Exploring Usefulness of C Comments with Silver-Standard Machine Learning Models

Aritra Mitra[1,*]

[1]*Indian Institute of Technology, Kharagpur (IIT-KGP), West Bengal-721302, India*

Abstract

Comments are very useful to the flow of code development. With the increasing use of code in commonplace life, commenting the codes becomes a hassle for rookie coders, and often they do not even think commenting as a part of the development process. This in general causes the quality of comments to degrade, and a considerable amount of useless comments are found in such codes. In these experiments, the usefulness of C comments are evaluated using LLM-generated silver standard machine learning models. The results of the experiments create a baseline for better results that can be found in the future through more research. Based on these findings, more complex and accurate machine learning models can be created that can improve the accuracy achieved in performing said task.

Keywords

Machine Learning, Natural Language Processing, Model Generation, Large Language Model,

1. Introduction

Comments add useful information to codebases, making maintenance of such codebases easier. But on the other hand, a comment can add bloat to the storage front if it is not succinct, or useless. As programming becomes increasingly prevalent, inexperienced coders often neglect the practice of commenting, resulting in a decline in both the quality and quantity of comments [1]. Numerous comments prove to be useless, and combing through useless comments while trying to maintain an old codebase is ineffective and possible futile.

Numerous deep learning-based automatic commenting algorithms can cull the number of comments [2]. Unfortunately, there has been inadequate research focused on the problem of comment quality. Recent initiatives are tackling these difficulties by using machine learning models that can discover and classify comments according to their use.

The author has examined different machine learning (ML) models to address this issue. This paper seeks to address essential inquiries within the framework of the Information Retrieval in Software Engineering (IRSE) shared task at the Forum for Information Retrieval Evaluation (FIRE) 2024.

- What degree of complexity is required for a Machine Learning model to consistently differentiate between valuable and worthless comments?
- As large language models become more and more capable of solving tasks, can these models be used for our task?

This research intends to illustrate that models generated by LLMs might function as viable initial approaches for addressing this issue. More intricate models may be developed upon these foundations, while also accounting for the risk of overfitting.

This work additionally examines the critical issue of whether the dataset for the learning job may be enhanced with data produced by large language models, such as GPT-3. This factor is crucial in examining the prospect of utilizing artificial intelligence to enhance comment quality evaluation. This study evaluates the effects of incorporating AI-generated data into the dataset, aiming to elucidate

Forum for Information Retrieval Evaluation, December 12-15, 2024, India

*Corresponding author.
aritramitra2002@gmail.com (A. Mitra)
https://cse.iitkgp.ac.in/~aritra.mitra/ (A. Mitra)

the advantages and problems of integrating advanced language models into the machine learning pipeline. This inquiry constitutes a crucial element of the research, highlighting the convergence of human-authored and AI-generated content in the realm of comment quality assessment.

2. Related Work

Software metadata [3] is essential for code maintenance and comprehension. A variety of tools have been created to facilitate the extraction of knowledge from software metadata, encompassing runtime traces and structural characteristics of code [4, 5, 6, 7, 8, 9, 10, 11, 12]. Numerous researchers have investigated the quality of code comments in the context of mining. Steidl et al. [13] utilize methods like Levenshtein distance and comment length to assess the similarity of words in code-comment pairs, thereby successfully eliminating trivial and non-informative comments. Rahman et al. [14] concentrate on differentiating valuable from inconsequential code review comments in review portals, utilizing insights derived from attributes recognized in a survey done with Microsoft developers [15]. Majumdar et al. [16, 17, 18, 19] have proposed a framework for assessing comments based on concepts essential for code understanding. Their methodology entails the creation of textual and code correlation attributes, employing a knowledge graph to semantically analyze the information contained in comments. These methodologies utilize both semantic and structural characteristics to tackle the predictive challenge of differentiating valuable comments from unhelpful ones, hence aiding in the cleaning of codebases. Given the advent of huge language models, such as GPT-3.5 or LLaMA, it is essential to evaluate the quality of code comments and juxtapose them with human interpretation. The IRSE track at FIRE 2024 builds upon the methodology introduced in a previous study [16]. This study investigates diverse vector space models and features for binary categorization and assessment of comments, particularly for their significance in understanding code. This study performs a comparative investigation of the prediction model's performance when incorporating GPT-generated labels for code and comment quality derived from open-source software.

