Automated Classification of C Code Comment Quality with SVM and Naïve Bayes

Kumar Shresth[1,*]

[1]*Indian Institute of Technology, Kharagpur (IIT-KGP), West Bengal-721302, India*

Abstract

Code comments are essential for effective software development, yet their quality often suffers, particularly among inexperienced programmers, leading to a high volume of unhelpful annotations. This study investigates the effectiveness of two machine learning models—the Support Vector Machine (SVM) and the Naïve Bayes Classifier—for automatically classifying the utility of comments in C source code. The results of these experiments provide a foundational benchmark for future research in this area. This work demonstrates that these models can serve as a starting point for developing more advanced machine learning solutions to improve the accuracy of comment quality assessment.

Keywords

Machine Learning, Natural Language Processing, SVM, Naïve Bayes Classifier

1. Introduction

Comments are a vital part of the software development lifecycle, intended to improve the readability and maintainability of code. However, not all comments contribute positively to this goal. With the increasing prevalence of programming, many novice developers do not learn to write effective comments, leading to a decrease in both the quantity and quality of documentation within codebases. This often results in a significant number of useless comments, which can be a source of frustration and wasted time for developers who must read through them.

While recent advancements in deep learning have produced models capable of automatically generating code comments, thereby increasing their quantity, the issue of comment quality has been largely overlooked in academic literature. To address this gap, current research is exploring the use of machine learning models to identify and classify the utility of code comments.

This paper details our participation in the Information Retrieval in Software Engineering (IRSE) shared task at the Forum for Information Retrieval Evaluation (FIRE) 2025. Our work seeks to answer two primary questions:

- What is the minimum complexity required for a Machine Learning model to effectively differentiate between useful and non-useful comments?
- How do established, general-purpose models like SVM and the Naïve Bayes Classifier perform on this specialized task?

The primary aim of this research is to establish that foundational models such as SVM and the Naïve Bayes Classifier can serve as effective baselines for this classification problem. These initial results can then inform the development of more complex and tailored models, while also considering the risk of overfitting.

Furthermore, this study also investigates the potential of using data generated by large language models, specifically GPT-5.0, to augment the training dataset. This is a critical area of inquiry as it explores the use of artificial intelligence to improve the evaluation of comment quality. By assessing the impact of this data augmentation, we aim to understand the benefits and challenges of integrating

Forum for Information Retrieval Evaluation, December 17-20, 2025, India

*Corresponding author.

kmrshresth@gmail.com (K. Shresth)
https://cse.iitkgp.ac.in/~kumar.shresth/ (K. Shresth)

AI-generated content into the machine learning pipeline for this task. This exploration of the synergy between human-written and AI-generated data forms a significant component of our research.

2. Related Work

The importance of software metadata for code maintenance and comprehension is well-established. A variety of tools have been developed to extract knowledge from different forms of software metadata, including code structure and runtime traces [1, 2, 3, 4, 5, 6].

The specific area of mining code comments and assessing their quality has been investigated by numerous authors. For instance, Steidl et al. [7] proposed methods to filter out irrelevant and uninformative comments by analyzing word similarity in code-comment pairs using techniques like Levenshtein distance and comment length. In a different approach, Rahman et al. [8] focused on distinguishing between helpful and unhelpful code review comments by leveraging features identified in a survey of developers at Microsoft [9].

More recently, Majumdar et al. [10, 11, 12, 13, 14, 15, 16] have introduced a framework for evaluating comments based on concepts central to code comprehension. Their approach utilizes a knowledge network to semantically assess the information within comments, developing features based on the correlation between the text and the code. Ultimately, these methodologies contribute to cleaning codebases by using both semantic and structural information to classify comments based on their utility.

The advent of large language models such as GPT-5.0 and Llama has introduced a new dimension to this field, making it crucial to evaluate the quality of automatically generated code comments against human standards. The IRSE track at FIRE 2023 [17, 18, 19, 20, 21, 22, 23] expanded upon the methodology from previous work [10] to address this. This track explored various vector space models [24] and features for the binary classification of comments, particularly concerning their role in code comprehension. A key aspect of this track was comparing the prediction model's performance with labels generated by GPT for both code and comment quality from open-source projects.

