Categorizing Code Comments based on its Relevance for Code Readability

Kashvi Aggarwal[1,*]

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

Abstract

In software development, code comments can become uniquely valuable, necessitating structured approaches to objectively assess their value. This study focuses upon augmenting code comment usefulness classification with a labelled dataset that was created manually, as well as synthetically through a data augmentation method. In the study, labelled comment samples were developed to add to the training dataset using the GPT-3.5-turbo approach. A baseline predicting usefulness from code comments was subsequently constructed using Logistic Regression and Random Forest models. In terms of results, the F1 score performance metric reached 0.79 across all conditions, whether or not the model was created with synthetic data. This study adds evidence for the advantages of using synthetically created data augmentation on the accuracy of code comment usefulness selections.

Keywords

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

1. Introduction

In software engineering, code is critical for many domains such as finance, healthcare, and infrastructure. As software systems adapt to meet new challenges, the complexity of their codebases increases, and often leads to inconsistent or outdated documentation, making maintenance difficult. For this reason, code comments become one of the most reliable sources for sharing information for developers and automated tools. Comments include valuable metadata about the intent and logic of program segments.

However, comments can vary widely in quality and clarity and cannot be relied upon to accurately grade them for usefulness. To address this issue, we present a method to enhance a manually labeled dataset of C language code comments through the addition of synthetic examples generated from a state-of-the-art language model called GPT-3.5-turbo. Our method evaluates the resulting dataset by exploring how the use of synthetic data produced through a language model affects rates of comment usefulness classification. The code comments were classified using a Random Forest model as a classifier baseline. We retained stable F1 scores of approximately 0.80 using both the original dataset and newly augmented datasets, with the analysis suggesting promise for language model-based synthetic data augmentation to complement human annotator codes comments without substantially changing rates of usefulness classification.

This research adds to the field evaluating the interaction of manual annotations and language model generated data, the contribution to practical aspects of using synthetic data augmentations to increase comment classification in dynamic 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 17-20, 2025, India

*Corresponding author.

kashviaggarwal78@gmail.com (K. Aggarwal)
0009-0006-3010-0688 (K. Aggarwal)

2. Related Work

Recognized by software experts, automated program understanding represents an area of research in software engineering. There are various tools for extracting knowledge from software metadata, including runtime traces and structure [1, 2, 3, 4, 5, 6, 7, 8, 9]. 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 [10, 11, 12, 13, 14, 15, 9, 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].

Assessing the relevance of source code comments is a critical step for ensuring program comprehension, yet not all comments are helpful. This has led to extensive research in automatic comment quality evaluation. Foundational studies established metrics for software maintainability [19] and confirmed the tight coupling between code and comment evolution [20]. More recent work has focused on creating multi-level classification systems [21] and automated frameworks like "CommentProbe" for C code [22]. While these and other studies [22, 23, 14, 13, 9, 16, 24, 25, 8, 26, 27, 28, 29] have made significant progress, the automatic evaluation of comment quality continues to be an important research challenge.

With the advent of Large Language Models (LLMs) such as GPT-3.5 [30], a key question emerges: how does their assessment of comment quality compare to human judgment? The IRSE track at FIRE 2024 [31, 25] tackles this question directly. It extends prior work [22, 26, 32, 13] by applying various vector space models [33] to the task of comment classification. Furthermore, the track explicitly evaluates the performance of predictive models when the training data is augmented with labels generated by GPT, providing insights into the utility of synthetic data for this software engineering task.

3. Task and Dataset Description

This study introduces a binary classification system to evaluate the utility of source code comments by labeling them as either *useful* or *not useful.* The model analyzes a comment and its corresponding code snippet to determine its relevance. Our methodology relies on a robust dataset of over 11,000 C code-comment pairs, which were professionally annotated by a team of 14 experts. To further enhance this dataset, we generated and manually validated an additional 200+ synthetic samples using GPT-3.5-turbo. This combined dataset serves as the foundation for training machine learning models, such as logistic regression, to perform the classification task.

