Categorizing Code Comments based on its Relevance for Code Readability

Om Joshi[1,*]

[1] *Indian Institute of Technology, Goa, India - 403401*

Abstract

In software engineering, the value of code comments is often variable, underscoring the need for systematic approaches to assess their relevance effectively. This study explores the enhancement of code comment usefulness classification by combining a manually labeled dataset with synthetic data generated through advanced augmentation methods. Leveraging GPT-3.5-turbo, a powerful language model, we generated additional labeled comment samples to expand the training dataset. A baseline classification model was implemented using Logistic Regression and Random Forest techniques. Notably, the performance metrics, including an F1 score of approximately 0.79, remained steady across experiments with and without synthetic data integration. These findings illuminate the strengths and limitations of using synthetic data augmentation in refining the accuracy of code comment usefulness evaluation.

Keywords

Large Language Models, GPT-3.5, Random Forests, Data Augmentation, Comment Classification, Qualitative Analysis

1. Introduction

In the realm of software engineering, code serves as the backbone for many sectors, spanning finance, healthcare, and infrastructure. As software systems evolve to meet new demands, the complexity of their codebases grows, necessitating effective maintenance strategies to ensure functionality and quality over time. Often, rapid development timelines require swift code modifications and bug fixes, which can lead to inconsistent or outdated documentation. As a result, code comments become one of the most reliable sources of information for developers and automated tools, encapsulating the intent and logic behind code segments.

However, the quality and clarity of comments vary widely, creating a need for automated methods to assess their usefulness accurately. Addressing this challenge, our study introduces an approach that augments a manually labeled dataset of C-language code comments with synthetic examples generated by GPT-3.5-turbo, a state-of-the-art language model. This augmented dataset enables us to explore the impact of synthetic data on the accuracy of comment usefulness classification. Using a baseline Random Forest model for binary classification, we observed stable F1 scores of approximately 0.80 across both the original and enhanced datasets, indicating that synthetic data generation can complement manually annotated data without significantly altering model performance.

This work contributes to the field by evaluating the interaction between manual annotations and language model-generated data, offering insights into the practicalities of synthetic data augmentation for improving comment classification in ever-evolving software environments. The structure of this paper proceeds as follows: Section 2 reviews related work; Section 3 introduces the task and dataset; Section 4 details the methodology; results are discussed in Section 5; and Section 6 provides concluding insights.

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

[*] Corresponding author.

joshi.om.23033@iitgoa.ac.in (O. Joshi)

0009-0009-3290-7292 (O. Joshi)

2. Related Work

Automated program understanding is a recognized research area among professionals in the software domain. Various tools have been developed to facilitate the extraction of knowledge from software metadata, encompassing components such as runtime traces and structural attributes of code [1, 2, 3, 4, 5, 6, 7, 8]. Researchers have developed various methods to mine and evaluate code comments, focusing on analyzing comment quality through code-comment pair comparisons. In assessing code comment quality, authors [9, 10, 11, 12, 13, 14, 15, 16] employ techniques such as word similarity measures (e.g., Levenshtein distance) and comment length analysis to filter out trivial and non-informative comments. Rahman et al. [17] detect useful and non-useful code review comments (logged-in review portals) based on attributes identified from a survey conducted with developers of Microsoft [18].

New programmers often rely on existing comments to comprehend code flow. However, not all comments contribute effectively to program comprehension, necessitating a relevancy assessment of source code comments prior to their use. Numerous researchers have focused on the automatic classification of source code comments in terms of quality evaluation. For instance, Omal et al. [19] noted that factors influencing software maintainability can be organized into hierarchical structures. The authors defined measurable attributes in the form of metrics for each factor, enabling the assessment of software characteristics, which can then be consolidated into a single index of software maintainability. Fluri et al.[20] examined whether the source code and associated comments are changed together along the multiple versions. They investigated three open source systems, such as *ArgoUML, Azureus*, and *JDT Core*, and found that 97% of the comment changes are done in the same revision as the associated source code changes. Yu Hai et al.[21] classified source code comments into four classes - unqualified, qualified, good, and excellent. The aggregation of basic classification algorithms further improved the classification result. Another work published in [22] in which author proposed an automatic classification mechanism "CommentProbe" for quality evaluation of code comments of C codebases. We see that people worked on source code comments with different aspects[22, 23, 13, 12, 15, 16], but still, automatic quality evaluation of source code comments is an important area and demands more research.

