Classification of Code and Comments in a Machine Learning Framework

Adwita Deshpande[1,*]

[1]Indian Institute of Technology, Goa, 403401

Abstract

In modern software development, source code comments serve as valuable guides for understanding the purpose and structure of code, yet their quality varies widely. This study focuses on assessing code comment usefulness, a task critical for program comprehension, especially within rapidly evolving codebases. We undertake a binary classification of C language comments into 'Useful' or 'Not Useful,' leveraging both a manually labeled dataset of over 11,000 comments and an additional 200 synthetic samples generated by GPT-3.5-turbo. Using Logistic Regression, we train and evaluate the classification model across both original and augmented datasets, observing an F1 score of 0.80, indicating that data augmentation did not significantly alter performance. Our results suggest that combining manual annotations with synthetic data can stabilize model performance and broaden comment diversity, supporting the development of automated tools to maintain code documentation quality across software systems.

Keywords

Large Language Models, GPT-3.5, Logistic Regression, Comment Classification, Data Augmentation, Qualitative Analysis, Binary classification Machine learning

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. Consequently, the volume of source code consistently rises, leading to greater complexity in the codebase to accommodate new functionalities. Effectively maintaining this extensive code is a vital component of the Software Development Life Cycle (SDLC).

Fast-paced development cycles often necessitate quick fixes for bugs, the addition of new code, or updates to existing applications, which can lead to less-than-ideal coding practices. As software evolves, accompanying documentation, such as requirement specifications and high-level designs, may fall out of date. In many cases, previous developers' insights or assistance might not be accessible. This scenario underscores the necessity for structured, quality-driven development processes, with program comprehension serving as a key strategy for managing existing code.[1].

Given the dynamic nature of software design, code comments become one of the most reliable and accessible sources for developers to understand a program's intent, especially when other documentation is lacking or outdated. Comments that are clear and relevant provide insights into a developer's rationale, clarifying complex logic or unique design choices. However, because comment quality can vary widely, determining the "usefulness" of a comment is crucial for creating reliable software maintenance tools that streamline code comprehension, reduce errors, and improve efficiency.

A persistent issue in investigating the usefulness of code comments is the scarcity of extensive, well-annotated datasets that reflect the diverse nature of comments across various programming environments. To mitigate this issue, innovative strategies are necessary to augment existing data, thereby improving model performance on new, real-world comments. Our approach combines manual data labeling with synthetic data augmentation through the GPT-3.5-turbo language model.

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

*Corresponding author.

adwita5b@gmail.com (A. Deshpande)

0009-0003-9442-3441 (A. Deshpande)

In this work, we focus on a binary classification task for comments in C language source code, categorizing them as 'Useful' or 'Not Useful'. We commence with a dataset comprising over 11,000 manually annotated comments and employ Logistic Regression for initial classification. This dataset is further enhanced 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.

By integrating manual annotation with synthetic data augmentation, this study contributes to the understanding of code comment usefulness classification. Our aim is to address existing challenges and facilitate the development of adaptable models suitable for the ever-evolving 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 plays a crucial role in maintaining code and understanding it over time. Numerous tools have been developed to assist in extracting valuable knowledge from software metadata, such as runtime traces or structural code attributes [2, 3, 4, 5, 6, 7, 8, 9].

To mine and evaluate code comments, researchers have developed various approaches that analyze the quality of comments by comparing code-comment pairs. In terms of mining code comments and assessing the quality, authors [10, 11, 12, 13, 14, 15, 16, 17] compare the similarity of words in code-comment pairs using the Levenshtein distance and length of comments to filter out trivial and non-informative comments. Rahman et al. [18] detect useful and non-useful code review comments (logged-in review portals) based on attributes identified from a survey conducted with developers of Microsoft [19]. Majumdar et al. [20, 21] proposed a framework to evaluate comments based on concepts that are relevant for code comprehension. They developed textual and code correlation features using a knowledge graph for semantic interpretation of information contained in comments. These approaches use semantic and structural features to design features to set up a prediction problem for useful and not useful comments that can be subsequently integrated into the process of decluttering codebases.

