Assessing the Quality of Source Code Comments with Binary Classification Models

Krishna K[1,*]

[1]*Indian Institute of Technology, Kharagpur, West Bengal, Kol-721302*

Abstract

This research paper presents a categorization framework designed to assess the quality of source code comments, aiming to distinguish between comments that aid developers in understanding code and those that lack utility. To achieve this, comments are classified into two main categories: *useful* and *not useful*. The classification process begins with the creation of a seed dataset, which contains a curated set of labeled comments that serve as the foundation for training. Three distinct machine learning models are initially trained on this seed dataset: Multinomial Naive Bayes, Logistic Regression, and Support Vector Machine (SVM). These models provide a variety of perspectives on the classification task. In terms of performance, the Multinomial Naive Bayes model achieves an accuracy of 82.89%, the Logistic Regression model reaches an accuracy of 83.9%, and the SVM model attains an accuracy of 50.7%. Following the initial training, the dataset is expanded with new data generated by a large language model (LLM), which synthesizes additional labeled comments and thus increases the size and diversity of the training dataset, offering the potential for enhanced model generalization. However, including LLM-generated data introduces some degree of label noise, resulting in a marginal decrease in overall accuracy.

Keywords

Logistic Regression, Support vector machine, Comment classification, Qualitative analysis

1. Introduction

Modern technology now relies heavily on software, which powers the applications embedded in devices and appliances that streamline daily tasks and improve quality of life. For example, GPS software enables seamless navigation, while smart home applications allow users to control lighting, temperature, and security remotely. As software continues to advance, frequent updates and the introduction of new features result in a substantial increase in source code volume, making its maintenance a critical phase in the Software Development Life Cycle (SDLC). This maintenance phase poses numerous challenges for developers, who must manage large and often complex codebases. The frequently encounter out-of-date or incomplete documentations may lack insights into the original design choices and coding conventions followed by prior developers. These issues can complicate the process of understanding the code's structure, logic, and purpose, leading to slower development cycles and potential introduction of bugs.

A systematic approach is essential for addressing these challenges effectively. When new developers are tasked with adding or modifying functionality, they typically have access to essential resources such as the source code, sample test cases, requirement documentation, and a debugger. However, before they can make any meaningful changes, it is crucial for them to fully understand the existing codebase. This understanding involves grasping the code's architecture, execution patterns, design principles, and underlying logic. Achieving this level of comprehension often requires developers to run the program multiple times, using various test cases to observe how different parts of the code interact and produce outcomes. This process can be labor-intensive, repetitive, and sometimes overwhelming, particularly for large, complex codebases. To save time, developers may take shortcuts, such as examining only parts of the code or skipping tests. While this might expedite the process initially, it often leads to errors, inconsistencies, and incomplete understanding, ultimately compromising software quality. These shortcuts can accumulate technical debt, complicating future maintenance and introducing

Forum for Information Retrieval Evaluation, December 12-15, 2024, India
*Corresponding author.

krishnakarthik1999@gmail.com (K. K)
0009-0006-3456-2409 (K. K)

potential bugs. A structured, quality-focused development process is essential to address these issues. Emphasizing careful code analysis, systematic debugging, and error reduction helps preserve software integrity and usability. Program comprehension plays a central role here, as it enables thorough understanding of existing code, which is critical for efficient reuse, thorough inspection, and sustainable maintenance—cornerstones of high-quality software in the engineering domain.

Adding comments to code is essential for improving code understanding. However, readability may be jeopardized if different developers interpret and comment on the same codebase in different ways. Therefore, for better readability, a consistent method for creating code and comments is necessary. However, this strategy might still not work well for previously written code, where a natural grasp of comments can greatly improve readability. Researchers have been looking into new tools in recent years to help programmers especially new developers to understand existing code more effectively.

This paper presents a classification framework that processes a dataset of code-comment pairs in C language. The work proceeds in three phases: training a classification model on a seed dataset, adding more data to the dataset using a large language model, and retraining the model with the larger dataset. In the first stage, the model divides code-comment pairs into two groups: *Useful* or *Not Useful*. For this categorization, we employ multinomial naive Bayes, logistic regression, and support vector machines (SVM). Five-fold cross-validation is used to validate the framework, which is tested on a test dataset of 1,001 samples after being trained on a dataset of 9,000 samples. The SVM model utilizes a linear kernel, while logistic regression incorporates L2 regularization. During the second stage, an additional set of code-comment pairs is collected from sources such as GitHub. The categorization of these pairs as either *useful* or *not useful* is performed using ChatGPT-4. The augmented dataset is integrated with the initial seed dataset for the purpose of retraining the classification models. A minor reduction in F1 scores and accuracy levels across the models has been noted, likely attributable to noise introduced by the generated data.