4. Methodology

Our approach to classifying code comments involves using **embeddings** derived from both the comment and the associated code snippet, which are then fed into a classification model.

4.1. Methodology Block Diagram

Figure 1 provides a visual overview of the classification pipeline, from data preparation and augmentation to final model evaluation.

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

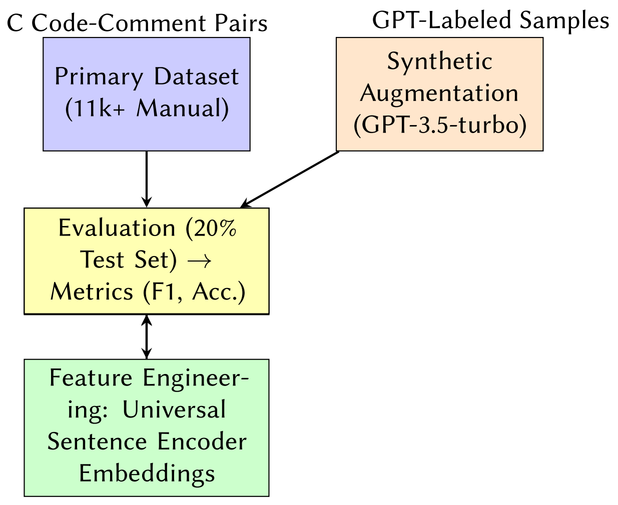
C Code-Comment Pairs
GPT-Labeled Samples
Primary Dataset (11k+ Manual)
Synthetic Augmentation (GPT-3.5-turbo)
Evaluation (20% Test Set) → Metrics (F1, Acc.)
Feature Engineering: Universal Sentence Encoder Embeddings

Figure 1: Block diagram illustrating the methodology pipeline for code comment usefulness classification.

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 $Z = Ax + B$ to calculate a linear combination of input characteristics, followed by the application of the logistic function $logistic(Z) = \frac{1}{1+\exp(-Z)}$ 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

We trained our Random Forest model independently with the original dataset and an augmented dataset with additional dataset produced by GPT. The original dataset consisted of a total of 11,452 labeled samples, and we later created an additional 233 sample from the GPT augmentation. In the first experiment, we used only the original dataset and the metrics follow below. The following metrics was captured after the original dataset was augmented by the GPT-generated samples:

	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 1
Performance metrics for binary classification using both datasets

The minor differences in measures across both datasets indicate that GPT-generated samples are successfully close to the original data in terms of quality, reinforcing the effectiveness of synthetic data augmentation in this setting.

6. Conclusion

In this paper, we present a binary classification model based on a Random Forest model as the machine learning algorithm to evaluate the utility of code comments. Our results indicate that synthetic data generated by GPT-3.5-turbo is very close to the quality of manually labeled data, which demonstrates the potential of augmenting training datasets with synthetic data when resources are constrained.

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.

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

[10] 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.

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

[13] 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.

[14] 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.

[15] 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.

[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] S. Majumdar, A. Deshpande, P. P. Das, P. P. Chakrabarti, Comprehending c codes with llms: Effective comment generation through retrieval and reasoning, Pattern Recognition Letters (2025).

[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, Overview of the irse track at fire 2024: Information retrieval in software engineering, in: FIRE (Working Notes), 2024.

[26] 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.

[27] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Tool assisted agile approach for legacy application migration, International Journal of System Assurance Engineering and Management (2025) 1–16.

[28] A. Deshpande, A. Maji, D. Mondol, P. P. Das, P. D. Clough, S. Majumdar, The code–llm handshake: Smarter maintenance through ai, in: Proceedings of the 17th annual meeting of the Forum for Information Retrieval Evaluation, 2025, pp. 9–12.

[29] A. Mitra, S. Majumdar, A. Mukhopadhyay, P. P. Das, P. D. Clough, P. P. Chakrabarti, Operationalizing large language models with design-aware contexts for code comment generation, arXiv preprint arXiv:2510.22338 (2025).

[30] 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.

[31] 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.

[32] 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).

[33] 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.