Binary Classification of Source Code Comments using Machine Learning Models

Lisa Sarkar[1,*,†]

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

Abstract

This paper reports a detailed analysis on the viability of a classification framework which can classify a comment based on its usefulness within the source code. This classification will be helpful for new developers for correctly comprehending the source code. Three machine learning models: such as logistic regression, support vector machine, and multinomial naive Bayes are trained using an initial dataset called seed dataset. Each comment is classified into one of the two categories - *useful* and *not useful*. An accuracy of 82.92%, 83.92%, and 50.75% respectively is achieved from the initial training of three models. The dataset is then augmented using a new set of data extracted from several online resources. The corresponding class for the new set are generated using chatGPT large language model (LLM). The augmented dataset is then again used to train those three machine learning models. It is observed that for the new augmented dataset, the accuracy drops down for all three models due to inclusion of noise and biasness owing to the LLM generated dataset.

Keywords

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

1. Introduction

Software is emerging as the backbone of modern technology as they warrant many promising applications owing to their integration into electronics and appliances. They are simplifying the challenges of our daily life and making it easier in all aspects. For example, GPS software facilitates driving from one place to another. Constant modification of existing software and building of new software is the key of improving the software functionality which leads to an increase in the source code. Maintaining this large amount of source code is a crucial phase of *Software Development Life Cycle* (SDLC). In most of the cases, developers face numerous challenges during this source code maintenance including comprehending large code base in short period of time, outdated and incomplete required documents, unavailability of knowledge from previous developer to name a few.

This type of scenario can be tackled by following systematic process flow. New developers generally have the source code, sample test cases, requirement documents, and a debugger to implement new functionality. For further modification of the code, the developer must understand

[*]Forum for Information Retrieval Evaluation, December 15-18, 2023, India
[*]Corresponding author.

lisasarkar11@gmail.com (L. Sarkar)
0009-0006-3456-2409 (L. Sarkar)

the existing source code. So, they repeatedly run the current application on the sample test cases to identify execution patterns, understand the design, and to comprehend the program. But this whole process is time-hungry, effort-intensive, monotonous and sometimes becomes unmanageable. To overcome those bottlenecks, developers often follow shortcut method which further introduce errors that are difficult to filter. This results in the degradation of software quality and developer's efficiency. These types of situations demand a systematic quality-controlled development process for ease of use by the developers. Program comprehension is one such process for maintaining existing source code in a better way. This reverse engineering process is beneficial for reuse, inspection, maintenance, and many others in the context of software engineering[1].

Inserting comments within program is incumbent for better understanding of the source-code. Many developers are working on the same code-base and describe differently while inserting comments. Different description of same code-base decreases the program's readability. Therefore, a standardized procedure of writing codes and comments is imperative in order to enhance the readability. Still this approach is not effective for understanding an already registered code. An intuitive understanding of source code comments may be a good idea to solve the readability program. In recent years, researchers are exploring this domain to develop new applications aiming to enhance the efficiency of new programmers through understanding the existing code.

In this paper, we present a classification framework applied on a dataset of code and comment pairs written in C language. The work is done in three stages - Training of classification framework based on seed dataset, augment the dataset volume using large language model, and again train the classification framework using the augmented dataset. In the first stage, the framework takes code and comment pair as an input and classify them into one of the two classes - Useful and Not Useful. In this case, logistic regression method, support vector machine (SVM), and multinomial naive Bayes techniques are employed for comment classification. A training dataset of 9000 samples and test dataset of 1001 samples are used for this purpose. The model is validated using five fold cross-validation process. We employ linear kernel for the SVM strategy and L2 regularization method for the logistic regression strategy. In the next stage, another set of code-comment pair is gathered from online sources such as github. Consequently, chatGPT-4 large language model is used to categorise the newly gathered code-comment pair into two classes- useful and not-useful. This generated dataset is augmented with previous seed dataset. The newly generated dataset is used to train these classification frameworks again. A small reduction in all the F1 score values and accuracies is obtained which can be due to noise inclusion as part of the newly generated dataset.

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 [2] 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 [3, 4, 5, 6, 7, 8, 9, 10, 11].

