Towards Using Data-Influence Methods to Detect Noisy Samples in Source Code Corpora

Anh T. V. Dau*
Thang Nguyen-Duc
Hoang Thanh-Tung
{anhdtv7,thangnd34,tunght18}@fsoft.com.vn
FPT Software AI Center, Viet Nam

Nghi D. Q. Bui*
dqnbui.2016@smu.edu.sg
School of Information Systems
Singapore Management University

ABSTRACT

Despite the recent trend of developing and applying neural source code models to software engineering tasks, the quality of such models is insufficient for real-world use. This is because there could be noise in the source code corpora used to train such models. We adapt *data-influence methods* to detect such noises in this paper. Data-influence methods are used in machine learning to evaluate the similarity of a target sample to the correct samples in order to determine whether or not the target sample is noisy. Our evaluation results show that data-influence methods can identify noisy samples from neural code models in classification-based tasks. This approach will contribute to the larger vision of developing better neural source code models from a *data-centric* perspective, which is a key driver for developing useful source code models in practice.

ACM Reference Format:
Anh T. V. Dau*, Thang Nguyen-Duc, Hoang Thanh-Tung, and Nghi D. Q. Bui*. 2022. Towards Using Data-Influence Methods to Detect Noisy Samples in Source Code Corpora. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Research in the area of *Deep Learning for Code* [1–3, 8, 10, 11, 18] mostly relies on a large code corpus of code that allows deep learning methods to reason about source code properties. However, the real-world usage of such code models is still limited due to their quality. We observe that the source code data used for training code models is collected using a variety of heuristics, such as commit messages, tags provided by code competition websites [4, 5] and people tend to assume that the label of such data is accurate, despite the fact that it may contain a lot of noise [5]. There are many kinds of noise for classification-based tasks, but in our work, mislabeled examples are referred to as noisy examples.

We find that the majority of code learning research focuses on improving performance from a *model-centric* standpoint. This means that the dataset will remain constant while new models are being proposed to improve performance. There are recent efforts to analyze or propose methods to create high-quality datasets for software engineering [6, 9, 16, 17, 19] from the *data-centric* standpoint. In the data-centric approach, the model remains fixed while the quality of the datasets used to train such model gets improved. [6, 16] used simple rules to filter noise. Rule-based methods do not scale well to large and complex datasets. [17] proposed a learning-based approach for measuring the alignment between code and text for code search data. In this paper, we approach the problem by adapting data-influence methods [14] to detect noises and propose some strategies to enhance the quality of datasets. By identifying noisy samples, we hope to improve the source code model's performance. Results from learning theory also suggest that models trained on clean datasets converge faster and are more robust [13]. Data influence methods calculate the influence of training examples on model predictions. They track the changes in loss at test data points whenever the training example of interest is used. Finally, these methods will provide an influence score. This score indicates whether the sample is noisy or not.

In this work, we concentrate on classification-based tasks, including code classification and defect prediction. For these two tasks, our evaluation shows that the data-influence methods can successfully identify a large number of noisy samples in the training dataset. Furthermore, retraining the models on good training data improves the models' performance and robustness.

2 TECHNICAL DETAILS

A source code model trained on a training set Z_{train} is denoted as $M(\cdot;\theta)$. Assuming Z_{train} contains some noise, our goal is to identify the set of noisy samples $Z_{noise} \subset Z_{train}$. By removing Z_{noise}, we get a new training set Z_{clean}. Retraining M on Z_{clean} results in a new model M_{clean}. We use the validation set Z_{val} as the anchor to detect noise and select from Z_{val} a set of correctly labeled samples by using M. A sample is considered correct if the prediction from the model M match its label with high confidence. We call this set of correctly labeled samples Z_{gold}.

Now, we introduce the data-influence methods. We focus on Influence Function (IF) [7] and TracIn [15] as they are currently state-of-the-art techniques. IF [1] [7] estimates the influence of a sample $Z^{(i)}_{train}$ on the model M by measuring the influence score

*Equal Contributions

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00

[1] TracIn [15] follows the same principle. Readers are encouraged to read the original paper to check the details of TracIn's formula.

Table 1: Results on identifying the mislabeled samples on Synthetic Noisy dataset.

	Method	$k = 1$	$k = 5$	$k = 10$
ASTNN	IF	94.20 ± 3.72	84.88 ± 2.26	59.71 ± 0.32
	TracIn	91.09 ± 5.06	79.96 ± 1.29	55.97 ± 1.14
CodeBERT	IF	64.15 ± 1.07	31.50 ± 1.28	14.92 ± 1.33
	TracIn	72.36 ± 2.39	49.30 ± 1.33	35.36 ± 1.15

in accordance with the change of the loss at $Z^{(j)}_{gold}$ when removing a training sample $Z^{(i)}_{train}$ from the training set.

We then present the pipeline for our evaluation with data-influence methods as the key component to detect noisy samples.

(1) Initially, we train the model on training set Z_{train}, result in model $\boldsymbol{M}$. We use $\boldsymbol{M}$ to randomly select N correctly predicted samples from the validation set Z_{val}, resulting in Z_{gold}.
(2) For each sample in Z_{train}, we compute the influence score with all samples in Z_{gold}. The score for each training sample is: $S(Z^{(i)}_{train}, Z_{gold}) = \sum_{j=1}^{N} S(Z^{(i)}_{train}, Z^{(j)}_{gold})$. A negative score shows that $Z^{(i)}_{train}$ has bad influence on the model $\boldsymbol{M}$, which means that $Z^{(i)}_{train}$ is likely to be mislabeled.
(3) The top k% samples with the lowest score, denoted as Z_{noise}, are deemed to be noisy. We then remove Z_{noise} from Z_{train} to create the new training set Z_{clean}. On Z_{clean}, the model is retrained from scratch using the same hyperparameters.

