbor classifier. The compressor first compresses the inputs, aiming to represent the inputs with as less number of bits as possible. Considering three inputs x_1, x_2, and x_3, where x_1 and x_2 share the same label (e.g., both are defective), and x_3 has a different label (e.g., non-defective). Let $C(x_1)$ represent the length of the compressed x_1 and $C(x_1x_2)$ represent the length of the compressed x_1 and x_2 concatenated. $C(x_1x_2) - C(x_1)$ can is the *additional bits required to encode x_2 given x_1*. Intuitively, $C(x_1x_2) - C(x_1)$ is expected to be smaller than $C(x_1x_3) - C(x_1)$, since x_1 and x_2 share more common information than x_1 and x_3. Jiang et al. [9] utilize another metric called *normalized compression distance* (NCD) as a more precise approximation of the '*information distance*' between two inputs, which is formally defined as:

$$\mathrm{NCD}(x_1, x_2) = \frac{C(x_1x_2) - \min(C(x_1), C(x_2))}{\max(C(x_1), C(x_2))} \quad (1)$$

The smaller the NCD, the more likely x_1 and x_2 share the same label. Then, a kNN classifier is used to classify a test input. It computes the NCD between a test input and all the training inputs to identify the k nearest neighbors. The label of the test input is determined by the majority of the labels of the k nearest neighbors. In the original implementation, k is set as 2.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3641229

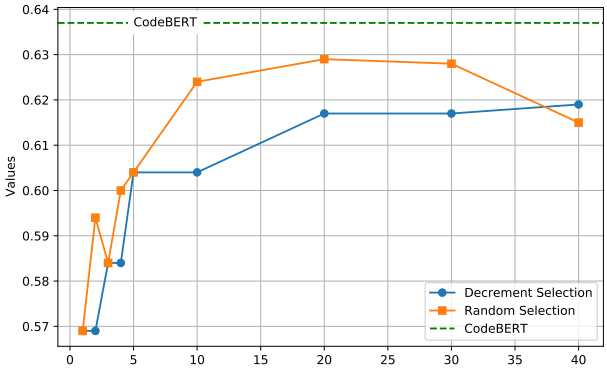

Figure 1. The classification accuracy of Cbc under different k values and tie breaking strategies.

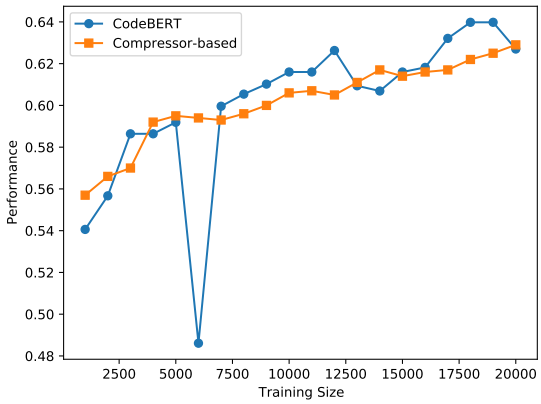

Figure 2. How the performance of Cbc and CodeBERT change when the training data size varies.

2.2 Related Work

A series of empirical studies [10] have shown the strong performance of pre-trained language models of code in a variety of code-related tasks, including code search, code summarization, defect prediction, etc. For example, one of the most commonly evaluated model is CodeBERT [5], which is an encoder-only model that demonstrates strong performance in classifying source code. Given an input (i.e., code snippet), it first converts the input into a vector called *code embeddings* and then uses a classifier to predict the label. Other relevant models include GraphCodeBERT [7], PLBART [1], CodeT5 [18], etc. We choose an important and popular task, defect prediction [19], to evaluate Cbc. We refer to a recent survey [8] for a comprehensive review of pre-trained language models of code.

The compressor-based classifier we evaluate falls into a category of works that use a compressor to approximate the distance between two inputs [2, 9, 12]. Another line of work [6, 11] uses a compressor to estimate entropy based on Shannon Information Theory for text classification. The classifier evaluated in our study is the most recent work and demonstrates promising results.

3 EMPIRICAL STUDY AND RESULTS

This paper presents the first empirical study to understand whether Cbc can generalize to software engineering tasks. We use a defect prediction dataset called Devign [19], consisting of 21,854 training and 2,732 testing examples. More than applying Cbc to this dataset, we conduct analyses not considered in the original study [9].

Correcting the Evaluation Metric. We find that the results reported in the original paper [9] might be overestimated and not achievable in practice. Specifically, they choose $k = 2$ in the kNN classifier and assumes that the classifier can always choose the correct label if a tie happens (i.e., the one neighbor has label 1 and the other has label 0). This finding is also confirmed in the discussion between the authors and interested users of Cbc [17]. To evaluate Cbc in a more realistic setting, we implement the kNN classifier with multiple k values and use two tie breaking strategies: (1) random selection and (2) decrement k until a tie is broken.

