Source Code Comment Classification

Devyani Remulkar[1,*]

[1]Indian Institute of Technology, Goa, 403401

Abstract

In software engineering, the significance of code comments is not uniform, emphasizing the necessity for effective strategies to evaluate their quality. In this research, we aimed to improve how we classify the usefulness of comments by combining traditionally labeled datasets with synthetic data created through augmentation techniques. For the latter, we utilized the capabilities of GPT-3.5-turbo, a highly advanced and effective model designed for understanding and generating human language, 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 whether or not we used synthetic data. This study highlights both the benefits and limitations of using synthetic data to evaluate the relevance of code comments.

Keywords

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

1. Introduction

In the current digital landscape, software serves as a foundational element in numerous critical sectors, including finance, healthcare, and transportation. As organizations continuously adapt to changing demands, existing software undergoes frequent modifications, and new code is 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 require quick solutions for bugs, new features, or updates, which can result in poor coding practices. As software evolves, accompanying documentation, such as requirement specifications and high-level designs, may fall out of date. In many cases, prior developers' insights or assistance is unavailable. This situation highlights the need for organized and quality-focused development processes, with understanding the code being a crucial strategy for managing existing software.[1].

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 study highlights how important code comments are for understanding program design, both for developers and automated systems. Comments give context about the reasons behind the code and its goals, which helps with understanding and maintaining the software. However, the quality of comments can differ, so there's a need for automated tools to assess how useful they are.

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 problem, new strategies are needed to improve the current 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

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

[*]Corresponding author.

devyani.remulkar.22031@iitgoa.ac.in (D. Remulkar)
0009-0002-8566-0632 (D. Remulkar)

comments and use Logistic Regression for initial comment classification. This dataset is then augumented with more than 200 samples labeled by GPT to assess potential performance improvements. Notably, the model's performance remained stable, with 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 structure of the paper is organized as follows. Section 2 outlines related work in the field of comment classification. The task and dataset are described in Section 3. Our methodology is detailed in Section 4. The results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Software metadata [2] is very important for maintaining and understanding code. Many tools have been created to help extract information from software metadata, which includes runtime traces and the structure of the code [3, 4, 5, 6, 7, 8, 9, 10, 11].

In the area of analyzing code comments and their quality, several researchers have done important work. Steidl et al. [12] use methods like Levenshtein distance and comment length to measure how similar words are in code-comment pairs, helping to filter out unhelpful comments. Rahman et al. [13] focus on identifying useful and non-useful comments in code reviews, using insights from a survey of Microsoft developers [14]. Majumdar et al. [15, 16, 17, 18] have created a framework to evaluate comments based on key concepts for understanding code. Their method includes developing features that link text and code, and they use a knowledge graph to understand the meaning of comments. These approaches combine both semantic and structural features to help predict which comments are useful, ultimately helping to clean up codebases.

With the rise of large language models like GPT-3.5 and 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] builds on earlier research [15]. It investigates different vector space models [21] and features for classifying and evaluating comments, especially regarding their usefulness in understanding code. This track also compares how well models perform when using labels generated by GPT to rate the quality of code and comments from open-source software

3. Task and Dataset Description

This section outlines the task undertaken in this research. Our objective is to implement a binary classification system to categorize source code comments as *useful* or *not useful.* The input for the system consists of a code comment along with the corresponding lines of code. The output will be a label, either *useful* or *not useful*, indicating the relevance of the comment to the associated code, aiding developers in understanding the code's purpose. Traditional machine learning algorithms, such as logistic regression, can be employed to develop the classification system. The two classes of source code comments are defined 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.

The dataset used in our study contains over 11,000 pairs of code comments and code snippets written in C language, each labeled to show whether the comments are 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.

Another similar dataset has been created for this work. It was made by collecting code-comment pairs from GitHub, with labels indicating whether the comments are useful or not provided by GPT.

#	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

This dataset has a similar structure to the original one and is used to augument the original dataset later on.

4. Methodology

The methodology for classifying code comments comprises several key steps, outlined as follows:

1. **Dataset Preparation**: This stage involves collecting the existing dataset of over 11,000 labeled comments and their associated code snippets. We also integrate additional synthetic comments generated by the GPT-3.5-turbo model, enhancing the dataset's size and diversity.
2. **Feature Engineering**: In this phase, we create features for our classification model, which may include factors like comment length, the presence of specific keywords, and a semantic analysis of how the comment content relates to the code.
3. **Model Training**: We employ Logistic Regression to build our classification model. This algorithm is chosen due to its simplicity and effectiveness in binary classification tasks. The model is trained on the prepared dataset, utilizing both the original and augmented data.

4. **Evaluation**: The model's performance is evaluated using standard metrics, including accuracy, precision, recall, and F1 score, to determine the effectiveness of comment classification.

We use logistic regression for binary classification. The system takes both comments and their corresponding code snippets as input. We create embeddings for each code segment and its related comment using a pre-trained Universal Sentence Encoder. The output of this embedding process is then used to train the machine learning model. The training dataset consists of 80% of the data instances along with their labels, while the remaining 20% is set aside for testing in both experiments.

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 from the linear regression equation (as shown in equation 1) is fed into the logistic function (refer to equation 2). The probability generated by the logistic function is then used for binary class prediction, based on an acceptance threshold. We set this threshold at 0.6 to give priority to the **useful** comment class. Each training instance is represented by a three-dimensional input feature vector, which is input into the regression function. During training, we use the Cross-Entropy loss function for tuning the hyperparameters.

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.Our baseline model, developed with Logistic Regression, yielded an F1 score of approximately 0.80 when tested on the original dataset.

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 addresses a binary classification problem related to classifying source code comments. The classification is based on the usefulness of comments in C language source code. We used logistic regression as our main classification method and conducted two experiments: one with the original dataset and another that included both the original dataset and the synthetic data generated by GPT. The similar results in both experiments indicate that the synthetic data aligns well with the original dataset and demonstrates how creating synthetic data can effectively increase the volume of data needed for training models. The accuracy of the synthetic data compared to the original is supported by the results presented. Overall, synthetic data generation is valuable for data augmentation and can be beneficial in various applications.

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] 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.