An Empirical Study on Synthetic Data Augmentation for Code Comment Quality Classification

Lakshay Khurana[1,*]

[1]Indian Institute of Technology, Goa, 403401

Abstract

Code comments are important for software maintenance tasks, but developing an automated assessment of the quality of code comments has proven more difficult than expected, often due to a shortage of large annotated datasets. In this paper, we will assess the utility of data augmentation using synthetically labeled comments to classify comments as useful or not useful. In particular, we will use GPT-3.5-turbo to label a second dataset that we will use to augment a manually labeled dataset. We develop and evaluate a baseline Support Vector Machine model with an F1 score of 0.80 on the original dataset and find that while we assumed we would improve on this performance by adding the synthetic data, the change in performance is negligible. The takeaway of the paper is to recognize the limitations of simple data augmentation with synthetically generated labels for our task, while acknowledging the model's ability to find signal in the data of the baseline model with sufficient data generated by human annotation.

Keywords

Large Language Models, GPT-3.5, Support Vector Machine, Comment Classification, Data Augmentation, Qualitative Analysis

1. Introduction

In today's increasingly digital world, software has become vital across a wide range of industries, including finance, healthcare, and transportation.The need for rapid innovation causes organizations to constantly update existing software and design new applications. Increased software updates lead to increasingly complex code, which makes managing substantial software systems a critical part of the software development life cycle (SDLC).

The time-consuming nature of these development processes can lead to poor code quality as developers rush to fix bugs quickly, and this takes precedence over the development of new features-and sometimes untested changes get released. When developers create software, they create documentation such as design specs, and this documentation that can be critical future reference to the specific domain predating developer onboarding may age or become no longer relevant to the software once it evolves, changing language evolves and various later stages such as developer involvement ceases, and they are no longer in the position to field questions relative to team members who inherit the project. This indicates that quality-oriented processes are critical in software to create understanding and domain knowledge regarding an projects existing code to be maintained.

Information sources that are more reliable than word of mouth include executing tests and assessing information derived using static code analysis methods and comments left by the developer(s) as even repository history may tell a story from one team member to another. This paper focuses on comments by developers relative to code comments, in which comments may save time and provide context for the code being analyzed. The usefulness of comment artifacts by developers that came before a member on a software development team is acknowledged. Importantly, the comments will serve as documentation for capturing the rationale and goals for the code that should aid in the understanding and maintenance of code. However, comments can also vary greatly in terms of quality, and therefore it is important to be able to utilize an automated method to evaluate the rating of comments.

Forum for Information Retrieval Evaluation, December 17-20, 2025, India

*Corresponding author.

lakshay.khurana.22033@iitgoa.ac.in (L. Khurana)

0009-0006-4391-9836 (L. Khurana)

One major issue in moving toward the development of such assessment tools is the lack of well-annotated datasets that describe the range of comments across different programming contexts. In this work, we evaluate the use of synthetic data augmentation to improve model performance. We start with a manually labeled dataset and enhance this using synthetic data labeled by GPT-3.5-turbo.

In this paper, we examine a binary classification problem for source code comments in the C programming language, classifying them as 'Useful' and 'Not Useful.' We first establish a baseline using a Support Vector Machine model trained on a dataset of over 11,000 manually labeled comments. We then supplement this dataset with over 200 additional samples labeled by the GPT-3 model and assess any improvement in overall performance. We found that the performance of the model remained unchanged, maintaining roughly an F1 score of 0.80 for both the original and synthetic datasets.

This research illustrates to the community a practical evaluation of using LLM-based synthetic data augmentation, in regards to code comment classification. We hope to address a few of the current challenges and guide the future creation of more robust and adaptable models in light of the growing software engineering workforce.

The paper is organized as follows. Section 2 reviews related work in comment classification. Section 3 details the task and dataset. Our methodology is presented in Section 4. The results are analyzed in Section 5, and Section 6 concludes the study.

2. Related Work

The research area of automated program comprehension is widely accepted in the field of software engineering. Various tools and techniques have been proposed to extract knowledge from software metadata, including runtime traces and structural code properties [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13].

Code comments are a primary resource for developers, yet their utility varies. Consequently, the automatic classification of source code comments has been a focus of extensive research. Omal et al. [14] proposed organizing factors of software maintainability into hierarchical structures with measurable attributes, allowing for a consolidated maintainability index. Fluri et al. [15] investigated the co-evolution of source code and comments, finding that 97% of comment changes occur in the same revision as the associated code changes in open-source systems like *ArgoUML*. Deissenboeck et al. [16] introduced a two-dimensional maintainability model linking system properties to maintenance activities, creating a structured quality knowledge base for industrial application. Other studies have explored the role of specific comment types, such as TODO annotations, in developers' task management [17].

The impact of comments on readability has also been empirically validated. An experiment by Tenny et al. [18] demonstrated that programs without comments were the least readable. More recent work has focused on fine-grained classification. Yu Hai et al. [19] classified comments into four quality tiers—unqualified, qualified, good, and excellent—and improved results by aggregating basic classification algorithms. Majumdar et al. [20] proposed "CommentProbe," an automatic classification mechanism for evaluating the quality of comments in C codebases. Despite these varied approaches [20, 21, 22, 23, 13, 24], the automatic quality evaluation of source code comments remains a challenging and active area of research.

With the advent of large language models (LLMs) [25], it has become pertinent to compare their assessment of code comment quality with human interpretation, like in the cases of [26, 27]. The IRSE track at FIRE 2024 [28, 10] extends methodologies from [20, 11, 29] to explore vector space models [30] for this task. This track specifically evaluates model performance when incorporating GPT-generated labels, aligning closely with the objectives of our study.

