Efficacy of Pretrained Architectures for Code Comment Usefulness Prediction

Sagar Joshi[1], Sumanth Balaji[1], Aditya Hari[1], Abhijeeth Singam[1] and Vasudeva Varma[1]

[1]International Institute of Information Technology, Hyderabad

Abstract

Source code is usually accompanied by comments which help in improving code comprehension. However, not all comments are helpful in this respect, with some comments being redundant, some being unclear which results in a poorer code readability. Sifting through large volumes of code to identify such comments manually being tedious, the task of automatically evaluating the usefulness of a comment in context of the code it lies in can add value. In this work, we evaluate the performance of various pretrained transformer encoders to solve this task. We demonstrate decent performance of a few models alongside abnormally high performance obtained by a few other pretrained architectures.

Keywords

Code comment, Usefulness prediction, Pretrained models, Classification

1. Introduction

Programmers rely on source comments to understand the functioning of code. It represents a way of documenting the code, with the comments explaining the logic behind the different parts of the code, the expected behavior, inputs, and outputs. Code comments not only help programmers working with the code, but they also aid in aspects related to software maintenance and reusability. However, not all comments are of high quality. Some examples include inconsistent comments, redundant or superfluous comments explaining the code rather than its rationale, and irrelevant comments. Such comments reduce the comprehensibility and readability of the code and are thus not desirable.

This motivates the need for a way to evaluate the quality of code comments given the piece of code they accompany. Different methods based on have been proposed in the literature for solving this problem, such as those based on heuristics and subjective and objective quality attributes [1]. Methods based on machine learning have also been proposed for solving this problem [2]. A previous work [3] deals with information extraction from code comments for obtaining a better representation for program analysis. A knowledge graph representing the software implementation and management information was constructed based on comments from C and C++ files. In another work, [4], categories of comments were studied based on their relation to concepts in software development with the aid of manual annotation,

Forum for Information Retrieval Evaluation, December 9-13, 2022, India

sagar.joshi@research.iiit.ac.in (S. Joshi); sumanth.balaji@research.iiit.ac.in (S. Balaji); aditya.hari@research.iiit.ac.in (A. Hari); abhijeeth.singam@students.iiit.ac.in (A. Singam); vv@iiit.ac.in (V. Varma)

following which comments were labeled as 'Useful', 'Partially Useful' or 'Useless'. The 3-way classification was annotated based on a rubric consisting of a set of rules to determine the characteristics of the comment with respect to the code context in which it lies. The best results were obtained using an LSTM architecture combined with handcrafted features from comment analysis. The shared task [5] deals with the same problem but on a binary scale. Given a comment and the code context in which it occurs, we classify it as 'Useful or 'Not Useful'.

Unlike previous work, which makes use of custom-trained embeddings and explicit features, we investigate the efficacy of pretrained language models in this particular task. Large pretrained models have achieved SoTA performance across many downstream tasks in different domains. Models such as CodeBERT [6] have also emerged recently, explicitly designed for software engineering tasks and capable of understanding both source code semantics and natural language semantics.

2. Method

Seven language models were chosen to experiment with for this task. These are - ALBERT [7], BERT [8], CodeBERT [6], DeBERTa [9], MPNet [10], RoBERTa [11], and XLNet [12]. Off-the-shelf pretrained versions of these models were then finetuned on the training set and optimized using the dev set. We also used a Transformer [13] encoder initialized with random weights for comparison against the pretrained models. Finally, the results were reported according to the performance on the test set. This method benefits from being agnostic to language, with the pretrained models capable of working with code in any language.

We have open-sourced our code for replication and further experimentation[1].

3. Experimentation

3.1. Data

The provided dataset for the shared task consisted of separate splits for training and evaluation (test) of sizes 8000 and 1001, respectively. However, duplicates were found in the samples provided, and some samples in the test data were present in the training set. Hence, we first removed all the duplicates from both sets and cleaned the test to remove all the samples already preset in the train set to not tamper with the integrity of the evaluation. We also separated 10% of the data from the deduplicated train set as the validation (dev) set for saving the best checkpoint during training.

	train	dev	test
# samples	5354	595	678

Table 1
Dataset splits following data duplication and test set cleaning

[1] https://github.com/sagarsj42/irse-fire-2022

Following data duplication and test cleaning, the sizes of the data splits obtained are indicated in Table 1.

3.2. Experimental Settings

For each of the experiments performed, base versions of the model were trained for 5 epochs on a batch size of 4 for training, and gradients were accumulated for two steps, thus providing an effective batch size of 8. The learning rate was increased till 40% of the training steps peaking at 6e-5, followed by a cosine decay. Validation was performed at the end of each training epoch using the dev set, and the best-performing model checkpoint was saved based on the F1 score. The best checkpoint thus determined was used to evaluate the test data.

4. Results

Model	Accuracy	F1	MCC	ROC-AUC
Transformer	67.55	51.33	29.97	0.78
BERT	69.91	44.86	37.74	0.63
ALBERT	70.35	73.09	53.86	0.94
XLNet	**88.05**	**85.66**	**75.57**	**0.93**
MPNet	87.02	83.88	73.02	0.93

Table 2
Results of different pretrained transformer architectures on the cleaned & deduplicated test set of the task that performed well on the data

The results of the code comment usefulness classification on the deduplicated and cleaned test data are in Table 2. In this table, we have only included models that gave a decent performance on the dataset. Other models that gave an anamolously high performance are included separately in Section 4.1. Evaluation was done using Accuracy, F1, Matthew's Correlation Coefficient (MCC) and ROC-AUC metrics [2].