With the advent of large language models [24], it is important to compare the quality assessment of code comments by the standard models like GPT 3.5 or llama with the human interpretation. The IRSE track at FIRE 2024 [25, 26] builds upon the methodologies proposed in [22, 27, 28, 12] to investigate various vector space models [29] and features for binary classification and evaluation of comments in relation to code comprehension. This track also assesses the performance of the predictive model by incorporating GPT-generated labels for the quality of code and comment snippets extracted from open-source software.

3. Task and Dataset Description

This research focuses on developing a binary classification model to label source code comments as either *useful* or *not useful.* The classification system takes a code comment along with its associated lines of code as input and outputs a label indicating the comment's relevance to the corresponding code, which assists developers in code understanding. Traditional machine learning algorithms, such as logistic regression, can be applied to construct this classification model. The two label categories for source code comments are defined as follows:

- *Useful* - The comment provides meaningful information related to the source code.
- *Not Useful* - The comment is deemed irrelevant to the associated source code.

The dataset used for this study comprises over 11,000 code-comment pairs extracted from open-source projects, primarily in C language. Each pair contains a code snippet and its corresponding comment, along with a label identifying its usefulness or lack thereof. This dataset was meticulously curated from GitHub and annotated by a team of 14 professionals. To further enrich the dataset, an

#	Comment	Code	Label
1	/*test 529*/	-10. int res = 0; -9. CURL *curl = NULL; -8. FILE *hd_src = NULL; -7. int hd; -6. struct_stat file_info; -5. CURLM *m = NULL; -4. int running; -3. start_test_timing(); -2. if(!libtest_arg2) { -1. #ifdef LIB529 /*test 529*/ 1. fprin	Not Useful

2	/*cr to cr,nul*/	-1. else
		/*cr to cr,nul*/
		1. newline = 0;
		2. }
		3. else {
		4. if(test->rcount) {
		5. c = test->rptr[0];
		6. test->rptr++;
		7. test->rcount–;
		8. }
		9. else
		10. break;
			Not Useful

3	/*convert minor status code (underlying routine error) to text*/	-10. break;
		-9. }
		-8. gss_release_buffer(&min_stat, &status_string);
		-7. }
		-6. if(sizeof(buf) > len + 3) {
		-5. strcpy(buf + len, ".\n");
		-4. len += 2;
		-3. }
		-2. msg_ctx = 0;
		-1. while(!msg_ctx) {
		/*con
			Useful

Table 1
Example of a data instance

additional synthetic dataset was generated by using the GPT-3.5-turbo language model to create new code-comment pairs. The synthetic comments were manually validated to ensure accuracy, contributing over 200 additional labeled samples that mirror the structure of the original dataset.

4. Methodology

Our approach to classifying code comments into *useful* and *not useful* involves several structured steps. Initially, during **Dataset Preparation**, we compile a dataset with over 11,000 labeled code-comment pairs, further enhancing it with synthetic comments generated by GPT-3.5-turbo to increase data variety. In **Feature Engineering**, we identify key features such as comment length, specific keyword presence, and semantic analysis to capture the relevance of comments to their associated code. For **Model Training**, logistic regression is used due to its efficiency in binary classification tasks, and we train the model on both the original and augmented datasets. To measure the model's performance, we employ **Evaluation** metrics, including accuracy, precision, recall, and F1 score, assessing the effectiveness of our classification model.

