Improving Code Readability: A Machine Learning Approach to Classifying Valuable Source Code Comments

Subhash Agrawal[1,*]

[1]Indian Institute of Technology, Kharagpur

Abstract

This paper introduces a framework designed to classify source code comments based on their practical value, helping new developers better understand the code. We applied three machine learning models—logistic regression, support vector machine, and multinomial naive Bayes—to an initial training dataset to sort comments into two groups: *useful* and *not useful*. The initial training results showed accuracies of 83%, 84.12%, and 51% for each model, respectively. Next, we expanded the dataset with additional code-comment pairs from various online sources, using the ChatGPT large language model (LLM) to label the new data. After retraining the models with this combined dataset, we found that the models' accuracy decreased slightly, suggesting that the new data introduced some level of noise and bias.

Keywords

Machine Learning, Comment classification, Qualitative analysis, Logistic Regression, Support vector machine, Multinomial naive Bayes, Large language model

1. Introduction

In today's technology-driven landscape, programming plays an integral role across industries—from healthcare and communications to transportation. As the demand for new applications grows, so does the volume of source code. Frequent updates to existing code bases only add to this volume, making it increasingly challenging to manage and debug substantial code repositories within tight timelines. This pressure often leads to poor code documentation, making it difficult to reuse older code effectively for future projects.

Without clear documentation, developers may need to repeatedly execute and analyze the code to understand its functionality, which is tedious and time-consuming. To speed up the process, developers might skip crucial steps, increasing the risk of errors and reducing the overall effectiveness of the new software. A common way to address these challenges is through code comments, which give insights into the original developer's thought process. However, variations in individual commenting styles can create inconsistency, making code difficult to interpret. For this reason, standard practices for writing comments are essential.

When dealing with existing source code that lacks clear comments or documentation, a different solution is needed. Developing tools that enhance the readability of legacy code has become a research focus, as such tools can save developers valuable time and help in building more robust applications.

In this paper, we present a framework to classify code comments based on their usefulness, aiming to categorize comments as either *useful* or *not useful* in helping developers understand the code. To build this framework, we used a dataset of approximately 10,000 code-comment pairs written in C, training three machine learning models—support vector machine, logistic regression, and multinomial naive Bayes—on this initial dataset. Additionally, we expanded our dataset by gathering more code-comment pairs from online sources, again in C language, and used ChatGPT-4 to label the new data as *useful* or *not useful*. With this combined dataset, we re-trained each model and compared the new accuracy and F1 scores to the initial results. This comparison helped us assess the impact of the additional data, highlighting both the benefits and potential drawbacks of using large language model-generated labels.

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

*Corresponding author.

subhashagrawal2701@gmail.com (S. Agrawal)

2. Related Work

New programmers often rely on existing comments to comprehend code flow. However, not all comments contribute effectively to program comprehension, necessitating a relevancy assessment of source code comments prior to their use. Numerous researchers have focused on the automatic classification of source code comments in terms of quality evaluation. For instance, Omal et al. [1] noted that factors influencing software maintainability can be organized into hierarchical structures. The authors defined measurable attributes in the form of metrics for each factor, enabling the assessment of software characteristics, which can then be consolidated into a single index of software maintainability. Fluri et al.[2] 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[3] 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[4]. 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.[5] 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.[6] 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 [7] 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[7, 8, 9, 10, 11, 12], but still, automatic quality evaluation of source code comments is an important area and demands more research.

With the advent of large language models [13], 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 2024 [14, 15] builds upon the methodologies proposed in [7, 16, 17, 10] to investigate various vector space models [18] and features for binary classification and evaluation of comments in relation to code comprehension. This track also assesses the performance of the predictive model by incorporating GPT-generated labels for the quality of code and comment snippets extracted from open-source software.

3. Task and Dataset Description

This section outlines the focus of our research, which is organized into three main components: designing a classification framework with a seed dataset, enhancing this dataset using a large language model, and retraining the classification framework with the newly augmented data. Our objective is to establish a binary classification system that categorizes pairs of source code and comments as either - *useful* and *not useful.* The input for this process includes a comment and its corresponding lines of code, while the output will be a label indicating its relevance either *useful* or *not useful.* This framework is intended to assist developers in better understanding the associated code. We employ traditional machine learning algorithms, including Naive Bayes, logistic regression, and support vector machines (SVM), to create the classification system.