The BERT model was performing only slightly ahead of the transformer encoder trained from scratch. There were differences in the training process, however, with the BERT model achieving the optimal performance within 3 epochs, but the transformer encoder showed a gradual increase in performance, thus distinguishing the effect of pretraining. XLNet and MPNet - the two permutation language modeling-based architectures - performed gave a decent performance with the former having an edge in performance over the latter. ALBERT performed close to BERT in accuracy but showed a significant gap w.r.t. the other three metrics.

Model	Accuracy	F1	MCC	ROC-AUC
RoBERTa	99.56	99.45	99.08	1.00
CodeBERT	99.56	99.45	99.08	1.00
DeBERTa	99.41	99.26	98.77	1.00

Table 3
Results of transformer architecture models that gave an anomalously high performance on the data

4.1. Anomalously High Performance

The transformer architectures that gave an unusually high performance are shown in Table 3. As can be seen, the models achieved an almost-perfect performance. Although CodeBERT and RoBERTa tied up in the performance on the test set, CodeBERT achieved optimal performance by the 3rd epoch in training while RoBERTa at the 4th epoch, CodeBERT being a model pretrained on the RoBERTa architecture on programming language data, thus highlighting the faster convergence achieved due to domain-specific pretraining. Both the models maintained accuracy of 100% on the dev set in the subsequent epochs. The closest performance to these models was achieved by DeBERTa. An ROC-AUC score of 1 indicates an almost perfect confidence achieved in the classification task, which indicates that the models have been easily able to fit almost perfectly on the distribution of the data, and the test and train data distributions do not have any significant variation in the distribution. Such an anomalous behaviour might have resulted from the annotations for the train and test set being done on the basis of a rule-set, and the models being able to learn those rules.

5. Conclusion & Future Scope

In this work, we evaluated the performance of different pretrained transformer architectures on the downstream task of code-comment usefulness prediction. While some of the models gave a decent performance, some of them performed abnormally well. The latter aspect needs to be investigated further by experimenting with different sets of data prepared over different programming languages, and diverse set of annotations. A further investigation of the mechanics behind some architectures performing exceedingly well on the dataset - which does not have a direct overlap between its train and evaluation splits - can add value to the effectiveness of a transformer-based solution to the problem.

References

[1] B. Yang, Z. Liping, Z. Fengrong, A survey on research of code comment, in: Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences, ICMSS 2019, Association for Computing Machinery, New York, NY,

[2]The results submitted to the shared task were based on the actual test set, and not the cleaned one - the values for accuracy and F1, in that case, is for these two models being 99.7% and 99.46% respectively. The other model results also slightly differ from the ones in the table due to this reason.

USA, 2019, p. 45–51. URL: https://doi.org/10.1145/3312662.3312710. doi:10.1145/3312662.3312710.

[2] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, in: 2013 21st International Conference on Program Comprehension (ICPC), 2013, pp. 83–92. doi:10.1109/ICPC.2013.6613836.

[3] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Comment-Mine—A Semantic Search Approach to Program Comprehension from Code Comments, Springer Singapore, Singapore, 2020, pp. 29–42. URL: https://doi.org/10.1007/978-981-15-2930-6_3. doi:10.1007/978-981-15-2930-6_3.

[4] S. Majumdar, A. Bansal, P. P. Das, P. D. Clough, K. Datta, S. K. Ghosh, Automated evaluation of comments to aid software maintenance, Journal of Software: Evolution and Process 34 (2022) e2463. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2463. doi:https://doi.org/10.1002/smr.2463. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2463.

[5] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. D Clough, S. Chattopadhyay, P. Majumder, Overview of the IRSE track at FIRE 2022: Information Retrieval in Software Engineering, in: Forum for Information Retrieval Evaluation, ACM, 2022.

[6] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, Codebert: A pre-trained model for programming and natural languages, 2020. URL: http://arxiv.org/abs/2002.08155, cite arxiv:2002.08155Comment: Accepted to Findings of EMNLP 2020. 12 pages.

[7] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, Albert: A lite bert for self-supervised learning of language representations., in: ICLR, OpenReview.net, 2020. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#LanCGGSS20.

[8] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, CoRR abs/1810.04805 (2018). URL: http://arxiv.org/abs/1810.04805. arXiv:1810.04805.

[9] P. He, X. Liu, J. Gao, W. Chen, Deberta: Decoding-enhanced bert with disentangled attention, CoRR abs/2006.03654 (2020). URL: http://dblp.uni-trier.de/db/journals/corr/corr2006.html#abs-2006-03654.

[10] K. Song, X. Tan, T. Qin, J. Lu, T.-Y. Liu, Mpnet: Masked and permuted pre-training for language understanding, 2020. arXiv:2004.09297.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, 2019. arXiv:1907.11692.

[12] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, Q. V. Le, Xlnet: Generalized autoregressive pretraining for language understanding, 2019. URL: http://arxiv.org/abs/1906.08237, cite arxiv:1906.08237Comment: Pretrained models and code are available at https://github.com/zihangdai/xlnet.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.