Code and Comments Categorization in terms of Program Comprehension

Abhinav Daggubelli[1,*]

[1]Indian Institute of Technology, Goa, 403401

Abstract

In software development, the effectiveness of code comments can differ significantly, highlighting the need for methods that can accurately assess their genuine value. This study aims to improve the classification of code comment usefulness by implementing a hybrid approach that integrates manually tagged datasets with synthetic data augmentation. For the augmentation process, we utilized GPT-3.5-turbo, a leading language model, to generate additional labeled examples of comments. We established a baseline classification model using random forests. Notably, even with the addition of synthetic data, the model's performance remained stable, achieving an F1 score of around 0.79 both prior to and following the integration of synthetic data. This research provides valuable insights into both the advantages and limitations of using synthetic data augmentation in the classification of code comment usefulness.

Keywords

Random Forest, Data Augmentation, Comment Classification, Qualitative Analysis

1. Introduction

Developers frequently face the challenge of fixing bugs, creating new source code, or upgrading existing applications under tight deadlines. This pressure can result in subpar coding practices. As software evolves, accompanying documentation—such as requirements specifications and high-level designs—can become outdated and insufficient, often complicating knowledge transfer when assistance from previous developers is unavailable. Such circumstances highlight the need for a systematic, quality-controlled development process. Automated program comprehension serves as an effective approach for enhancing the maintenance of existing source code, ensuring better management and understanding of the codebase.[1].

Given that the software design of a codebase is constantly evolving, the most reliable sources of truth are the traces from test executions, static program analyses, and, significantly, code comments. This paper centers on code comments as valuable insights into program design, beneficial for both developers and automated program comprehension systems. Code comments provide essential understanding of the logic, decisions, and intentions behind the code, facilitating better comprehension, maintenance, and debugging. However, not all comments carry the same level of informative value, underscoring the need for automated methods to effectively classify their usefulness.

A persistent challenge in studies of code comment usefulness is the limited availability of extensive, well-annotated datasets that capture the diverse nature of comments across different programming contexts. This situation calls for innovative strategies to enhance existing data for better model generalization when dealing with unseen, real-world comments. Addressing this gap, we propose a hybrid approach that combines manual annotations with synthetic data augmentation. We utilize GPT-3.5-turbo, a cutting-edge language model, to label code comment samples extracted from open-source codebases.

In this paper, we introduce a binary classification task aimed at evaluating source code comments within C programs, categorizing each comment as either Useful or Not Useful. We begin with a training

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

*Corresponding author.

daggubelli.abhinav.23031@gmail.com (A. Daggubelli)

0000-0002-4551-748X (A. Daggubelli)

dataset consisting of over 11,000 manually annotated samples. Using Random Forests, we establish a baseline for comment classification and subsequently augment this dataset with over 200 GPT-generated labels to assess any performance enhancements. Interestingly, we found that the model's performance remained stable, with an F1 score of 0.79 for both the baseline and the model trained on the augmented dataset.

By investigating the nuanced relationship between manual annotations and synthetic data augmentation, this study seeks to contribute a fresh perspective to the existing knowledge on code comment usefulness classification. It aims to provide an innovative solution to the ongoing challenges in the field, encouraging further exploration and development of robust, scalable models that can adapt effectively to 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 is integral to code maintenance and subsequent comprehension. A significant number of tools [2, 3, 4, 5, 6, 7] have been proposed to aid in extracting knowledge from software metadata [8] like runtime traces or structural attributes of codes.

In terms of mining code comments and assessing the quality, authors [9, 10, 11, 12, 13, 14] 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. [15] detect useful and non-useful code review comments (logged-in review portals) based on attributes identified from a survey conducted with developers of Microsoft [16]. Majumdar et al. [17, 18] 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 emergence of large language models [19], it has become essential to evaluate how the quality of code comments assessed by standard models like GPT-3.5 or LLaMA compares with human interpretations. The IRSE track at FIRE 2023 [20] builds on the methodology introduced in [17], investigating various vector space models [21] and features for the binary classification and evaluation of comments, specifically in the context of their utility in code comprehension. This track also examines the performance of the prediction model when incorporating GPT-generated labels to assess the quality of code and comment snippets sourced from open-source software.

