Source Code Comment Classification using Naive Bayes and Support Vector Machine

Raj Shah[1,*,†]

[1]*Indian Institute of Technology Goa, Goa-403401*

Abstract

This paper proposes a framework for source code comment classification, which classify a comment based on its usefulness within the source code. This qualitative classification assists new developers in correctly comprehending the source code. We implement two binary classification mechanisms of source code comments based on two machine learning models: Naive Bayes and support vector machine. The classifier will classify each comment into two categories - *useful* and *not useful*. We extract comment features such as comment length, the position of comment within source code, and significant word ratio before training both models. We use a source code database of over 9000 instances written in C language in our work. Both models achieve an F1-score value of 0.632 and 0.765, respectively.

Keywords

Naive Bayes, Support vector machine, Comment classification, Qualitative analysis

1. Introduction

In software development, code comments play a pivotal role in enhancing code understanding and reusability. Our research centers on the critical evaluation of comment quality, emphasizing clarity and the avoidance of redundancy. Through our work, we endeavor to elevate overall code quality and boost developer productivity by filtering and prioritizing useful comments, ultimately decluttering the codebase and streamlining the development process.

In our paper, we introduce a binary classification algorithm tailored to C language source code comments, categorizing them as either "Useful" or "Not Useful." Leveraging Naive Bayes and Support Vector Machine (SVM) techniques, we analyze over 8,000 training samples and 1,000 test samples. We extract structural features, including comment length, position within the source code, and significant word ratio[1], to train our models.

For SVM, we employ the hinge loss function with a linear kernel. In contrast, the Multinomial Naive Bayes uses a probabilistic model that calculates class probabilities based on the training data, selecting the most probable class as the prediction. both models consistently achieve an average F1-score of 69.85%, showcasing their effectiveness.

Furthermore, to bolster our model's capabilities, we employed ChatGPT to augment our dataset. This innovative approach involved generating data labels for an additional 500 instances of code comment pairs, which we extracted from open-source GitHub C code repositories. The

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

raj2100789@gmail.com (R. Shah)

extraction process was systematically carried out using regular expressions, allowing us to obtain these code-comment pairs.

To evaluate the enhanced performance of our model, we leveraged a Large Language Model (LLM), specifically the chatgpt-3.5-turbo model, which played a pivotal role in generating labels for our data. We classified each code-comment pair as either "Useful" or "Not Useful" with the help of this LLM. This categorization was essential for assessing the effectiveness of our model in understanding and contextualizing code comments. The integration of ChatGPT not only expanded our dataset but also significantly contributed to an increase in accuracy, to 70.84%, further validating the robustness of our comment classification model. This augmentation process underscores the potential of combining state-of-the-art language models with machine learning techniques to enhance the performance and versatility of software development tools.

It's worth noting that, while there are alternative options available for LLMs, such as BARD, LLaMA, and several others, we chose to utilize the OpenAI LLM for several reasons. One of the primary considerations was cost efficiency, as it offers a cost-effective solution for our research needs. Additionally, the accessibility of OpenAI LLM's API keys was a significant factor, facilitating our research efforts when compared to other alternatives.

The rest of the paper is organized as follows. Section 2 discusses the background work done in the domain of comment classification. Details of existing methods are discussed in section 3. We discuss the proposed method 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 [1] extends the approach proposed in [17] to explore various vector space models [20] 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.

Our previous experience with LLMs involved their application in generating data related to voting processes and project selections. Beyond label generation, we have also employed machine learning models for various classification tasks, encompassing binary classification, multiclass classification, and multilabel classification, utilizing the extensive dataset at our disposal.

3. Task and Dataset Description

In this section, we describe 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 Naive Bayes and SVM are 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 9000 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 seed data is illustrated in table 1. The development dataset consists of 8000 instances, and the test dataset consists of 1000 instances. The model is trained again using an additional dataset, consisting of 500 instances, generated using ChatGPT. A sample LLM-generated data is illustrated in table 2.

4. Working Principle

We try two machine learning models - Multinomial Naive Bayes and support vector machine (SVM) to implement the binary classification functionality. The system takes comments as well as surrounding code snippets as input. We extract features such as comment length, the position of comment within source code, and significant word ratio[17] from the given input. The output of the feature extraction process is used to train both machine learning models. The training dataset consists of 8591 data instances (including an additional 500 data instances generated using ChatGPT) along with their labels. Among them, 3791 data instances are labeled as *not useful* and 4796 data instances are marked as *useful*. The description of each model is discussed in the following section.

4.1. Multinomial Naive Bayes

Multinomial Naive Bayes is a classification algorithm used for text data. It assumes that features (e.g., word counts) are conditionally independent given the class label.