In the realm of mining code comments and assessing their quality, several authors have conducted research. Steidl et al. [12] 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. [13] 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 [14]. Majumdar et al. [15, 16, 17, 18] 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 [19], it becomes crucial to assess the quality of code comments and compare them to human interpretation. The IRSE track at FIRE 2023 [20] expands upon the approach presented in a prior work [15]. It delves into the exploration of various vector space models [21] 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

The task of implementing binary classification framework is accomplished in three consecutive steps. Those are designing of classification framework with seed dataset, augmenting the seed dataset volume using large language model and train the same framework using the new augmented dataset. The source code and comments pairs are classified into two classes *useful* and *not useful* using the trained framework. The procedure takes a comment description with associated lines of code as input and generates a label such as *useful* or *not useful* corresponding to each code-comment pair. The classification system was developed using classical machine learning models such as logistic regression, naive Bayes, and SVM.

- *Useful* - The specified comment is appropriate for the corresponding source code.
- *Not Useful* - The specified comment is not appropriate for the corresponding source code.

The seed dataset has 9000 code-comment pairs which are written in C language. Each data contains comment text, surrounding code snippet, and a label that describes its usefullness. The whole dataset is gathered from the GitHub and is annotated with the help of a 14 annotators

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

group. An example of dataset is presented in table 1. Another set of code-comment pairs is collected from a different online resources and is augmented with the above mentioned dataset. The set of code-comment pairs is categorized into two above-mentioned classes using a large language model. This newly generated dataset in then added with the seed dataset.

The classification model is then again trained using this augmented dataset in order to understand the effect of augmentation. Different factors are analysed including noise inclusion, distribution of dataset, which cause the change in accuracy during the training of classification framework with augmented dataset.

4. Working Principle

A binary classification system was implemented with the help of three machine learning models - logistic regression, support vector machine, and multinomial naïve Bayes. The system took both the source code and corresponding comments as input. Considering the task criteria we did not use any deep learning frameworks in our classification system. The comments are first tokenized using English word lemmatizer. Following that the set of tokens are vectorized using TF-IDF vectorizer. The TF-IDF matrix generated from the vectorization step along with class labels was used as feature fed into the classification models. These models are trained using the primary seed data and tested using test dataset. The unlikeliness in training data was controlled using five-fold cross validation. We will briefly elucidate each machine learning models in the subsequent subsections.

4.1. Logistic Regression

We use logistic regression for the binary comment classification task where a logistic function is used in order to keep the output 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) is referred as the linear regression equation whose output (Z) is passed to the logistic function. The logistic function is defined in equation 2. The binary class is predicted from the probability value generated by the logistic function based on the acceptance threshold. The threshold value is kept to 0.6 which is in favor of the *useful* comment class. A three-dimensional input feature is extracted from each training instance which is passed to the regression function. During training the Cross entropy loss function is used for the hyper-parameter tuning.

4.2. Support Vector Machine

In the next step, a support vector machine model is implemented for binary classification task. Classification is done based on the output of the linear function (equation 1). If the output is greater than 1, then it is classified with one type of class, and if the output is less than -1, then it is classified it with another class. We train the SVM model using the hinge loss function, as shown below.

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

It is noticed from the loss function, that the cost will be 0 if the predicted and actual values are of the same sign. In this case, the loss value is called if the predicted and actual values are of different signs. The Hinge loss function is used for the SVM model hyper-parameter tuning.

4.3. Multinomial Naive Bayes

Multinomial Naïve Bayes model is also used in this task mainly for text classification. This model uses the Bayes' theorem mentioned as below:

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

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

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

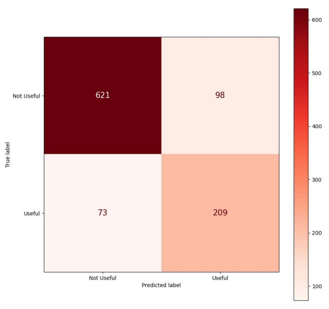
$P(X|y)$ is the likelihood, representing the probability of observing features X given class y.
$P(y)$ is the prior probability of class y.
$P(X)$ is the probability of observing features X, which acts as a normalization constant.