The logistic regression model works by applying a logistic function to constrain the output between 0 and 1. This process starts with the formula $Z = Ax+B$ (Equation **??**) for calculating a linear combination of input features, followed by the application of the logistic function $logistic(Z) = \frac{1}{1+\exp(-Z)}$ (Equation **??**) to produce a probability score. A threshold of 0.6 is set to favor predictions toward the *useful* comment category. Each training example is represented with a three-dimensional feature vector, and the Cross-Entropy loss function is used to optimize hyperparameters. In training, 80% of the dataset is utilized, with the remaining 20% reserved for testing.

5. Results

Our Random Forest model was trained separately on the original dataset and on an augmented version that included additional data generated by GPT. The original dataset comprised 11,452 labeled samples, with an additional 233 samples sourced from GPT augmentation. In the initial experiment, only the original dataset was used, and the resulting performance metrics are shown below.

Upon augmenting the original dataset with the GPT-generated samples, the following outcomes were observed:

	Accuracy	Precision	Recall	F1 Score
Original Dataset	81.05679%	0.7913	0.8035	0.7967
Augmented Dataset	81.53476%	0.7945	0.8078	0.7913

Table 2
Performance metrics for binary classification using both datasets

The small changes in metrics across both datasets suggest that the GPT-generated samples effectively resemble the original data in quality, supporting the efficacy of synthetic data augmentation in this context.

6. Conclusion

This paper introduces a binary classification model to assess the usefulness of code comments, using a Random Forest model as the core algorithm. Our findings highlight that GPT-3.5-turbo-generated synthetic data closely approximates the quality of manually labeled data. This underscores the potential of synthetic data augmentation for broadening training datasets, especially when resources are limited.

References

[1] S. C. B. de Souza, N. Anquetil, K. M. de Oliveira, A study of the documentation essential to software maintenance, Conference on Design of communication, ACM, 2005, pp. 68–75.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access (2023).

[9] 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.

[10] 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).

[11] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[12] 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.

[13] 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., in: FIRE (Working Notes), 2022, pp. 1–9.

[14] J. L. Freitas, D. da Cruz, P. R. Henriques, A comment analysis approach for program comprehension, Annual Software Engineering Workshop (SEW), IEEE, 2012, pp. 11–20.

[15] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).

[16] P. Chakraborty, S. Dutta, D. K. Sanyal, S. Majumdar, P. P. Das, Bringing order to chaos: Conceptualizing a personal research knowledge graph for scientists., IEEE Data Eng. Bull. 46 (2023) 43–56.

[17] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 215–226.

[18] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, Working Conference on Mining Software Repositories, IEEE, 2015, pp. 146–156.

[19] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.

[20] B. Fluri, M. Wursch, H. C. Gall, Do code and comments co-evolve? on the relation between source code and comment changes, in: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE, 2007, pp. 70–79.

[21] H. Yu, B. Li, P. Wang, D. Jia, Y. Wang, Source code comments quality assessment method based on aggregation of classification algorithms, Journal of Computer Applications 36 (2016) 3448.

[22] 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.

[23] 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.

[24] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[25] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D Clough, A. Bandyopadhyay, S. Chattopadhyay, Generative ai for code metadata quality

assessment, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024.

[26] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D Clough, A. Bandyopadhyay, S. Chattopadhyay, Overview of the irse track at fire 2024: Information retrieval in software engineering, in: FIRE (Working Notes), 2024.

[27] S. Paul, S. Majumdar, A. Bandyopadhyay, B. Dave, S. Chattopadhyay, P. Das, P. D. Clough, P. Majumder, Efficiency of large language models to scale up ground truth: Overview of the irse track at forum for information retrieval 2023, in: Proceedings of the 15th Annual Meeting of the Forum for Information Retrieval Evaluation, 2023, pp. 16–18.

[28] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Generative ai for software metadata: Overview of the information retrieval in software engineering track at fire 2023, arXiv preprint arXiv:2311.03374 (2023).

[29] S. Majumdar, A. Varshney, P. P. Das, P. D. Clough, S. Chattopadhyay, An effective low-dimensional software code representation using bert and elmo, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2022, pp. 763–774.