Exploring LLM-based Data Augmentation Techniques for Code Comment Quality Classification

Priyam Dalmia[1,*,†]

[1]American Express, Gurugram, Haryana, India - 122003

Abstract

In software engineering, the significance of code comments is not uniform, underscoring the requirement for precise methods that can discern their inherent quality. In this investigation, we aimed to improve the classification of the utility of code comments by integrating both traditionally annotated datasets and synthetic data augmentation techniques. For the latter, we utilized the capabilities of GPT-3.5-turbo, a contemporary linguistic model, to label additional comment instances. We used Logistic regression to create a baseline model for comment usefulness classification task. We observed that, irrespective of the inclusion of synthetic data, the classification efficacy remained consistent, recording an F1 score of approximately 0.80 both before and after the synthetic data amalgamation. This study elucidates the implications and boundaries of deploying synthetic data augmentation in the context of evaluating code comment relevance.

Keywords

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

1. Introduction

In today's digital age, software has embedded itself as the cornerstone of many pivotal sectors, ranging from finance and healthcare to transportation. As organizations tirelessly adapt to evolving requirements, the existing software is continually modified and new software gets written. Thus, the volume of source code increases constantly, leading to increased code complexity to support new software functionality. Maintaining this large amount of source code is a crucial phase of Software Development Life Cycle (SDLC).

Rapid cycles of development often force quick and dirty resolution of bugs, introduction of new source code, or the updating of existing applications. Such accelerated timelines can result in suboptimal coding practices. As software undergoes changes, associated documentation, such as requirement specifications and high-level designs, may become outdated. In many cases, prior developers' insights or assistance is unavailable. This highlights the need for structured, quality-focused development processes, with program comprehension serving as a key method for maintaining existing source code.[1].

*Forum for Information Retrieval Evaluation, December 15-18, 2023, India

*Corresponding author.

priyam.dalmia@aexp.com (P. Dalmia)

0009-0002-8566-0632 (P. Dalmia)

Considering the evolving nature of software design, the primary dependable sources of information become test execution traces, static analyses of programs, and code comments. This research emphasizes the importance of code comments as indicators of program design for both developers and automated systems. Code comments provide context on the reasoning and objectives of the associated source code, facilitating better understanding and maintenance. However, the quality of comments varies, necessitating automated tools to evaluate their usefulness.

A recurring challenge in studying code comment usefulness is the limited availability of comprehensive, well-annotated datasets that cover the diverse nature of comments across different programming contexts. To address this, there's a need for innovative methods to augment existing data to enhance model performance on new, real-world comments. Our approach in this study combines manual data labeling with synthetic data augmentation using the GPT-3.5-turbo language model.

In this paper, we focus on a binary classification task for source code comments in the C language, categorizing them as 'Useful' or 'Not Useful'. We begin with a dataset of over 11,000 manually-labeled comments and use Logistic Regression for initial comment classification. This dataset is then augmented with more than 200 GPT-labelled samples to test for performance improvements. Interestingly, the model's performance was consistent, yielding an F1 score of 0.80 for both the original and augmented datasets.

Through this study's combination of manual annotation and synthetic data augmentation, we aim to provide a contribution to the current understanding of code comment usefulness classification. The goal is to address existing challenges and promote the development of adaptable models for the ever-changing landscape of software development.

The rest of the paper is organized as follows. Section 2 discusses the background work done in the domain of comment classification. The task and dataset are described in 3. Our methodology is discussed in section 4. Results are addressed in section 5. Section 6 concludes the paper.

2. Related Work

Software metadata [2] plays a crucial role in the maintenance of code and its subsequent understanding. Numerous tools have been developed to assist in extracting knowledge from software metadata, which includes runtime traces and structural attributes of code [3, 4, 5, 6, 7, 8, 9, 10, 11].

In the realm of mining code comments and assessing their quality, several authors have conducted research. Steidl et al. [12] employ techniques such as Levenshtein distance and comment length to gauge the similarity of words in code-comment pairs, effectively filtering out trivial and non-informative comments. Rahman et al. [13] focus on distinguishing useful from non-useful code review comments within review portals, drawing insights from attributes identified in a survey conducted with Microsoft developers [14]. Majumdar et al. [15, 16, 17, 18] have introduced a framework for evaluating comments based on concepts crucial for code comprehension. Their approach involves the development of textual and code correlation features, utilizing a knowledge graph to semantically interpret the information within comments. These approaches employ both semantic and structural features to address the prediction

problem of distinguishing useful from non-useful comments, ultimately contributing to the process of decluttering codebases

In light of the emergence of large language models, such as GPT-3.5 or llama [19], it becomes crucial to assess the quality of code comments and compare them to human interpretation. The IRSE track at FIRE 2023 [20] expands upon the approach presented in a prior work [15]. It delves into the exploration of various vector space models [21] and features for binary classification and evaluation of comments, specifically in the context of their role in comprehending code. Furthermore, this track conducts a comparative analysis of the prediction model's performance when GPT-generated labels for code and comment quality, extracted from open-source software, are included.

3. Task and Dataset Description

In this section, we have described the task addressed in this paper. We aim to implement a binary classification system to classify source code comments into *useful* and *not useful*. The procedure takes a code comment with associated lines of code as input. The output will be a label such as *useful* or *not useful* for the corresponding comment, which helps developers comprehend the associated code. Classical machine learning algorithms such as logistic regression can be used to develop the classification system. The two classes of source code comments can be described as follows:

- *Useful* - The given comment is relevant to the corresponding source code.
- *Not Useful* - The given comment is not relevant to the corresponding source code.

A dataset consisting of over 11000 code-comment pairs written in C language is used in our work. Each instance of data consists of comment text, a surrounding code snippet, and a label that specifies whether the comment is useful or not. The whole dataset is collected from GitHub and annotated by a team of 14 annotators. A sample data is illustrated in table 1.

There is another similar dataset that is created and used in this work. That dataset is created by getting code-comment pairs from Github, and the label of useful or not useful was given by GPT. This dataset has a similar structure to the original dataset, and is used to augment the original dataset later on.

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.

#	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

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 paper has addressed a binary classification problem in the domain of source code comment classification. The classification has been done based on the usefulness of the comment present within a source code written in C language. We have used logistic regression as our base classification method. We conducted two experiments, one with the original dataset and another with the original dataset plus the synthetic GPT generated data. The similar results in both cases show that the synthetic data falls in line with the original dataset, and how synthetic data creation can help in effectively increasing data volume required for training models. The synthetic data's correctness as compared to the original dataset is proven by the results shown above. Synthetic data generation can help a lot with data augmentation, finding its use in many pipelines.

References

[1] M. Berón, P. R. Henriques, M. J. Varanda Pereira, R. Uzal, G. A. Montejano, A language processing tool for program comprehension, in: XII Congreso Argentino de Ciencias de la Computación, 2006.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[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] 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.

[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] 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.

[14] 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.

[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: Forum for Information Retrieval Evaluation, ACM, 2022.
[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] 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.
[20] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, B. Dave, 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, in: Forum for Information Retrieval Evaluation, ACM, 2023.
[21] 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.