#	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 seed data instance

In the probability model, We want to find the class label C_i that maximizes the posterior probability $P(C_i|D)$, where D is a comment.

$$P(C_i|D) = \frac{P(C_i) \cdot P(D|C_i)}{P(D)}$$

In text classification, we use a multinomial distribution for the likelihood model:

#	Comment	Code	Label
1	// Turns on test of xor function	{ "nosombrero", NULL, NULL, &nosombrero, PL_OPT_BOOL, "nosombrero", "No sombrero plot" },	Not Useful

2	// flip image up-down	free(img);
		*width = w;
		*height = h;
		*img_f = imf;
		return 0;
		}
			Not Useful

3	// save the plot	if (f_name)
		save_plot(f_name);
		}
		plFree2dGrid(z, XDIM, YDIM);
			Useful

Table 2
Sample LLM-generated (ChatGPT) data instance

$$P(D|C_i) = \prod_{j=1}^{n} P(X_j|C_i)$$

Where X_j is the count of each term in the comment.

To classify a comment, calculate

$$\text{argmax}_{C_i} \left(\log(P(C_i)) + \sum_{j=1}^{n} \log(P(X_j|C_i)) \right)$$

for each class and choose the class with the highest probability.

Tune the smoothing parameter (α) and class prior probabilities ($P(C_i)$) during training. For binary classification, we set a threshold on the posterior probabilities. Multinomial Naive Bayes is effective for text classification tasks, especially when features represent word counts.

4.2. Support Vector Machine

We have incorporated a Support Vector Machine (SVM) model into our binary classification task. The decision boundary is determined by the linear function outlined in Equation 1. In this setup, instances with an output exceeding 1 are assigned to one class, while those with an output less than -1 are categorized into the other class. The training of the SVM model involves the utilization of the hinge loss function, as shown below.

$$H(x, y, Z) = 0, \quad if y * Z \geq 1$$
$$= 1 - y * Z, \quad otherwise \tag{1}$$

The loss function suggests that the cost is 0 if the predicted and actual values are of the same sign. We calculate the loss value if they are of different signs. The Hinge loss function is used for the SVM model hyper-parameter tuning.

5. Results

We train both models in a system having an Intel i5 processor and 16GB RAM. We test both models using the test dataset. The test dataset consists of 1001 data instances, among which 719 data instances are labeled as *not useful* and 282 instances are marked as *useful*. Our Multinomial Naive Bayes model has been tested on this dataset and achieved an F1-score value of 0.632 (without ChatGPT-generated data) and 0.640 (with the augmented model). Similarly, the SVM model achieves a 0.765 F1-score value (without ChatGPT-generated data) and 0.776 (with the augmented model). Both models achieve high recall values of 0.652 and 0.763 without the LLM-generated data and 0.659 and 0.774, respectively with the augmented model. It shows that both models correctly predict useful comments in a better way. Both models achieve lower precision, such as 0.608 and 0.763, compared to the recall value. Apart from this, our model does not use any qualitative feature, which may be important to understand the usefulness of a comment within a source code. Using these qualitative features may increase the overall accuracy of the binary classification.

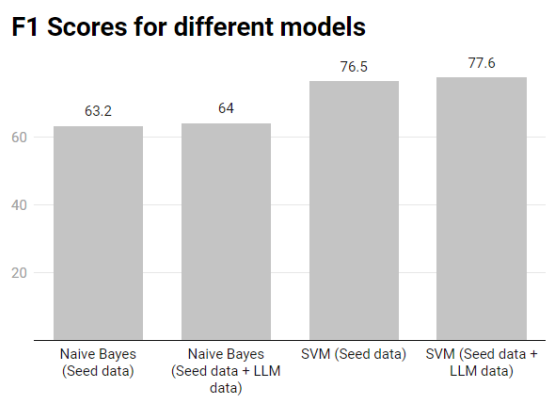

Figure 1: F1 Score vs Model

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 two machine learning models, Multinomial Naive Bayes, and support vector machine, to implement the binary classification task. We extracted three structural features: the length of the comment, the position of the comment within the source code, and the significant word ratio from each data instance. Both models have been trained using a training dataset with more than 8,000 data instances. Hinge loss has been used during hyper-parameter tuning for the SVM model. The models are tested on a test dataset of 1000 data instances. The Multinomial Naive Bayes model has been tested on this dataset and achieved an F1-score value of 0.632 (without ChatGPT-generated data) and 0.640 (with the augmented model), and the SVM model achieves a 0.765 F1-score value (without ChatGPT-generated data) and 0.776 (with the augmented model). Currently, we are using structural features for the classification task, which may not be sufficient for the qualitative analysis of the source code comments. In the future, we will use some qualitative features of the comment, which may increase the accuracy of the comment classification task.

References

[1] 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.

[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, 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.