Source Code Comment Classification

Atharva Keny[1,*]

[1]Indian Institute of Technology, Goa, India - 403401

Abstract

In software development, the quality of code comments can greatly impact how well developers understand the code. This study focuses on improving the classification of code comments into categories of usefulness by combining real, manually labeled examples with synthetic data. We used the GPT-3.5-turbo language model to generate additional comment examples, enriching our dataset for better analysis. Our baseline classification model was built using random forests. Interestingly, although we introduced synthetic data to enhance model performance, the F1 score remained stable at around 0.79 before and after augmentation. This research highlights both the potential and the challenges of using synthetic data to improve the classification of source code comments, laying the groundwork for further investigation in this field.

Keywords

Random Forests, Data Augmentation, Comment Classification, Qualitative Analysis

1. Introduction

Software developers often face tight deadlines, leading to improper coding practices. As software evolves, related documentation can become outdated, and the original developers may be unavailable for guidance. This creates a need for methods like automated program comprehension to maintain and improve existing code [1].

Among the primary sources of information in a codebase are code comments, which provide valuable insights into the logic and decisions behind the code. However, not all comments are equally helpful, making it necessary to develop automated methods to classify the usefulness of comments effectively.

A major challenge in this task is the lack of large, annotated datasets representing diverse code comments across different contexts. This paper addresses this issue by augmenting a manually annotated dataset with synthetic data generated by GPT-3.5-turbo, a state-of-the-art language model. We propose a binary classification task to categorize C language code comments as either "useful" or "not useful." Using random forests as a baseline model, we find that the synthetic data did not improve the performance significantly, with the F1 score remaining stable at 0.79.

This study contributes to the understanding of how manual annotation and synthetic data augmentation interact in the task of code comment usefulness classification, offering new perspectives for future research in this domain.

The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 describes the task and dataset. Section 4 explains our methodology. Results are presented in Section 5, and conclusions are drawn in Section 6.

2. Related Work

Code comments are an essential aspect of software maintenance and understanding. Numerous tools [2, 3, 4, 5, 6, 7] have been developed to extract and analyze metadata from source code [8], such as runtime traces and structural code attributes.

Several studies [9, 10, 11, 12, 13, 14] have explored methods for filtering and assessing the quality of code comments. For example, Rahman et al. [15] identified useful and non-useful code review comments

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

*Corresponding author.

atharvakeny.0812.2004@gmail.com (A. Keny)

based on developer surveys at Microsoft [16]. More recently, large language models [17] have been employed to evaluate comments and their relevance to code [18, 19].

This work builds on these efforts by exploring the impact of synthetic data generated by GPT-3.5 on comment classification models.

3. Task and Dataset Description

In this section, we outline the task addressed in this paper. Our goal is to implement a binary classification system that categorizes source code comments into two distinct classes: *useful* and *not useful*. The system takes a code comment along with its corresponding lines of code as input and generates a label indicating whether the comment is *useful* or *not useful*. This classification aids developers in understanding the associated code more effectively. We utilize traditional machine learning algorithms, such as random forests, to build the classification model. The two categories of source code comments are defined as follows:

- *Useful* - The comment is pertinent to the associated source code.
- *Not Useful* - The comment does not provide relevant information about the associated source code.

Our study employs a dataset containing over 11,000 pairs of code comments and corresponding C language code snippets. Each data instance comprises the comment text, a related code snippet, and a label that indicates whether the comment is considered useful. This dataset was sourced from GitHub and annotated by a team of 14 annotators. A sample instance of this data is presented in Table 1.

In addition to the primary dataset, we created a similar dataset for this research. This secondary dataset was generated by extracting code-comment pairs from GitHub, with labels of useful or not useful assigned by GPT. The structure of this dataset closely resembles that of the original dataset and is intended to augment the original dataset in subsequent analyses.

4. Methodology

We implemented a binary classification system using random forests, which operates by training an ensemble of decision trees. The model takes both the comment and surrounding code as input, which are converted into embeddings using a pre-trained Universal Sentence Encoder.

4.1. Random Forest

In this study, Random Forest (RF) is employed for binary comment classification, utilizing an ensemble of decision trees to enhance predictive accuracy and mitigate overfitting. The core idea behind Random Forest is to generate multiple decision trees during training and predict the class that is the mode of the individual tree predictions during the inference phase.

The construction of each tree in the Random Forest follows these steps:

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

The final classification is determined by aggregating the predictions from all trees in the forest through majority voting:

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

#	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
Example of a data instance

where $T_i(x)$ represents the prediction of the i-th tree for the input vector x, and n is the total number of trees in the forest. A default threshold of 0.5 is typically used for binary classification, though this can be adjusted to bias the model towards the "useful" comment class, similar to threshold adjustment techniques in Random Forest.

Random Forest can effectively handle multi-dimensional feature spaces without requiring feature scaling. It addresses missing values by selecting splits that minimize impurity among non-missing values, and imputes missing values based on the majority class or mean/mode as applicable.

During training, the out-of-bag (OOB) error, computed from data not included in the bootstrap samples, provides an unbiased estimate of the model's generalization error and can be used for hyper-parameter tuning.

5. Results

We trained our Random Forest model on both the original dataset and the augmented dataset. The original dataset consisted of 11,452 samples, while the GPT-generated data added 233 samples. In the first experiment, we used only the original dataset, yielding the following performance metrics.

After incorporating the GPT-generated data to augment the original dataset, we observed the following results:

	Accuracy	Precision	Recall	F1 Score
Original Dataset	81.06%	0.7902	0.8016	0.7949
Augmented Dataset	81.00%	0.7908	0.8014	0.7952

Table 2
Performance metrics for binary classification using both datasets

The minimal variation in the results across all metrics indicates that the GPT-generated data closely mirrored the original dataset in terms of quality, demonstrating the effectiveness of synthetic data augmentation.

6. Conclusion

This paper presents a binary classification system for code comment usefulness in C programs, using random forests as the primary model. Our experiments showed that synthetic data generated by GPT-3.5-turbo performs similarly to manually annotated data. This demonstrates the value of synthetic data augmentation in machine learning tasks, particularly for expanding training datasets in resource-constrained settings.

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] 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.

[18] 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.

[19] 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.