The rest of the paper is structured as follows. Section 2 provides an overview of the foundational research conducted in the area of comment classification. Section 3 provides a discussion of the details pertaining to existing methods. The proposed method is discussed in section 4. The results are presented in section 5 and finally the conclusion of the paper is presented in Section 6.

2. Related Work

Software metadata [1]is essential for maintaining code and enhancing its future comprehension. Various tools have been created to help extract insights from software metadata, encompassing runtime traces and structural code attributes. [2, 3, 4, 5, 6, 7, 8, 9, 10].

In the field of mining code comments and evaluating their quality, numerous researchers have carried out studies. Steidl et al. [11] applies techniques such as Levenshtein distance and the length of comments to evaluate the similarity of words in code-comment pairs, efficiently removing trivial and uninformative remarks. Rahman et al. [12] emphasize distinguishing valuable code review comments from non-valuable ones within review portals. This differentiation is informed by insights obtained from a survey conducted among Microsoft developers, which helps in identifying pertinent attributes. [13]. Majumdar et al. [14, 15, 16, 17] has established a framework for analyzing comments grounded in fundamental concepts important for code comprehension. Their methodology encompasses the creation of features that link textual content with code and the use of a knowledge graph to interpret the semantic meaning of the information found in comments. These techniques harness both semantic and structural elements to address the challenge of predicting which comments are valuable versus those that are not, ultimately facilitating the process of streamlining codebases.

With the rise of large language models like GPT-3.5 and Llama [18], evaluating the quality of code comments and contrasting them with human interpretation has become essential. The IRSE track at FIRE 2023 [19] builds on the methodology presented in earlier research [14]. It delves into investigating various vector space models [20] and features for binary classification and assessment of comments, particularly regarding their significance in understanding code. Additionally, this track performs a comparative study of the prediction model's effectiveness when incorporating GPT-generated labels for the quality of code and comments obtained from open-source software.

3. Task and Dataset Description

The binary classification framework is implemented through three main stages: first, creating the framework using an initial seed dataset; second, increasing the seed dataset's size by leveraging a large language model; and third, training the framework with this expanded dataset. Using the trained model, code-comment pairs are categorized into *useful* and *not useful* classes. This approach takes a description of the comment alongside its related code lines as input and assigns a label such as *useful* and *not useful* to each code-comment pair. Classical machine learning models, including logistic regression, naive Bayes, and SVM, were employed in developing this classification system.

- *Useful* - The provided comment is well-suited to the corresponding source code.
- *Not Useful* - The provided comment is not well-suited to the corresponding source code.

The initial dataset consists of 9,000 code-comment pairs written in the C language. Each entry includes the comment text, the surrounding code snippet, and a label indicating whether it is useful. This dataset was sourced from GitHub and annotated by a team of 15 individuals. An example entry can be seen in Table 1. Additionally, a separate collection of code-comment pairs was gathered from various online sources and then combined with the original dataset. These pairs were classified into the two specified categories using a large language model, and this newly created dataset was subsequently merged with the seed dataset.

The classification model is subsequently retrained using this expanded dataset to assess the impact of augmentation. Various factors are examined, such as noise inclusion and dataset distribution, which influence the changes in accuracy observed during the training of the classification framework with the augmented dataset.

4. Working Principle

Three machine learning models—logistic regression, support vector machine, and multinomial naïve Bayes—were used to create a binary classification system. Source code and associated comments are both inputs to the system. Deep learning frameworks were not used because of the task requirements. First, an English word lemmatizer is used to tokenize the comments. TF-IDF is then used to vectorize these tokens. Class labels and the resulting TF-IDF matrix are used as the input features for the classification models. The core seed dataset is used to train these models, and a test dataset is used to assess them. Using five-fold cross-validation, any variation in the training data was controlled for. Each machine learning model employed is briefly explained in the ensuing subsections.