3. Description of Task and Dataset

This section presents a description of the task and the associated dataset. The assignment at IRSE, FIRE 2024 was as follows:
A binary code comment quality classification model requires enhancement through the incorporation of generated code and comment pairs to elevate its accuracy.
The associated dataset was divided into two parts:

The training dataset comprises 8,048 items.

The testing dataset comprises 1,000 entries.

The training dataset was randomized and divided into 70% for model training and 30% for cross-validation. The data was categorized as follows:

- **Useful**: Comments that enhance understanding of the code
- **Not Useful**: Comments that do not contribute to code understanding

4. Augmentation

The dataset augentation was performed using GPT-4o-mini and GPT-3.5-Turbo. The augmented dataset was then fed to CodeT5, GPT-4o-mini, GPT-3.5-Turbo and Code-LLaMA to create labels for the data. three different types of prompting techniques were used for this tasks, the description of which are as follows:

Table 1
Description of the Dataset for the Task

Label	Example
Useful	/*not interested in the downloaded bytes, return the size*/
Useful	/*Fill in the file upload part*/
Not Useful	/*The following works both in 1.5.4 and earlier versions:*/
Not Useful	/*lock_time*/

I/O Prompting: In this case, we ask the LLM to perform the labelling as well as the data generation using a roleplay-based method, where the LLM provides us with only the label or the generated data without any additional burden in the response.

Chain of Thoughts (CoT) Prompting: [20] In this situation, the LLM is asked to generate the sought output, as well as to provide the thought process through which the answer was reached. This makes the LLM provide more sensible data, but requires processing of the response.

Tree of Thoughts (ToT) Prompting: [21, 22] Here, the LLM is allowed to output multiple different responses for a task, and then based on those responses, further questions are asked. This creates a tree of thoughts, and the best response can be selected from the leaves. This results in a further increase in the accuracy, but requires high amounts of processing afterwards.

5. System Description

5.1. Text Preprocessing

All links, punctuation, numerals, and stop words had been eliminated. Subsequently, all words possessing a POS tag other than a Noun, Verb, Adverb, or Adjective were eliminated. Lemmatization was employed to consolidate several versions of a word into a singular term. NLTK WordNet [23] was utilized for lemmatization. The training and testing datasets employ identical preparation procedures.

5.2. Feature Extraction

The TfidfVectorizer [24] had been employed to transform text into numerical characteristics. The Keras library's Tokenizer was utilized in conjunction with the TfidfVectorizer from the SciKit-Learn package.

5.3. Machine Learning Models

Three models were used for this task, the description of which are as follows:

SilverCodeBERT: This model uses a **CodeBERT-base** model to create embeddings for the Natural Language (NL) comment as well as the Programming Language (PL) code context. The inference is based on the resultant embedding, using a Multi-Layer Perceptron (MLP).

SilverDoubleBERT: This model uses a **BERT-base-uncased** model to create embeddings for the Natural Language (NL) comment, and a **CodeBERT-base** model for the Programming Language (PL) code context. The inference is based on the resultant embedding, using a Multi-Layer Perceptron (MLP).

SilverLSTM: This model uses a **GRU** model to create embeddings for the Natural Language (NL) comment as well as for the Programming Language (PL) code context. The inference is based on the resultant embedding, using a Support Vector Machine (SVM) classifier.

NOTE that these models, however generated by LLMs, had to be slightly modified to be syntactically correct. These were then finetuned on the original as well as the augmented dataset.

6. Findings

6.1. Without Augmentation

Table 2
Results of Classifier Runs

Run	Macro F1 Score	Macro Precision	Macro Recall	Accuracy%
SilverCodeBERT	0.802	0.813	0.772	78.05
SilverDoubleBERT	0.813	0.819	0.788	79.11
SilverLSTM	0.794	0.790	0.767	77.25

6.2. With Augmentation

Table 3
Result of Classifier Runs

Run	Macro F1 Score	Macro Precision	Macro Recall	Accuracy%
SilverCodeBERT	0.812	0.821	0.794	79.35
SilverDoubleBERT	0.825	0.829	0.804	81.31
SilverLSTM	0.800	0.781	0.803	78.52

7. Conclusion

The tasks were executed using machine learning models. The findings from the SilverDoubleBERT classifier suggest potential for improvement, facilitating the creation of more complex models that correspond more effectively with the problem description and produce better outcomes. Srijoni Majumdar et al. [25] have attained exceptional outcomes utilizing ELMo and BERT-based models, and the author expects further enhancement of these results in the future.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

Acknowledgments

Thanks to the creators of IRSE FIRE for giving this wonderful opportunity to work on such a project, and their constant technical support throughout the timespan.