3. Task and Dataset Description

This section outlines the specifics of the experimental task and the dataset provided for it. The core objective is as follows:

To develop a binary classification model for assessing the quality of code comments, with the model's accuracy being potentially enhanced through data augmentation using generated code and comment pairs.

The dataset provided for this task was divided into two main parts:

- A **training dataset** containing 8048 entries.
- A **testing dataset** containing 1000 entries.

For the purpose of model development, the training dataset was shuffled and then partitioned, with 70% allocated for training the models and the remaining 30% reserved for cross-validation. Each comment in the dataset is categorized with one of two labels:

- **Useful**: Comments that contribute positively to code comprehension.
- **Not Useful**: Comments that do not aid in understanding the code.

Table 1 provides illustrative examples from the dataset for each label.

4. Dataset Augmentation

To enhance the training dataset, we employed a data augmentation strategy utilizing the large language model, GPT-5.0-turbo. The primary objective of this augmentation was to increase both the size and

Table 1
Examples of Labeled Comments from the Dataset

Label	Example
Useful	/*not interested in the downloaded bytes, return the size*/
Useful	/*Fill in the file upload part*/
Not Useful	/*The following works both in 1.5.4 and earlier versions:*/
Not Useful	/*lock_time*/

diversity of the dataset. By leveraging the natural language generation capabilities of GPT-5.0, we produced additional comment data.

This process was designed to introduce a broader spectrum of writing styles, formats, and topics into the training set. The inclusion of this GPT-generated data allowed us to evaluate its potential for improving the performance of machine learning models on the task of comment quality classification. This approach facilitated an investigation into how AI-generated content can supplement human-written data, with the goal of creating a more comprehensive and robust dataset to ultimately enhance model performance.

5. System Description

This section details the methodology used to build and evaluate the comment classification models, including the text preprocessing pipeline, feature extraction techniques, and the machine learning models themselves.

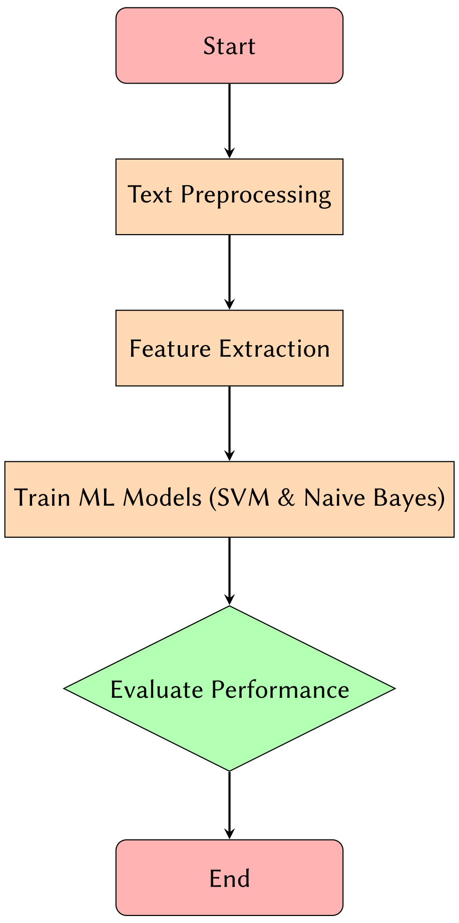

Figure 1: The workflow of the comment classification system.

5.1. Text Preprocessing

A multi-step text preprocessing pipeline was applied to both the training and testing datasets to clean and normalize the comment text. The steps were as follows:

1. **Initial Cleaning:** All stop words, punctuation, numerical digits, and hyperlinks were removed from the comments.
2. **Part-of-Speech (POS) Filtering:** Words that were not identified as nouns, verbs, adverbs, or adjectives were eliminated to retain the most semantically meaningful terms.
3. **Lemmatization:** The NLTK WordNet library was used to perform lemmatization, which is the process of reducing the different forms of a word to a single root form.

5.2. Feature Extraction

To convert the preprocessed text into a numerical format suitable for machine learning, the TfidfVectorizer from the SciKit-Learn library was employed. This technique creates a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features, which reflect the importance of a word in a document relative to the entire corpus. The Keras library's tokenizer was also utilized in this process.