4.1. Logistic Regression

For the binary comment classification task, we employ logistic regression, which uses a logistic function to constrain the output values between 0 and 1. The logistic function is defined as follows:

$$Z = Ax + B \tag{1}$$

$$logistic(Z) = \frac{1}{1 + exp(-Z)} \tag{2}$$

Equation 1 denotes the linear regression formula, where the output (Z) is input into the logistic function outlined in Equation 2. The binary class is determined from the probability value produced by the logistic function, relying on a predetermined acceptance threshold. This threshold is set to 0.65, prioritizing the *useful* comment class. Each training example yields a three-dimensional feature set that is provided to the regression function. The cross-entropy loss function is utilized during training for optimizing hyperparameters.

Table 1
Sample data instances from the seed dataset

#	Comment	Code	Label
1	/*enable verbose*/	-1. test_setopt(curl, CURLOPT_UPLOAD, 1L); /*enable verbose*/ 1. test_setopt(curl, CURLOPT_VERBOSE, 1L);	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	/*See if this is
	an UIDVALIDITY response*/ to text*/
		-1. if(imapcode == ’*’) {
		1. char tmp[20];
		2. if(sscanf(line + 2, "OK [UIDVALIDITY
		%19[0123456789]]" , tmp) == 1) {
		3. Curl_safefree(imapc->mailbox_uidvalidity);
		4. imapc->mailbox_uidvalidity = strdup(tmp);
		}
		}
		else if(imapcode == IMAP_RESP_OK) {
			Useful

4.2. Support Vector Machine

In the following stage, we implement a support vector machine (SVM) model to tackle the binary classification task. The classification is determined by the output of the linear function (Equation 1). If this output is greater than 1, the instance is assigned to one class, while if it falls below -1, it is categorized into another class. The SVM model is trained using the hinge loss function, as demonstrated below.

$$\begin{aligned} H(x,y,Z) &= 0, & ify * Z \geq 1 \\ &= 1 - y * Z, & otherwise \end{aligned} \tag{3}$$

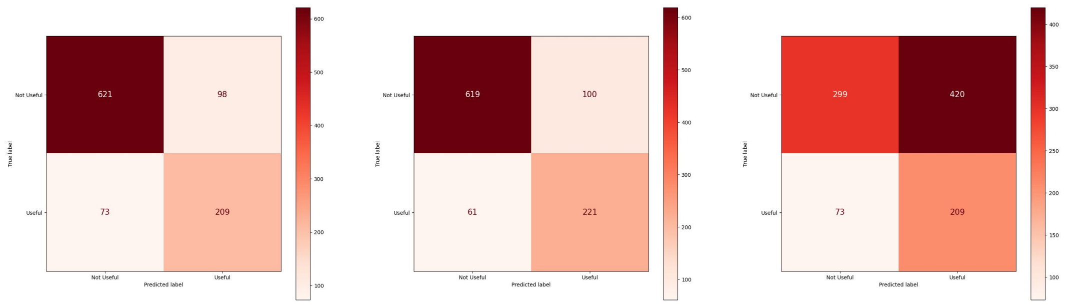
It is observed from the loss function that the cost becomes 0 when the predicted and actual values share the same sign. Conversely, a loss value is incurred if the predicted and actual values have different signs. The hinge loss function is utilized for tuning the hyperparameters of the SVM model.

4.3. Multinomial Naive Bayes

The Multinomial Naïve Bayes model is employed in this task primarily for text classification. This model applies Bayes’ theorem, which is described as follows:

$$P(y|X) = \frac{P(X|y).P(y)}{P(X)} \tag{4}$$

where,
$P(y|X)$ is the posterior probability of class y given features X.
$P(X|y)$ is the likelihood, indicating the probability of encountering features X given class y.
$P(y)$ is the prior probability of class y.



(a) Logistic regression (b) Support vector machine (c) Multinomial naive Bayes

Figure 1: Confusion matrix for classification models related to seed data

$P(X)$ is the probability of observing features X, which serves as a normalization constant.

Given a specific class, Multinomial Naive Bayes assumes that each feature is conditionally independent of the others.

