Assessing the Utility of C Comments with SVM and Naïve Bayes Classifier

Aritra Mitra[1,*]

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

Abstract

Comments are very useful to the flow of code development. With the increasing use of code in commonplace life, commenting the codes becomes a hassle for rookie coders, and often they do not even think commenting as a part of the development process. This in general causes the quality of comments to degrade, and a considerable amount of useless comments are found in such codes. In these experiments, the usefulness of C comments are evaluated using Support Vector Machine (SVM) and Naïve Bayes Classifier. The results of the experiments create a baseline for better results that can be found in the future through more research. Based on these findings, more complex and intricate machine learning models can be created that can improve the accuracy achieved in performing said task.

Keywords

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

1. Introduction

Comments play an essential role in code development [1], consuming a significant amount of time to enhance code readability [2]. However, not all comments contribute to this objective. As coding becomes more commonplace, novice programmers tend to overlook the art of commenting, leading to a deterioration in both the quality and quantity of comments [3]. Many comments turn out to be unproductive, and sifting through lengthy comments only to discover their futility can be frustrating and time-consuming.

Various deep learning-based automatic commenting models can boost the quantity of comments [4]. Regrettably, there has been insufficient research dedicated to addressing the issue of comment quality. Nevertheless, recent efforts are addressing these challenges by developing machine learning models capable of identifying and categorizing comments based on their usefulness.

The author has explored a range of Machine Learning (ML) models in pursuit of solutions to this problem. This paper aims to answer critical questions as part of the Information Retrieval in Software Engineering (IRSE) shared task at the Forum for Information Retrieval Evaluation (FIRE) 2022 [5], conducted under the team name FaultySegment:

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

*Corresponding author.

aritramitra2002@gmail.com (A. Mitra)

https://cse.iitkgp.ac.in/~aritra.mitra/ (A. Mitra)

- What level of complexity is necessary for a Machine Learning model to reliably distinguish useful comments from useless ones?
- How do well-known general-purpose models like SVM and Naïve Bayes Classifier perform in this context, even if they are not built for this particular scenario?

This paper aims to demonstrate that models such as SVM or Naïve Bayes Classifier can serve as effective starting points for tackling this problem. More complex models can be built upon these foundations, while also considering the risk of overfitting.

In addition to the aforementioned objectives, this paper also addresses the crucial question of whether the dataset for the learning task can be augmented with data generated by large language models, such as GPT-3. This consideration plays a pivotal role in exploring the potential of leveraging artificial intelligence for improving comment quality assessment. By evaluating the impact of augmenting the dataset with AI-generated data, the study aims to shed light on the benefits and challenges of integrating advanced language models into the machine learning pipeline. This inquiry represents a significant aspect of the research, emphasizing the intersection of human-written and AI-generated content in the context of comment quality evaluation.

2. Related Work

Software metadata [6] plays a crucial role in the maintenance of code and its subsequent understanding. Numerous tools have been developed to assist in extracting knowledge from software metadata, which includes runtime traces and structural attributes of code [7, 8, 9, 10, 11, 12, 13, 14, 15].

In the realm of mining code comments and assessing their quality, several authors have conducted research. Steidl et al. [16] employ techniques such as Levenshtein distance and comment length to gauge the similarity of words in code-comment pairs, effectively filtering out trivial and non-informative comments. Rahman et al. [17] focus on distinguishing useful from non-useful code review comments within review portals, drawing insights from attributes identified in a survey conducted with Microsoft developers [18]. Majumdar et al. [19, 20, 21, 22] have introduced a framework for evaluating comments based on concepts crucial for code comprehension. Their approach involves the development of textual and code correlation features, utilizing a knowledge graph to semantically interpret the information within comments. These approaches employ both semantic and structural features to address the prediction problem of distinguishing useful from non-useful comments, ultimately contributing to the process of decluttering codebases

In light of the emergence of large language models, such as GPT-3.5 or llama [23], it becomes crucial to assess the quality of code comments and compare them to human interpretation. The IRSE track at FIRE 2023 [5] expands upon the approach presented in a prior work [19]. It delves into the exploration of various vector space models [24] and features for binary classification and evaluation of comments, specifically in the context of their role in comprehending code. Furthermore, this track conducts a comparative analysis of the prediction model's performance when GPT-generated labels for code and comment quality, extracted from open-source software, are included.

3. Task and Dataset Description

In this section, a description of the task at hand and the dataset provided are given. The task at IRSE, FIRE 2023 was as follows: *A binary code comment quality classification model needs to be augmented with generated code and comment pairs that can improve the accuracy of the model.* The corresponding dataset was split into two:

- The training dataset with 8048 entries, and
- The testing dataset with 1000 entries.