With the advent of large language models [22], 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 [23, 24] extends the approach proposed in [20, 25, 26, 13] to explore various vector space models [27] and features for binary classification and evaluation of comments in the context of their use in understanding the code. This track also compares the performance of the prediction model with the inclusion of the GPT-generated labels for the quality of code and comment snippets extracted 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 comment is pertinent to the corresponding source code.
- **Not Useful** - The comment is irrelevant to the corresponding source code.

The dataset utilized in our study consists of over 11,000 pairs of code comments and code snippets, sourced from various open-source projects, each annotated with labels indicating their usefulness or lack thereof. This dataset serves as the foundation for our classification model. To enhance the dataset, we additionally generated synthetic data using the GPT-3.5-turbo language model. This augmentation process involves the model creating code comments, which are then manually verified and classified, resulting in more than 200 extra samples labeled by the model.

#	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. Methodology

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

1. **Dataset Preparation**: This stage involves gathering the existing dataset of over 11,000 labeled comments and their corresponding code snippets. The additional synthetic comments generated through the GPT-3.5-turbo model are integrated into the dataset, increasing its size and diversity.
2. **Feature Engineering**: In this phase, we construct features for our classification model, which may include attributes such as comment length, presence of certain keywords, and semantic

analysis of the comment's content relative 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 utilize logistic regression to perform binary classification. The system accepts both comments and the corresponding code snippets as input. Using a pre-trained Universal Sentence Encoder, we generate embeddings for each code segment and its associated comment. The results of this embedding process are employed to train the machine learning model. The training dataset comprises 80% of the data instances along with their labels, while the remaining 20% is reserved 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 input into the logistic function (refer to equation 2). The probability value produced by the logistic function is then utilized for binary class prediction, guided by an acceptance threshold. We set this threshold at 0.6 to prioritize the useful comment class. Each training instance is represented by a three-dimensional input feature vector, which is fed into the regression function. During training, the Cross-Entropy loss function is employed for hyperparameter tuning.

For feature extraction, we use the Universal Sentence Encoder, a pre-trained deep learning model that encodes textual data into high-dimensional vectors. By representing each code comment and its associated code snippet in this way, we capture semantic relationships between comments and code, facilitating more accurate classifications by highlighting patterns that logistic regression can effectively learn from.

5. Results

The performance of the classification model was analyzed based on the F1 score, a widely used metric that accounts for both precision and recall. 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	81.88293758	0.793348986	0.817565024	0.801166962
Augmented Dataset	81.21523324	0.792243023	0.803115991	0.798028602

Table 2
Results for binary classification on both datasets

The results demonstrate that while synthetic data can enhance dataset size and variety, it does not necessarily lead to improved classification accuracy in this specific task. This outcome could be due to subtle differences between manually annotated and GPT-generated data, as synthetic comments might lack some contextual intricacies present in authentic developer comments. Nonetheless, the stable F1 score across both datasets suggests that the model effectively generalizes from the manually annotated comments, underscoring the importance of high-quality human-labeled data in creating reliable software documentation tools.

6. Conclusion

This study provides a framework for assessing the usefulness of code comments, demonstrating the feasibility of automated binary classification in identifying relevant comments within source code. Our study reveals that, while synthetic data augmentation can enrich dataset variety, it does not guarantee a performance boost, as indicated by the stable F1 scores across both the original and augmented datasets. This outcome underscores the importance of high-quality, manually annotated data in creating effective models for software documentation tasks. Moving forward, incorporating semantic and contextual features may further improve classification accuracy and help address the challenges of evolving codebases. Our findings contribute toward establishing adaptable and scalable comment classification models, ultimately enhancing program comprehension and supporting efficient code maintenance.

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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access (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] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).
[17] 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.
[18] 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.
[19] 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.
[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] 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.
[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, Generative ai for code metadata quality assessment, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024.
[24] 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.
[25] 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.
[26] 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).
[27] 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.