3. Task and Dataset Description

In this section, we outline the task addressed in this paper, which involves implementing a binary classification system designed to categorize source code comments as either useful or not useful. The process begins with inputting a code comment along with its associated lines of code. The output will be a label, indicating whether the comment is deemed useful or not useful, thereby assisting developers in understanding the related code more effectively. To develop this classification system, classical machine learning algorithms, such as random forests, will be employed. The two categories 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.

Our study utilizes a dataset comprising over 11,000 code-comment pairs written in the C programming language. Each data instance includes the comment text, a corresponding code snippet, and a label

#	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

indicating whether the comment is useful or not. This comprehensive dataset was sourced from GitHub and annotated by a team of 14 annotators. A sample of the data is presented in Table 1.

In addition to the primary dataset, we have created another similar dataset for this study. This new dataset consists of code-comment pairs sourced from GitHub, with labels indicating whether the comments are useful or not assigned by GPT. It shares a comparable structure to the original dataset and is utilized to augment the original data in subsequent analyses.

4. Working Principle

We employ random forests to implement the binary classification functionality, where the system takes both comments and their surrounding code snippets as input. To facilitate this, we generate embeddings for each code snippet and its corresponding comment using a pre-trained Universal Sentence Encoder. The resulting embeddings are then used 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. A detailed description of the model is provided in the following section.

4.1. Random Forest

In our study, we utilize Random Forest (RF) for binary comment classification, taking advantage of an ensemble of decision trees to enhance the model's predictive accuracy while mitigating the risk of overfitting. The fundamental principle of Random Forest involves generating multiple decision trees during the training process. During the prediction phase, the model outputs the class that corresponds to the majority vote among the classes predicted by the individual trees.

Each tree in the Random Forest is constructed as follows:

1. A subset of the training data is selected with replacement (bootstrap sample).
2. A subset of features is randomly chosen at each node.
3. The best split based on a criterion (such as Gini impurity or entropy) is chosen to partition the data.
4. Steps 2 and 3 are repeated at each node until the tree is fully grown.

The classification decision is obtained by aggregating the predictions made by all trees in the forest through majority voting:

$$RF(x) = \text{majority}\left(\{T_i(x)\}_{i=1}^{n}\right) \tag{1}$$

where $T_i(x)$ denotes the prediction of the i-th tree for the input vector x, and n is the number of trees in the forest. A threshold of 0.5 is typically employed for binary classification; however, this threshold can be adjusted to prioritize the **useful** comment class, mirroring the threshold modification approach used in Random Forests.

Random Forest inherently manages multi-dimensional feature spaces and does not necessitate feature scaling. It effectively addresses missing values by selecting splits that minimize impurity among non-missing values, thereby imputing the missing values based on the majority class or the mean/mode value.

During the training process, the out-of-bag (OOB) error—calculated on the data not included in bootstrap samples—provides an unbiased estimate of the generalization error, which can be utilized for hyperparameter tuning.

5. Results

We train our Random Forest model on both datasets. The original dataset comprises 11,452 samples, while the dataset generated by GPT contains 233 samples. In the first experiment, we utilize only the original data, resulting in 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	81.05630729	0.790190835	0.801640488	0.794906015
Augmented Dataset	81.0012837	0.790785274	0.801383776	0.795175139

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 in the field of source code comment classification, focusing on the usefulness of comments within C language source code. We employed Random Forests as our primary classification method. We conducted two experiments: one utilizing only the original

dataset and the other incorporating both the original dataset and the synthetic GPT-generated data. The similar results obtained in both experiments indicate that the synthetic data aligns well with the original dataset, demonstrating how synthetic data generation can effectively enhance the volume of data required for training models. The accuracy of the synthetic data, in comparison to the original dataset, is supported by the results presented. Overall, synthetic data generation proves to be a valuable strategy for data augmentation, with potential applications in various pipelines.

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. 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] 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.

[9] 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.

[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] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[12] 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.

[13] 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[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.