The training dataset was shuffled, and split into 70% for training the models, and 30% for cross-validation. The data was labelled as follows:

- **Useful**: Comments that are useful for code comprehension
- **Not Useful**: Comments that are not useful for code comprehension

Table 1
Description of the Dataset for the Task

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*/

4. Augmentation

The dataset augmentation process involved the integration of data generated by GPT-3, a powerful language model, to enrich the existing dataset. By leveraging GPT-3's natural language generation capabilities, additional comment data was created to expand the diversity and scale of the dataset. This augmentation strategy aimed to introduce a broader spectrum of comments, encompassing a wide range of writing styles, structures, and content. The incorporation of GPT-3 generated data was carried out to assess its potential in enhancing the training of machine learning models for comment quality evaluation. This approach allowed for the exploration of how AI-generated content could complement human-written data, contributing to a more comprehensive and robust dataset for improved model performance.

5. System Description

5.1. Text Preprocessing

All the links, punctuations, numbers and stop words have been removed. Then all words which have a POS tag other than Noun, Verb, Adverb and Adjective are removed. Lemmatization is used for grouping together the different forms of a word into a single word. NLTK wordnet [25] is used for lemmatization. Both training and testing datasets use same preprocessing steps.

5.2. Feature Extraction

TfidfVectorizer [26] is used for converting the text into numerical features. Tokenizer by Keras [27] library is used, along with TfidfVectorizer that was used from SciKit-Learn library.

5.3. Machine Learning Models

Two runs have been submitted for the task: one using Support Vector Machine (**SVM**) model, and another with **Naïve Bayes classifer** model. We have used the SciKit-Learn library for both of the models, with the parameters for the SVM model as follows:

- **C**: `(regularization parameter)` = 1
- **kernel**: `(kernel type)` = 'linear'

6. Findings

6.1. Without Augmentations

With these parameters set for the SVM model, the validation set gives a 77.26708074534162% accuracy score, along with an F1 score of 0.786464410735123.
Also, with the Naïve Bayes Classifier, the validation set gives a 60.993788819875775% accuracy score, along with an F1 score of 0.699233716475096.

Table 2
Results of Classifier Runs

Run	Macro F1 Score	Macro Precision	Macro Recall	Accuracy%
SVM	0.786464	0.772345	0.771381	77.2671
Naïve Bayes	0.699234	0.609193	0.644289	60.9938

6.2. With Augmentation

With these parameters set for the SVM model, the validation set gives a 77.64842840512223% accuracy score, along with an F1 score of 0.7830939828521743.
Also, with the Naïve Bayes Classifier, the validation set gives a 64.0279394644936% accuracy score, along with an F1 score of 0.695431994330003.

Table 3
Results of Classifier Runs

Run	Macro F1 Score	Macro Precision	Macro Recall	Accuracy%
SVM	0.783094	0.772302	0.774198	77.6484
Naïve Bayes	0.695432	0.611399	0.659775	64.0279

7. Conclusion

The tasks were accomplished employing basic machine learning models like SVM and the Naïve Bayes Classifier. The outcomes obtained from the SVM classifier indicate that there is room for enhancement, enabling the development of more intricate models that align better with the problem statement and yield superior results. Notably, Srijoni Majumdar et al. [28] have already achieved superior results using neural networks, and the author anticipates continuous improvement in these results over time.

Acknowledgments

Thanks to the creators of IRSE FIRE for giving this wonderful opportunity to work on such a project, and their constant technical support throughout the timespan.

References

[1] B. Fluri, M. Würsch, H. Gall, Do code and comments co-evolve? on the relation between source code and comment changes, 2007, pp. 70–79. doi:10.1109/WCRE.2007.21.

[2] M. Kajko-Mattsson, A survey of documentation practice within corrective maintenance, Empirical Software Engineering 10 (2005) 31–55. URL: https://doi.org/10.1023/B:LIDA.0000048322.42751.ca. doi:10.1023/B:LIDA.0000048322.42751.ca.

[3] J. Raskin, Comments are more important than code, ACM Queue 3 (2005) 64–. doi:10.1145/1053331.1053354.

[4] E. Wong, J. Yang, L. Tan, Autocomment: Mining question and answer sites for automatic comment generation, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, pp. 562–567. doi:10.1109/ASE.2013.6693113.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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).

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[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] E. Loper, S. Bird, Nltk: The natural language toolkit, 2002. URL: https://arxiv.org/abs/cs/

0205028. doi:10.48550/ARXIV.CS/0205028.

[26] V. Kumar, B. Subba, A tfidfvectorizer and svm based sentiment analysis framework for text data corpus, in: 2020 National Conference on Communications (NCC), 2020, pp. 1–6. doi:10.1109/NCC48643.2020.9056085.

[27] N. Ketkar, Introduction to Keras, 2017, pp. 95–109. doi:10.1007/978-1-4842-2766-4_7.

[28] 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.