3. Task and Dataset Description

In this study, we tackle the problem of establishing a binary classification system for source code comments. Specifically, the objective is to assign a code comment, along with the relevant source

code context, a label indicating whether the comment is useful or not useful. We formulate the two classification categories as follows:

- *Useful*: The comment accurately describes or provides relevant, non-obvious information about the associated code.
- *Not Useful*: The comment is redundant, inaccurate, or fails to convey meaningful information about the code.

Our main dataset includes over 15,000 pairs of C code and comments. These pairs have been annotated by a group of 14 human annotators who have labeled each comment as either useful or not useful. To explore the possibilities of synthetic data, we created a secondary dataset of 233 code-comment pairs taken from GitHub. These pairs were automatically annotated using GPT-3.5-turbo. This secondary dataset is also in the same structure as the original dataset and will serve as an augmentation corpus for our experiments.

#	Description	Surrounding Code	Class
2	/*Example 1 for data augmentation*/	-12. void setup() { -11. int val = 0; -2. #endif // AUGMENT /*Example 1 for data augmentation*/ 1. print(val);	Unnecessary
4	/*Calculate the square root of the distance vector magnitude*/	-8. if (delta > 0) -3. double dist_sq = (x*x + y*y); -2. if (dist_sq > 0)	Informative

Table 1
Modified sample data instances with new formatting and content.

4. Methodology

Our methodology employs a Support Vector Machine model for binary classification. The model accepts a set of input features, created from both the code comment and the code snippet that surrounds it. To represent the comment and code context as vector embeddings, we employ a pre-trained Universal Sentence Encoder to compute one independently for the comment and the code context. The two vector embeddings are then combined to serve as the model's feature vector.

The joint dataset was separated into training and testing datasets, using 80% of the instances for training and 20% for testing, which was consistent for both experiments (one where we used data augmentation and one where we did not).

4.1. Support Vector Machine Model

The **Support Vector Machine (SVM)** is a binary classifier that finds an optimal **hyperplane** maximizing the **margin** between classes, defined by **support vectors**.

Model Equations

The linear output score Z, representing the signed distance from the hyperplane, is calculated as:

$$Z = \mathbf{w}^T \mathbf{x} + b \tag{1}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the feature vector, and b is the bias.

Classification

The classification is determined by the sign of Z:

$$\text{Class} = \begin{cases} +1 & \text{if } Z \geq 0 \\ -1 & \text{if } Z < 0 \end{cases} \tag{2}$$

The model is trained by minimizing the **Hinge Loss** and can use the **Kernel Trick** for non-linear data.

5. Results

We conducted two experiments to evaluate the impact of synthetic data augmentation. In the first experiment, the Support Vector Machine model was trained and tested solely on the original human-annotated dataset of 11,452 samples. In the second experiment, the training data was augmented with the 233 GPT-labeled samples.

The performance of the model in both scenarios is presented in Table 2.

Table 2
Classification results on the original and augmented datasets.

Dataset	Accuracy	Precision	Recall	F1 Score
Original	0.8027	0.7732	0.8065	0.7921
Augmented	0.8212	0.7821	0.8109	0.7809

The results indicate that the model had an F1 score close to 0.80 on the original dataset. After augmenting the training data with samples generated by GPT, the performance metrics were similar, with only a very small change in the F1 score.

Since the change was small, we can assume that none of the synthetic data generated new patterns for the Support Vector Machine model to exploit for an improved classification task. The augmented samples appeared to provide a similar statistical distribution in that they did not make meaningful changes to the model's decision boundary. The results demonstrated that adding additional LLM-labeled examples in augmented samples did not yield improvements in performance for this model architecture and task. It is also possible that there were not performance improvements due to the large diversity and size of the original dataset from which the Support Vector Machine model could exploit.

6. Conclusion

In this research, we explored the potential of using synthetic data to support the task of code comment quality classification. A solid baseline was established with a Support Vector Machine model that led to an F1 score of 0.80 on an extensive human-annotated dataset. We primarily focused on the efficacy of synthetic data augmentation using GPT-3.5-turbo, and this initial analysis suggested that synthetic data labels did not improve performance.

These results imply that synthetic labels did not introduce enough novelty and variance to improve the performance of the model and/or additional high-quality human-annotated data had already pushed the baseline model towards the performance ceiling. Our research findings suggest that synthetic data is not helpful for certain types of low-N task when an adequate high-quality human-annotated initial data is available, and thus, naive synthetic data augmentation would provide diminishing returns.

Future work may implement better-performing architectures, such as transformer-based models, to take advantage of subtle variations in the data. Another avenue for future work is the consideration of more elaborate data augmentation approaches that seek to introduce more semantic variation would likely outperform typical label generation.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to perform grammar and spelling checks. After using this tool, the author(s) reviewed and edited the content as needed and take 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, A. Deshpande, P. P. Das, P. P. Chakrabarti, Comprehending c codes with llms: Effective comment generation through retrieval and reasoning, Pattern Recognition Letters (2025).

[10] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D. Clough, et al., Overview of the "information retrieval in software engineering"(irse) track at forum for information retrieval 2024, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024, pp. 18–21.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard, An activity-based quality model for

maintainability, in: 2007 IEEE International Conference on Software Maintenance, IEEE, 2007, pp. 184–193.
[17] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, J. Singer, Todo or to bug, in: 2008 ACM/IEEE 30th International Conference on Software Engineering, IEEE, 2008, pp. 251–260.
[18] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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).
[28] 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.
[29] 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).
[30] 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.