Can we predict Comment Quality using Machine Learning?

Ashwin Gaikwad[1,*]

[1]Indian Institute of Technology, Goa, 403401

Abstract

In software development, code comment quality holds varying importance, highlighting the need for effective assessment methods. This study aims to improve the classification of comment utility by combining traditional annotated datasets with synthetic data. For augmentation, we utilized GPT-3.5-turbo to label additional comments. A baseline model using logistic regression was implemented for classification, yielding an F1 score of around 0.80, with little change upon adding synthetic data. The research explores both the potential and limitations of using synthetic data augmentation to assess code comment relevance.

Keywords

Large Language Models, GPT-3.5, Logistic Regression, Comment Classification, Data Augmentation, Qualitative Analysis

1. Introduction

In today's digital era, software plays a crucial role in key sectors such as finance, healthcare, and transportation. Organizations must continually adapt to changing demands, resulting in frequent updates to existing software and the development of new applications. This continuous growth increases code complexity, which is necessary to support new features. Managing this large codebase is an essential phase of the Software Development Life Cycle (SDLC).

Rapid development cycles often lead to quick fixes, adding new code, or updating existing applications. These accelerated timelines can lead to less-than-optimal coding practices. As software evolves, associated documents, such as requirements and designs, may become outdated. In many cases, the original developers are not available for consultation. This underscores the need for structured, quality-driven processes in software development, with program comprehension being a key approach to maintaining existing code bases.

Given the dynamic nature of software, primary sources of reliable information become test execution traces, static code analysis, and code comments. This research focuses on code comments as indicators of software design for developers and automated tools. Comments provide context about the rationale and goals behind code, aiding in understanding and maintenance. However, comment quality varies, highlighting the need for automated tools to assess their usefulness.

One challenge in assessing code comment utility is the limited availability of well-annotated datasets that cover the diversity of comments across different programming contexts. To address this, innovative methods are needed to augment existing datasets and improve model performance on new, real-world comments. Our study combines manual data labeling with synthetic data augmentation using GPT-3.5-turbo.

In this paper, we tackle a binary classification task for source code comments in the C programming language, categorizing them as 'Useful' or 'Not Useful.' Starting with a dataset of over 11,000 manually labeled comments, we use logistic regression as an initial classification model. We then augment this dataset with over 200 additional samples labeled using GPT to evaluate performance improvements.

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

*Corresponding author.

ashwin.rajaram.23033@iitgoa.ac.in (A. Gaikwad)
0009-0001-4632-1470 (A. Gaikwad)

Notably, the model's performance remained consistent, maintaining an F1 score of 0.80 for both the original and augmented datasets.

This research contributes to the understanding of code comment utility classification by combining manual annotation with synthetic data augmentation. Our goal is to tackle current challenges and promote the development of adaptable models for the evolving landscape of software development.

The paper is organized as follows. Section 2 covers background work in comment classification. Section 3 details the task and dataset. Our methodology is discussed in Section 4. Results are presented in Section 5, and Section 6 concludes the study.

2. Related Work

Understanding a program automatically is a well-known research area among people working in the software domain. Numerous tools have been developed to aid in the extraction of knowledge from software metadata, including elements such as runtime traces and structural attributes of code [1, 2, 3, 4, 5, 6, 7, 8].

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. [9] 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.[10] 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. Another work[11] published in 2007 which proposed a two-dimensional maintainability model that explicitly associates system properties with the activities carried out during maintenance. The author claimed that this approach transforms the quality model into a structured quality knowledge base that is usable in industrial environments. Storey et al. did an empirical study on task annotations embedding within a source code and how it plays a vital role in a developer's task management[12]. The paper described how task management is negotiated between formal issue tracking systems and manual annotations that programmers include within their source code. Ted et al.[13] performed a 3×2 experiment to compare the efforts of procedure format with those of comments on the readability of a PL/I program. The readability accuracy was checked by questioning students about the program after reading it. The result said that the program without comment was the least readable. Yu Hai et al.[14] 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 [15] 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[15, 16, 17, 18, 19, 20], but still, automatic quality evaluation of source code comments is an important area and demands more research.

With the advent of large language models [21], 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 [22, 23] builds upon the methodologies proposed in [15, 24, 25] to investigate various vector space models [26] 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.

#	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
Sample data instance

3. Task and Dataset Description

In this paper, we tackle the task of developing a binary classification system to categorize source code comments as either *useful* or *not useful*. The system takes a comment and its associated code as input and outputs a label indicating the comment's relevance. Classical machine learning algorithms, such as logistic regression, can be employed to perform this classification. The two categories of comments are:

- *Useful* - The comment accurately describes or provides relevant information about the code.
- *Not Useful* - The comment does not convey relevant information about the code.

Our dataset comprises over 11,000 C code-comment pairs annotated by a team of 14 annotators to label comments as useful or not. Additionally, we created a supplementary dataset by obtaining code-comment pairs from GitHub and labeling them using GPT. This secondary dataset, structured similarly to the original, serves as an augmentation for our main dataset.

4. Working Principle

We use logistic regression to implement the binary classification functionality. The system takes comments as well as surrounding code snippets as input. We create embeddings of each piece of code and the associated comment using a pre-trained Universal sentence encoder. The output of the embedding process is used to train both machine learning model. The training dataset consists of 80% data instances along with their labels. The rest is used for testing, in both experiments. The description of the model is discussed in the following section.

4.1. Logistic Regression

We use logistic regression for the binary comment classification task which uses a logistic function to keep the regression output between 0 and 1. The logistic function is defined as follows:

$$Z = Ax + B \tag{1}$$

$$logistic(Z) = \frac{1}{1 + exp(-Z)} \tag{2}$$

The output of the linear regression equation (refer to equation 1) is passed to the logistic function (see equation 2). The probability value generated by the logistic function is used for binary class prediction based on the acceptance threshold. We keep the threshold value of 0.6 in favor of the *useful* comment class. We have a three-dimensional input feature extracted from each training instance which is passed to the regression function. The Cross entropy loss function is used during training for the hyper-parameter tuning.

5. Results

We train our logistic regression model on both datasets. The original dataset has 11,452 samples and the GPT generated data has 233 samples. The first experiment uses only the original data and produces the following scores.

After augmenting the original dataset with the GPT generated data, the following results were seen.

	Accuracy	Precision	Recall	F1 Score
Original Dataset	81.97293758	0.792349986	0.817765006	0.801166962
Augmented Dataset	81.2152332	0.794243029	0.803115991	0.798028602

Table 2
Results for binary classification on both datasets

The very slight change in the scores across metrics suggests that the newly generated data was practically indifferentiable from the original dataset, highlighting the validity of using GPT generated data for data augmentation.

6. Conclusion

This study investigated using synthetic data for code comment quality classification and established a logistic regression baseline with an F1 score of 0.80. Results indicate that synthetic augmentation did not improve classification, suggesting that traditional annotated data can still be highly effective in some cases. Future work could involve exploring more complex models or incorporating domain-specific knowledge.

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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.

[14] 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.

[15] 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.

[16] 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.

[17] 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.
[18] 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.
[19] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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).
[26] 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.