Code and Comment Classification in C using Machine Learning

SAHIL WALUNJ[1,*]

[1]*Indian Institute of Technology, Goa, 403401*

Abstract

In software engineering, the importance attributed to code comments is inconsistent, emphasizing the necessity for effective strategies to evaluate their intrinsic quality. This research aimed to enhance the classification of comment utility by merging traditionally annotated datasets with synthetic data created through augmentation techniques.For augmentation, we employed GPT-3.5-turbo, a state-of-the-art language model, to label additional comment examples. A baseline model was established using random forests for classification. Interestingly, despite the data augmentation, the model performance remained consistent, with an F1 score of approximately 0.79 both before and after the synthetic data integration. This research offers insights into the potential and limitations of synthetic data augmentation in the realm of code comment usefulness classification.

Keywords

Random Forests, Data Augmentation, Comment Classification, Qualitative Analysis

1. Introduction

In today's digital age, software is a crucial component in various key industries, including finance, healthcare, and transportation. As organizations adapt to evolving demands, software is regularly modified, and new code is continually developed. This rapid development often requires developers to fix bugs, generate fresh source code, or update existing applications within limited time frames, which can sometimes lead to less-than-ideal coding practices. As software grows and changes, documents like requirement specifications and high-level designs frequently become outdated, and knowledge from prior developers may not always be accessible. This underscores the need for a structured, quality-focused development process. One method for managing and understanding existing code is through automated program comprehension [1].

As codebases evolve quickly, traditional documentation tends to lag, making it harder to keep a current understanding of the software. Therefore, developers increasingly rely on test execution traces, static code analysis, and especially code comments for insight into the program structure. This study focuses on code comments as essential sources for understanding program design, supporting both developers and automated comprehension tools. Comments often reveal the rationale, decisions, and intentions within the code, aiding in critical tasks like comprehension, maintenance, and debugging. However, the consistency of comment quality varies, highlighting the need for automated methods to assess their value effectively.

A significant challenge in studying the effectiveness of code comments is the limited availability of large, well-annotated datasets that capture a variety of comments across different programming contexts. To address this, innovative data augmentation techniques are essential to enhance model performance on real-world comments. Our approach integrates manual data labeling with synthetic augmentation using the GPT-3.5-turbo language model.

In this paper, we propose a binary classification task aimed at evaluating code comments in C programs. Each comment is categorized into one of two classes—Useful or Not Useful. We begin with a dataset of over 11,000 manually labeled samples and establish a Random Forest model as a baseline for

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

*Corresponding author.

sahil.shivaji.23031@iitgoa.ac.in (S. WALUNJ)

0009-0008-2316-360X (S. WALUNJ)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-4054/T9-4.pdf

classifying comments. Additionally, we augment the dataset with over 200 samples labeled using GPT, allowing us to analyze any performance gains. Our findings show consistent performance, with an F1 score of 0.79 for both the baseline and the model trained with augmented data.

Through the exploration of manual annotation combined with synthetic augmentation, this study contributes fresh insights to the field of code comment usefulness classification. It aims to offer a new solution to the ongoing challenges in this area, encouraging further research and the development of robust, scalable models that can adapt to the ever-evolving landscape of software development.

The paper is organized as follows: Section 2 reviews the background work in comment classification. The task and dataset are described in Section 3. Section 4 explains our methodology. Results are presented in Section 5, and Section 6 provides the conclusion.

2. Related Work

Understanding a program automatically is a well-known research area among people working in the software domain. Numerous tools have been developed to aid in the extraction of knowledge from software metadata, including elements such as runtime traces and structural attributes of code [2, 3, 4, 5, 6, 7, 8, 9].