3 EVALUATION

We introduce the datasets and source code models used in our experiments.

Datasets: Two types of dataset are involved in our evaluation:

(1) **Synthetic Noisy Dataset**: We inject random noise to a clean dataset. We chose the POJ-104 [12], a commonly used dataset for code classification. This dataset contains 52,000 C programs divided into 104 classes of 500 programs each. We randomly select 10% samples in each class and randomly relabel these samples. Now we have a training set Z_{train} containing both clean and noisy data.
(2) **Real Noisy Dataset**: As shown in [6], the dataset used in defect prediction task might contain a significant amount of noise. We chose Devign [20] as the representative after confirming that there are noises in the dataset. Devign dataset includes 21,854 potententially vulnerable C functions collected from open source projects. Each function is manually labeled by software security experts as vulnerable or not.

Source code Models: We take the public code artifacts from ASTNN[2], CodeBERT[3] to reproduce results reported in the original works.

[2]https://github.com/zhangj111/astnn
[3]https://github.com/microsoft/CodeBERT

Table 2: Results after retraining on the Devign dataset.

	Test ACC	Method	Test ACC after removing
CodeBERT	62.91 ± 0.08	IF	**63.31 ± 0.10**
		TracIn	**63.40 ± 0.20**
		Random	61.73 ± 0.05

3.1 Evaluation Results

Firstly, we evaluate our method on the synthetic noisy dataset. Table 1 shows the performance of IF and TracIn in identifying the noisy examples on Z_{train}. In practice, we do not know how many percent of the samples in the dataset are noisy, so we choose different values of k. With ASTNN, in top $k = 10\%$ samples with the lowest score, both IF and TracIn can detect more than 55% noisy samples, and when $k = 1\%$, more than 91% samples in this subset are noise. Next, we evaluate data-influence methods on real noisy dataset. Column *Test ACC* in table 2 shows the performance of models trained on the original dataset. We then calculate the score for all training instances with the same procedure in our evaluation pipeline. The top 1% (best hyperparameter) samples with the lowest score are selected. We also include Random - a baseline randomly selecting a 1% sample set as Z_{noise}. The results in table 2 show that by using data-influence methods, there are improvements in terms of ACC after retraining. In general, the results in Table 1 and Table 2 support our hypothesis that data-influence methods are effective at detecting noise in code corpus; and removing noises and re-training with cleaner datasets improves the performance of code models

4 DISCUSSION & CONCLUSION

We present a novel data-centric perspective for enhancing the quality of source code models by using data-influence methods. We performed various analyses on several baselines and obtained potentially promising results for improving the quality of source code data. There are numerous aspects that we can investigate in the future. We mostly rely on synthetic noisy datasets to perform the evaluation. Also, we only concentrate on classification-based tasks while we can do the same for many other tasks.For example, when performing a generation-based task like code summarization, the comments and method body are extracted from code snippets collected on Github. However, not all of the developers' comments reflect the functionality of the given code snippet; this can also be interpreted as noise and should be carefully examined too. In the future, we intend to pursue our research in three directions: (1) Identifying more noisy datasets to analyze and providing insights on the noises of such datasets; (2) Improving the methods to detect noisy data; and (3) Applying the methods to a broader range of software engineering tasks, such as code summarization, bug detection, and code translation.

5 ACKNOWLEDGEMENTS

This work is partly funded by FPT Software AI Center. We also thank the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to Represent Programs with Graphs, In International Conference on Learning Representations (ICLR). *CoRR*. https://doi.org/arXiv:1711.00740

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*. PMLR, 2091–2100.

[3] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.

[4] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.

[5] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[6] Shihab Shahriar Khan, Nishat Tasnim Niloy, Md Aquib Azmain, and Ahmedul Kabir. 2020. Impact of Label Noise and Efficacy of Noise Filters in Software Defect Prediction.. In *SEKE*. 347–352.

[7] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[8] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks, In International Conference on Learning Representations (ICLR). *arXiv:1511.05493 [cs, stat]*. arXiv: 1511.05493.

[9] Chao Liu, Xin Xia, David Lo, Cuiyun Gao, Xiaohu Yang, and John Grundy. 2021. Opportunities and challenges in code search tools. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–40.

[10] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2021. Deep Learning for Android Malware Defenses: a Systematic Literature Review. *arXiv preprint arXiv:2103.05292* (2021).

[11] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 1287–1293.

[12] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Thirtieth AAAI conference on artificial intelligence*.

[13] Andrew Ng. 2022. Andrew Ng "the data-centric AI approach". https://www.youtube.com/watch?v=TU6u_T-s68Y

[14] Pouya Pezeshkpour, Sarthak Jain, Byron Wallace, and Sameer Singh. 2021. An Empirical Comparison of Instance Attribution Methods for NLP. 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, 967–975. https://doi.org/10.18653/v1/2021.naacl-main.75

[15] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems* 33 (2020), 19920–19930.

[16] Arumoy Shome, Luis Cruz, and Arie van Deursen. 2022. Data Smells in Public Datasets. *arXiv preprint arXiv:2203.08007* (2022).

[17] Zhensu Sun, Li Li, Yan Liu, Xiaoning Du, and Li Li. 2022. On the importance of building high-quality training datasets for neural code search. In *Proceedings of the 44th International Conference on Software Engineering*. 1609–1620.

[18] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2020. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *arXiv preprint arXiv:2009.06520* (2020).

[19] Yanjie Zhao, Li Li, Haoyu Wang, Haipeng Cai, Tegawendé F Bissyandé, Jacques Klein, and John Grundy. 2021. On the impact of sample duplication in machine-learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–38.

[20] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 10197–10207. https://proceedings.neurips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html