We first run Cbc with the original implementation [16] and obtain a high test accuracy of 80.7%, much higher than the accuracy achieved by CodeBERT (63.7%). We vary the k value and use two tie breaking strategies to simulate a more realistic evaluating setting. The results are shown in Figure 1. We observe that the accuracy of Cbc significantly drops. When $k = 2$ and using random selection to break the tie, the accuracy is 59.4%, lower than CodeBERT. When k increases, which generally means that the classification is based on a broader set of data points, the accuracy tends to increase. We also observe that the random selection strategy seems to be more effective than the decrement strategy. To validate this hypothesis, we conduct a paired t-test and find that the difference is statistically significant ($p < 0.05$). When $k = 20$ and using random selection, the accuracy is 63.0%, comparable to CodeBERT.

Data Size Analysis. Pre-trained models usually require much training data to achieve a good performance, which also consumes many computational resources on specialized hardware, i.e., GPUs. We analyze how the performance of two models changes when the size of training data varies. Figure 2 shows the results. The data points for Cbc is obtained using the optimal hyperparameter setting found in the previous analysis (i.e., $k = 20$ and random selection). We can observe a trend for both models that the accuracy increases when more data is used. However, Cbc has a more stable increase than CodeBERT. Due to the random nature of deep learning, the accuracy of CodeBERT fluctuates more than Cbc. When the training data is small (less than 6,000), Cbc achieves a better performance, indicating that Cbc can be a good alternative in low-resource settings, in terms of both computational resources and training data.

4 CONCLUSION AND FUTURE WORK

This paper presents the first empirical study on applying compressor-based classifiers to the defect prediction task. After correcting the evaluation and comparing it with CodeBERT, we find that Cbc can achieve a comparable performance to pre-trained models of code. Further results show that Cbc can be a good alternative in low-resource settings, where the training data is small and computational resources are limited. We hope our study can inspire more research on exploring lightweight alternatives to pre-trained models of code. In the future, we plan to extend the study, including evaluating generalizability on more datasets, analyzing how different compressors affect the performance, etc.

REFERENCES

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 2655–2668.

[2] David Pereira Coutinho and Mario AT Figueiredo. 2015. Text classification using compression-based dissimilarity measures. *International Journal of Pattern Recognition and Artificial Intelligence* 29, 05 (2015), 1553004.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[4] Ahmed Elnaggar, Wei Ding, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Silvia Severini, Florian Matthes, and Burkhard Rost. 2021. CodeTrans: Towards Cracking the Language of Silicon's Code Through Self-Supervised Deep Learning and High Performance Computing. arXiv:2104.02443 [cs.SE]

[5] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 1536–1547.

[6] Eibe Frank, Chang Chui, and Ian H Witten. 2000. Text categorization using compression models. (2000).

[7] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu andz Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

[8] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE]

[9] Zhiying Jiang, Matthew Yang, Mikhail Tsirlin, Raphael Tang, Yiqin Dai, and Jimmy Lin. 2023. "Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 6810–6828. https://doi.org/10.18653/v1/2023.findings-acl.426

[10] Hong Jin Kang, Tegawendé F. Bissyandé, and David Lo. 2020. Assessing the Generalizability of Code2vec Token Embeddings. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 1–12. https://doi.org/10.1109/ASE.2019.00011

[11] Nitya Kasturi and Igor L Markov. 2022. Text Ranking and Classification using Data Compression. In *I (Still) Can't Believe It's Not Better! Workshop at NeurIPS 2021*. PMLR, 48–53.

[12] Yuval Marton, Ning Wu, and Lisa Hellerstein. 2005. On compression-based text classification. In *Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings 27*. Springer, 300–314.

[13] Long Phan, Hieu Tran, Daniel Le, Hieu Nguyen, James Anibal, Alec Peltekian, and Yanfang Ye. 2021. CoTexT: Multi-task Learning with Code-Text Transformer. arXiv:2105.08645 [cs.AI]

[14] GNU Project. 2023. *GNU Gzip*. https://www.gnu.org/software/gzip/

[15] Jieke Shi, Zhou Yang, Bowen Xu, Hong Jin Kang, and David Lo. 2023. Compressing Pre-Trained Models of Code into 3 MB *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 24, 12 pages. https://doi.org/10.1145/3551349.3556964

[16] GitHub Users. 2023. *npc_gzip*. https://github.com/bazingagin/npc_gzip

[17] GitHub Users. 2023. *Problem with accuracy calculation?* https://github.com/bazingagin/npc_gzip/issues/3

[18] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*.

[19] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. *Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks*. Curran Associates Inc., Red Hook, NY, USA.