New programmers generally check for existing comments to understand a code flow. Although, every comment is not helpful for program comprehension, which demands a relevancy check of source code comments beforehand. Many researchers worked on the automatic classification of source code comments in terms of quality evaluation. For example, Omal et al.[10] discussed that the factors influencing software maintainability can be organized into hierarchical structures. The author defined measurable attributes in the form of metrics for each factor which helps measure software characteristics, and those metrics can be combined into a single index of software maintainability. Fluri et al.[11] examined whether the source code and associated comments are changed together along the multiple versions. They investigated three open source systems, such as *ArgoUML, Azureus*, and *JDT Core*, and found that 97% of the comment changes are done in the same revision as the associated source code changes. Another work[12] published in 2007 which proposed a two-dimensional maintainability model that explicitly associates system properties with the activities carried out during maintenance. The author claimed that this approach transforms the quality model into a structured quality knowledge base that is usable in industrial environments. Storey et al. did an empirical study on task annotations embedding within a source code and how it plays a vital role in a developer's task management[13]. The paper described how task management is negotiated between formal issue tracking systems and manual annotations that programmers include within their source code. Ted et al.[14] performed a 3×2 experiment to compare the efforts of procedure format with those of comments on the readability of a PL/I program. The readability accuracy was checked by questioning students about the program after reading it. The result said that the program without comment was the least readable. Yu Hai et al.[15] classified source code comments into four classes - unqualified, qualified, good, and excellent. The aggregation of basic classification algorithms further improved the classification result. Another work published in [16] in which author proposed an automatic classification mechanism "CommentProbe" for quality evaluation of code comments of C codebases. We see that people worked on source code comments with different aspects[16, 17, 18, 19, 20, 21], but still, automatic quality evaluation of source code comments is an important area and demands more research.

The advent of large language models (LLMs) [22] necessitates a comparison between the quality assessment of code comments performed by established models, such as GPT-3.5 and LLaMA, and evaluations based on human interpretation. The IRSE track at FIRE 2024 [23, 24] extends the approach proposed in [16, 25, 26, 19] 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

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 random forests 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.

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.

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

Random forest algorithm is used to implement binary classification functionality.System uses comments as well as surrounding code snippets as input.Using pre-trained universal sentence encoder we create embeddings of code and comments connected to it.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.

4.1. Random Forest

Random Forest (RF) algorithm is employed for binary comment classification in our study, leveraging an ensemble of decision trees to improve the model's predictive accuracy and control overfitting. The basic premise of Random Forest is to generate numerous decision trees during training, and output the class that is the mode of the classes output by individual trees during the prediction phase.

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, while n is the number of trees in the forest. A conventional threshold of 0.5 is employed for binary classification, however, this threshold can be adjusted to prioritize the *useful* comment class, analogous to threshold adjustments in random forests.

Random Forest effectively handles multi-dimensional feature spaces and does not require feature scaling. It deals with missing values by selecting the split that minimizes impurity among non-missing values, thereby imputing the missing ones based on the majority class or mean/mode value.

#	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

During the training phase, the out-of-bag (OOB) error, calculated from data not utilized in bootstrap samples, acts as an unbiased estimate of the generalization error and can be used for hyper-parameter tuning.

5. Results

We trained our random forest model on two datasets: the original dataset, containing 11,452 samples, and an augmented dataset, which included an additional 233 samples generated by GPT. Initially, we used only the original dataset and achieved the following scores.

After adding the GPT-generated samples to the original dataset, the results were as follows:

The minimal difference across the metrics suggests that the newly generated samples closely matched the original data, reinforcing the reliability of GPT-generated data for augmentation purposes.

	Accuracy	Precision	Recall	F1 Score
Original Dataset	81.0563	0.7902	0.8016	0.7949
Augmented Dataset	81.0013	0.7908	0.8014	0.7952

Table 2
Results for binary classification on both datasets

6. Conclusion

This paper explores a binary classification task focused on evaluating the usefulness of comments within C language source code. We utilized a random forest model as the primary classifier and ran two experiments: one with the original dataset and another using a combination of the original data and synthetic samples generated by GPT. The comparable results from both setups indicate that the synthetic data aligns well with the original, demonstrating its effectiveness in expanding the dataset for model training. These findings confirm the accuracy of the synthetic data and underscore its potential for enhancing data augmentation, making it valuable for various data 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. 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] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.

[11] B. Fluri, M. Wursch, H. C. Gall, Do code and comments co-evolve? on the relation between source code and comment changes, in: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE, 2007, pp. 70–79.
[12] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard, An activity-based quality model for maintainability, in: 2007 IEEE International Conference on Software Maintenance, IEEE, 2007, pp. 184–193.
[13] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, J. Singer, Todo or to bug, in: 2008 ACM/IEEE 30th International Conference on Software Engineering, IEEE, 2008, pp. 251–260.
[14] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.
[15] H. Yu, B. Li, P. Wang, D. Jia, Y. Wang, Source code comments quality assessment method based on aggregation of classification algorithms, Journal of Computer Applications 36 (2016) 3448.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).
[21] 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.
[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.