The two classes of source code comments can be described as follows:

- *Useful* - The comment is relevant and provides insight into the corresponding source code.
- *Not Useful* - The comment does not relate meaningfully to the source code it references.

#	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 Instance

For our study, we utilize a seed dataset comprising around than 9,500 pairs of code and comments, all written in C. Each entry contains the comment text, an accompanying code snippet, and a label indicating whether the comment is useful or not. This dataset was sourced from GitHub and meticulously annotated by a team of 11 individuals. A sample data is illustrated in table 1. To enhance our dataset, we collected additional code-comment pairs from various online platforms. These new pairs were classified into the aforementioned categories using a large language model. We then merged this new subset with our original seed dataset.

The augmented dataset is utilized to retrain the classification model, allowing us to evaluate the impact of the augmentation process. We also investigate the introduction of noise, the distribution of the dataset, and several other factors that may influence changes in accuracy during training with the enriched dataset.

4. Working Principle

We aim to train three different machine learning models—Logistic Regression, Support Vector Machine (SVM), and Multinomial Naïve Bayes—to achieve the binary classification task. It's important to note that we are not utilizing any deep learning frameworks due to the specific constraints of our project. The system processes both comments and their corresponding code segments as input. Before converting the comments into a numerical vector space, we preprocess the text by lemmatizing and tokenizing it using an English word tokenizer. To transform each English keyword into numerical representations, we employ a TF-IDF vectorizer. This approach produces a TF-IDF matrix encompassing all keywords found within the comment dataset, which serves as our feature set for the classification framework. We split the entire dataset into two distinct subsets: a training set comprising 90% of the data and a test set containing the remaining 10%. Features are extracted from the training data using the aforementioned methods. The TF-IDF matrix along with corresponding class levels are then fed into machine learning models for training. These models will subsequently classify each code-comment pair into one of the two categories. In the following subsections, we will briefly discuss overview of the three classification models employed in our study.

4.1. Logistic Regression

For our binary comment classification task, we employ Logistic Regression. To ensure that the regression output remains within the range of 0 to 1, we utilize a logistic function defined as follows:

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

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

In this process, the output from the linear regression equation (see the first equation) is fed into the logistic function (see the second equation). The logistic function generates a probability value, which we use to determine the binary class based on a specified acceptance threshold. We set this threshold at 0.6 to favor classifying comments as "useful."

From each training example, we extract a three-dimensional input feature that is then processed through the regression function. During training, we apply the Cross-Entropy Loss function for hyperparameter tuning in the Logistic Regression model.

4.2. Support Vector Machine

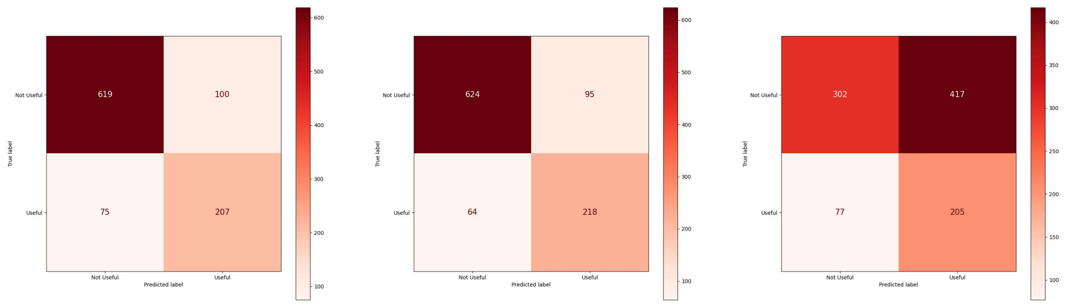
Next, we implement a Support Vector Machine (SVM) model for our binary classification task. The classification process begins with the output of a linear function. If this output exceeds 1, we categorize it as one class; conversely, if the output falls below -1, we assign it to the other class. To train the SVM model, we utilize the Hinge Loss function, defined as follows:

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

This loss function indicates that there is no cost when the predicted and actual values align (i.e., they have the same sign). However, when they differ in sign, we compute a loss value. The Hinge Loss function plays a crucial role in fine-tuning the hyperparameters of the Support Vector Machine model.