References

[1] J. Raskin, Comments are more important than code, ACM Queue 3 (2005) 64–. doi:10.1145/1053331.1053354.

[2] E. Wong, J. Yang, L. Tan, Autocomment: Mining question and answer sites for automatic comment generation, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, pp. 562–567. doi:10.1109/ASE.2013.6693113.

[3] S. C. B. de Souza, N. Anquetil, K. M. de Oliveira, A study of the documentation essential to software maintenance, 2005.

[4] L. Tan, D. Yuan, Y. Zhou, Hotcomments: how to make program comments more useful?, in: Conference on Programming language design and implementation (SIGPLAN), ACM, 2007, pp. 20–27.

[5] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Smartkt: a search framework to assist program comprehension using smart knowledge transfer, in: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2019, pp. 97–108.

[6] N. Chatterjee, S. Majumdar, S. R. Sahoo, P. P. Das, Debugging multi-threaded applications using pin-augmented gdb (pgdb), in: International conference on software engineering research and practice (SERP). Springer, 2015, pp. 109–115.

[7] S. Majumdar, N. Chatterjee, S. R. Sahoo, P. P. Das, D-cube: tool for dynamic design discovery from multi-threaded applications using pin, in: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2016, pp. 25–32.

[8] S. Majumdar, N. Chatterjee, P. P. Das, A. Chakrabarti, A mathematical framework for design discovery from multi-threaded applications using neural sequence solvers, Innovations in Systems and Software Engineering 17 (2021) 289–307.

[9] S. Majumdar, N. Chatterjee, P. Pratim Das, A. Chakrabarti, Dcube_ NN D cube NN: Tool for Dynamic Design Discovery from Multi-threaded Applications Using Neural Sequence Models, Advanced Computing and Systems for Security: Volume 14 (2021) 75–92.

[10] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, A. Brechmann, Measuring neural efficiency of program comprehension, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 140–150.

[11] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, S. C. Hoi, Codet5+: Open code large language models for code understanding and generation, arXiv preprint arXiv:2305.07922 (2023).

[12] J. L. Freitas, D. da Cruz, P. R. Henriques, A comment analysis approach for program comprehension, 2012.

[13] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, 2013.

[14] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, 2017.

[15] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, 2015.

[16] 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.

[17] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Comment-mine—a semantic search approach to program comprehension from code comments, in: Advanced Computing and Systems for Security, Springer, 2020, pp. 29–42.

[18] S. Majumdar, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Overview of the irse track at fire 2022: Information retrieval in software engineering, 2022.

[19] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. Clough, S. Chattopadhyay, P. Majumder, Can we predict useful comments in source codes?-analysis of findings from information retrieval in software engineering track@ fire 2022, in: Proceedings of the 14th Annual Meeting of the Forum for Information Retrieval Evaluation, 2022, pp. 15–17.

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, 2023. URL: https://arxiv.org/abs/2201.11903. arXiv:2201.11903.

[21] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023. URL: https://arxiv.org/abs/2305.10601. arXiv:2305.10601.

[22] J. Long, Large Language Model Guided Tree-of-Thought, 2023. URL: https://arxiv.org/abs/2305.08291. arXiv:2305.08291.

[23] E. Loper, S. Bird, Nltk: The natural language toolkit, 2002. URL: https://arxiv.org/abs/cs/0205028. doi:10.48550/ARXIV.CS/0205028.

[24] V. Kumar, B. Subba, A tfidfvectorizer and svm based sentiment analysis framework for text data corpus, in: 2020 National Conference on Communications (NCC), 2020, pp. 1–6. doi:10.1109/NCC48643.2020.9056085.

[25] 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.