

On the Impact of Synthetic Data on Code Comment Usefulness Prediction

Vishesh Agarwal^{1,*},†

¹Microsoft Corporation, Redmond, WA 98052, USA

Abstract

In the domain of software development, the utility of code comments varies, necessitating methodologies capable of distinguishing their substantive value. This study delves into enhancing code comment usefulness classification by adopting a hybrid approach, combining manually tagged datasets with synthetic data augmentation. 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

Developers often need to fix bugs, develop new source code, or upgrade already deployed applications on a reduced time frame. This can lead to improper coding practices. As the software changes dynamically, the documentation, such as requirement specification, high-level design etc., becomes outdated and incomplete, and the knowledge transfer process or help from the earlier developers is often unobtainable. These types of situations demand a systematic quality-controlled development process. Automated Program comprehension is one such method of maintaining existing source code in a better way.[1].


Since the software design of a codebase is a moving target, the real source of truth are the traces of test execution, static analysis of the programs and, to a large extent, code comments. This paper focuses on code comments as information about the program design - both for developers, and for automated program comprehension. Code comments offer deep insights into the logic, decisions, and intentions behind the code, thereby aiding in code comprehension, maintenance, and debugging. However, not all comments are equally informative or useful, creating a compelling need to develop automated methods to classify the usefulness of code comments effectively.


* Forum for Information Retrieval Evaluation, December 15-18, 2023, India

† Corresponding author.

✉ visagarwal@microsoft.com (V. Agarwal)

ORCID 0000-0002-4551-748X (V. Agarwal)

 © 2023 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)

A common hurdle across studies for code comment usefulness is the scarcity of extensive, well-annotated datasets that encompass the diverse nature of comments in various programming contexts. This necessitates innovating strategies to enhance available data for improved model generalization on unseen, real-world comments. Recognizing this gap, we aim to integrate manual annotation with synthetic data augmentation. We employ GPT-3.5-turbo, a state-of-the-art language model, to label code comment samples scraped from open source code bases.

In this paper, we propose a binary classification task to understand the source code comments present in a program written in C language. We classify each comment into two classes - Useful and Not Useful. We start with a training data set of over 11000 manually-annotated samples. We use Random Forests to create a baseline for comment classification. Then, we augment the data with over 200 GPT-labelled samples to examine the improvement in performance. We observed that the model performance remained consistent, with an F1 score of 0.79 both for the baseline and model trained on augmented data.

By exploring the intricate interplay between manual annotation and synthetic data augmentation, this study aspires to contribute a novel perspective to the existing body of knowledge on code comment usefulness classification. It strives to offer an innovative solution to the prevailing challenges in the field, inspiring further exploration and development of robust, scalable models that can seamlessly adapt to the dynamic, evolving 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 advent of large language models [19], 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 2023 [20] extends the approach proposed in [17] to explore various vector space models [21] 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.

A dataset consisting of over 11000 code-comment pairs written in C language is used in our work. Each instance of data consists of comment text, a surrounding code snippet, and a label that specifies whether the comment is 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

We use random forests to implement the binary classification functionality. The system takes comments as well as surrounding code snippets as input. We create embeddings of each piece of code and the associated comment using a pre-trained Universal sentence encoder. 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) 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).

#	Comment	Code	Label
1	/*test 529*/	<pre> -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 </pre>	Not Useful
2	/*cr to cr,nul*/	<pre> -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; </pre>	Not Useful
3	/*convert minor status code (underlying routine error) to text*/	<pre> -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 </pre>	Useful

Table 1
Sample data instance

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:

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

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 conventionally used for binary classification, although this can be adjusted to favor the *useful* comment class, similar to the threshold adjustment in random forests.

Random Forest inherently handles multi-dimensional feature space and does not require the feature scaling. It handles missing values by choosing the split that minimizes the impurity among non-missing values, hence imputing the missing ones based on the majority class or mean/mode value.

During training, the out-of-bag (OOB) error, computed on the data not used in bootstrap samples, serves as an unbiased estimate of the generalization error and can be employed for hyper-parameter tuning.

5. Results

We train our random forests 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 and produces 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 indistinguishable from the original dataset, highlighting the validity of using GPT generated data for data augmentation.

6. Conclusion

This paper has addressed a binary classification problem in the domain of source code comment classification. The classification has been done based on the usefulness of the comment present within a source code written in C language. We have used random forests as our base classification method. We conducted two experiments, one with the original dataset and another with the original dataset plus the synthetic GPT generated data. The similar results in both cases show that the synthetic data falls in line with the original dataset, and how synthetic data creation can help in effectively increasing data volume required for training models. The synthetic data's correctness as compared to the original dataset is proven by the results shown above. Synthetic data generation can help a lot with data augmentation, finding its use in many pipelines.

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.