4.3. Multinomial Naive Bayes

We also employ the Multinomial Naïve Bayes model, which is particularly effective for text classification tasks, to perform our binary classification. This model uses the Bayes' theorem mentioned as below:



(a) Logistic regression (b) Enter Caption (c) Multinomial naive Bayes

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

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

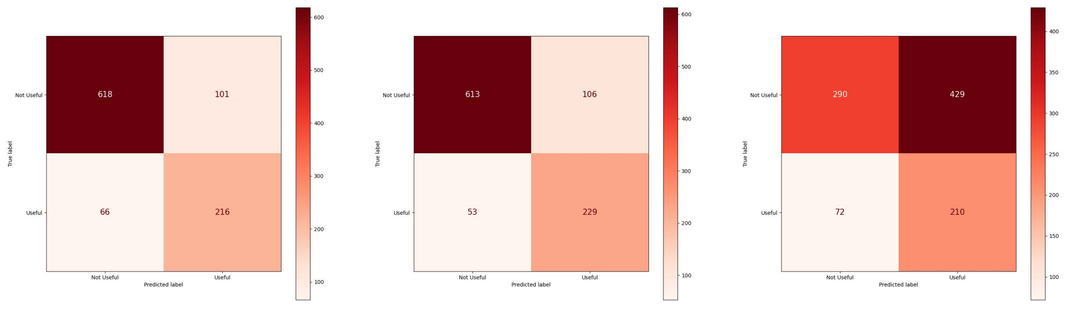
where,
$P(y|X)$ is the posterior probability of class y given features X.
$P(X|y)$ is the likelihood, which indicates the probability of observing the features X under class y.
$P(y)$ is the prior probability of class y.
$P(X)$ The overall probability of observing the features X, serving as a normalization factor.

The Multinomial Naïve Bayes model operates under the assumption that each feature is conditionally independent of others when conditioned on a particular class.

5. Results

The implementation of our task was carried out on a system equipped with an Intel i7 processor and 32 GB of RAM. As outlined in Section 3, the entire process comprises three stages. We began by splitting the seed dataset into two segments: 90% for training and 10% for validation. We then trained three classification models—logistic regression, support vector machine, and multinomial naive Bayes—using the same training dataset. For testing, we utilized a dataset containing 1,001 instances, which were labeled as 719 instances of *not useful* and 282 instances of *useful*. Upon evaluation, the models achieved overall accuracies of 83%, 84.12%, and 51%, respectively. The confusion matrices for these models can be found in Figure1. Notably, the naive Bayes algorithm struggled to accurately classify *not useful* instances, leading to a lower accuracy compared to the other two models.

To enhance our dataset, we incorporated additional data generated by a large language model, which included 309 *useful* samples and 25 *not useful* samples. The complete dataset was then split again into training and validation subsets. We retrained the same classification models with this newly formed dataset and tested them using the original test data. The updated models yielded overall accuracies of 83.92%, 83.92%, and 50%, respectively. The individual confusion matrices for these models are shown in Figure 2. It is evident that the models trained on the augmented dataset experienced a slight decline in accuracy compared to those trained on the seed data. This indicates that the introduction of data from large language models contributed some noise to the original seed data, negatively impacting the overall accuracy of all three models. This noise stemmed from the limitations of the large language model used, specifically ChatGPT-4 in this instance. Nonetheless, we can assert that the augmented dataset remains well-balanced for training machine learning models.



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

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

6. Conclusion

In this study, we presented the development of three binary classification models aimed at categorizing code and comment pairs. These models were trained using a seed dataset containing two distinct classes, where all source code was written in C and the comments were in English. We employed tokenization and TF-IDF vectorization to transform the comments into numerical vectors, which served as the features for our classification frameworks.

Additionally, we enhanced the initial seed dataset by incorporating new data sourced from online platforms. This newly collected data was classified into the same two categories using a large language model, specifically ChatGPT. Upon retraining the classification models with the augmented dataset, we noted a slight decline in accuracy compared to the models trained solely on the seed data. This suggests that the inclusion of noise from the large language model-generated dataset may have impacted the performance.

Furthermore, we conducted a comparative analysis of the results from both datasets. Our findings indicate that the biases and noise present in the augmented dataset contributed to the reduced accuracy of our classification 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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).

[18] 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.