5.3. Machine Learning Models

Two distinct machine learning models were implemented for this classification task: a Support Vector Machine (SVM) and a Naïve Bayes classifier. Both models were built using the SciKit-Learn library. The SVM model was configured with the following parameters:

- **C**: The regularization parameter was set to 1.
- **kernel**: A 'linear' kernel was used for the classification.

6. Findings

This section presents the performance of the SVM and Naïve Bayes models on the comment classification task. The results are presented in two parts: first without data augmentation, and then with the inclusion of GPT-5.0 generated data.

6.1. Performance Without Data Augmentation

On the original, un-augmented validation set, the Support Vector Machine (SVM) model achieved an accuracy of 77.27% with a corresponding Macro F1 score of 0.771. The Naive Bayes classifier, in comparison, yielded a 60.99% accuracy score and a Macro F1 score of 0.686. The detailed performance metrics, including precision and recall, are presented in Table 2.

Table 2
Classifier Performance on the Original Validation Set

Model	Macro F1 Score	Macro Precision	Macro Recall	Accuracy (%)
SVM	0.771	0.785	0.758	77.27
Naïve Bayes	0.686	0.736	0.642	60.99

6.2. Performance With Data Augmentation

After augmenting the training data with comments generated by GPT-5, both models showed an improvement in performance. The SVM model's accuracy increased to 77.65%, with a corresponding Macro F1 score of 0.778. The Naive Bayes classifier demonstrated a more significant improvement, reaching an accuracy of 64.03% and a Macro F1 score of 0.730. A comprehensive breakdown of these results is provided in Table 3.

Table 3
Classifier Performance on the Augmented Validation Set

Model	Macro F1 Score	Macro Precision	Macro Recall	Accuracy (%)
SVM	0.778	0.791	0.765	77.65
Naïve Bayes	0.730	0.736	0.652	64.03

7. Conclusion

This study successfully employed two fundamental machine learning models, the Support Vector Machine and the Naïve Bayes classifier, to address the task of classifying C code comments. The results obtained from the SVM classifier are promising and indicate that there is significant potential for further improvement. These findings serve as a solid baseline and justify the exploration of more sophisticated models that can better capture the nuances of the problem domain and achieve higher accuracy. It is worth noting that prior research by Majumdar et al. [25] has already demonstrated the effectiveness of neural networks for this task, and we anticipate that future work will continue to build upon these successes.

Declaration on Generative AI

During the preparation of this manuscript, the author utilized ChatGPT for assistance with grammar and spelling checks. Following the use of this tool, the author thoroughly reviewed and edited the content to ensure its accuracy and originality, and takes full responsibility for the final content of this publication.

Acknowledgments

The author would like to extend sincere thanks to the organizers of the IRSE track at FIRE for providing this excellent opportunity to engage with such a compelling research problem. Their continuous technical support throughout the duration of the project was invaluable.

References

[1] 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.

[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] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] A. Mitra, S. Majumdar, A. Mukhopadhyay, P. P. Das, P. D. Clough, P. P. Chakrabarti, Operationalizing large language models with design-aware contexts for code comment generation, arXiv preprint arXiv:2510.22338 (2025).

[16] A. Deshpande, A. Maji, D. Mondol, P. P. Das, P. D. Clough, S. Majumdar, The code–llm handshake: Smarter maintenance through ai, in: Proceedings of the 17th annual meeting of the Forum for Information Retrieval Evaluation, 2025, pp. 9–12.

[17] 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.

[18] S. Majumdar, A. Deshpande, P. P. Das, P. P. Chakrabarti, Comprehending c codes with llms: Effective comment generation through retrieval and reasoning, Pattern Recognition Letters (2025).

[19] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D. Clough, et al., Overview of the "information retrieval in software engineering"(irse) track at forum for information retrieval 2024, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024, pp. 18–21.

[20] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access 11 (2023) 73599–73612.

[21] 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.

[22] 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.

[23] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Tool assisted agile approach for legacy application migration, International Journal of System Assurance Engineering and Management (2025) 1–16.

[24] 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.

[25] 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. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2463. doi:`https://doi.org/10.1002/smr.2463`. `arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2463`.