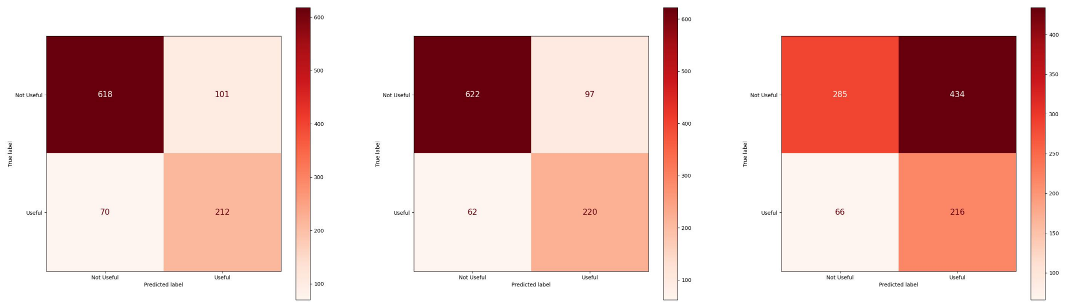
5. Results

The task is carried out on a system that has an Intel i5 processor and 32 GB of RAM. As mentioned earlier, there are three steps in the complete procedure. The seed dataset is first divided into two sections: 90% of the data is used for training, and 10% is used for validation. The three machine learning models—multinomial naive Bayes, support vector machines, and logistic regression—are trained using the training dataset. Of the 1,001 events in the test dataset, 719 have been classified as *not useful* and 282 as *useful*. Using this test dataset, the three models are assessed; the logistic regression, support vector machine, and multinomial naive Bayes models have overall accuracies of 82.89%, 83.9%, and 50.7%, respectively. Figure 1 shows the corresponding confusion matrices. Overall accuracy was shown to decrease as a result of the naive Bayes algorithm's difficulty in correctly predicting the occurrences that are not beneficial.

A new dataset is created using a large language model, containing 311 *useful* samples and 21 *not useful* samples. This generated data is then combined with the seed dataset. The resulting dataset is subsequently divided into two sections: training and validation datasets. These newly formed training and validation datasets are utilized to train the same classification models again. The models that have been retrained are then assessed using the original test data. Overall accuracies of 82.91%, 84.1%, and 50% are achieved by the three models, respectively. The individual confusion matrices for each of the models are also shown in Figure 2. The evaluation outcomes for all three models are summarized in Table 2. It is clear that the models trained on the augmented dataset experience a slight drop in accuracy compared to the earlier results obtained from the seed dataset. This decline may be due to the introduction of noise in the seed data originating from the large language models. Such noise primarily arises from the limitations of the large language model, specifically ChatGPT-4 in this case, leading to a reduction in overall accuracy. Nonetheless, we can argue that the augmented dataset remains well-balanced for training machine learning models and yields accuracy levels comparable to those of the initial seed dataset.

6. Conclusion

This paper presents a framework for the classification of source code comments based on their usefulness. Three machine learning models—logistic regression, support vector machine, and multinomial naive



(a) Logistic regression (b) Support vector machine (c) Multinomial naive Bayes

Figure 2: Confusion matrix for classification models related to seed + LLM generated data

Table 2
Experimental results of three classification model on the test dataset

Dataset	Models	Precision	Recall	F1-score	Accuracy (%)
Seed data	Logistic regression	0.8024	0.7878	0.7943	82.92
	Support vector machine	0.8223	0.7994	0.809	83.92
	Naive Bayes	0.5785	0.568	0.5035	50.75
Seed data + LLM generated data	Logistic regression	0.8057	0.7879	0.7955	82.92
	Support vector machine	0.8226	0.8017	0.8106	84.12
	Naive Bayes	0.5812	0.5721	0.4981	50.05

Bayes—are developed and trained utilizing the seed dataset. These classifiers categorize each comment into two distinct groups: *useful* and *not useful*. The achieved accuracies for these models are 82.89%, 83.9%, and 50.7%, respectively. Subsequently, the seed dataset is expanded with a newly created dataset obtained from online sources. The associated labels for this new dataset are generated using the ChatGPT large language model (LLM). This augmented dataset is then employed to retrain all models. It has been noted that the introduction of noise and biases from the LLM-generated dataset leads to a reduction in accuracy across all three models.

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] 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.

[2] 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.

[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] 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).

[10] 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.

[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] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[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.

[20] 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.