Multinomial Naive Bayes operates on the assumption that each of the features are conditionally independent of the other given some class.

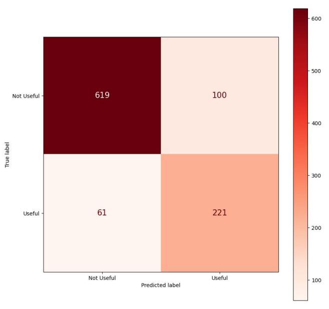
5. Results

A system with Intel i5 processor and 32 GB RAM is employed for the task implementation. The whole task has three steps, as mentioned earlier. At first, the seed dataset is divided into two segments training data (90%) and validation data (10%). Training dataset is exploited to train the three ML models - logistic regression, support vector machine, and multinomial naive Bayes. The test dataset contains 1001 instances. Among them, 719 instances are labeled as *not useful* and 282 instances are *useful*. All three models are tested on this test dataset and an overall accuracy of 82.92%, 83.92%, and 50.75% are achieved for logistic regression, support vector machine, and multinomial naive Bayes model respectively. The corresponding confusion matrix are plotted in figure 1. We noticed that the naive Bayes algorithm fails to predict the *not useful* data efficiently which leads to a degradation in the overall accuracy.

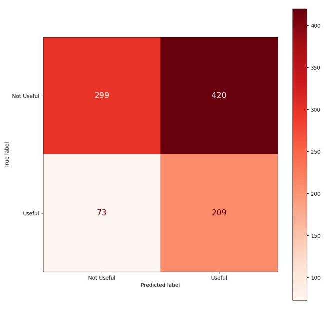
Another dataset is generated using large language model which consists of 311 *useful* samples and 21 *not useful* samples. This generated data is then augmented with seed data. This new dataset is then again divided into two parts - training and validation dataset. Furthermore, those new training and validation dataset are used to train the same classification models. The newly trained models are tested with the same test data. An overall accuracy of 82.92%, 84.12%, and 50.05% are achieved from the three models respectively. The individual confusion matrix for all three models are also displayed in figure 2. The evaluation results for all three models are illustrated in table 2. It is evident that the models trained with the augmented dataset experience a slight decrease in the accuracy compared to the previously achieved accuracy from seed dataset. This may be attributed to the incorporation of noises in the seed data from large language models. This noises are mainly generated because of the imperfection of large language model such as chatGPT 4 in our case, which leads to a decrease in the overall accuracy. Still we can argue that the augmented dataset is well-balanced for machine learning model training and generates a similar accuracy as for the initial seed dataset.



(a) Logistic regression

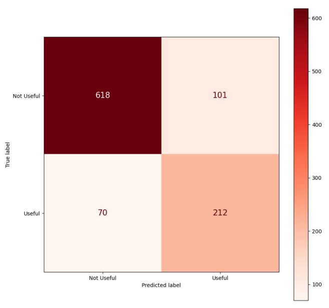


(b) Support vector machine

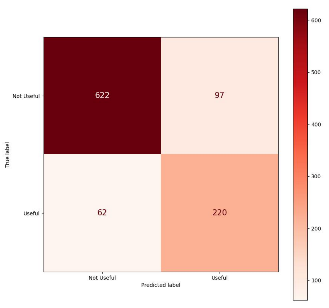


(c) Multinomial naive Bayes

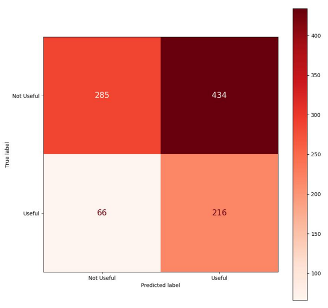
Figure 1: Confusion matrix for classification models related to seed data



(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

This paper proposes a framework for source code comment classification, which classify a comment based on its usefulness within the source code. Three machine learning models such as logistic regression, support vector machine, and multinomial naive Bayes are implemented and trained using seed dataset. These classifier classify each comment into two categories - *useful* and *not useful*. These three models exhibit an accuracy of 82.92%, 83.92%, and 50.75% respectively. Subsequently this seed dataset is augmented with a newly generated dataset that are gathered from online sources. The corresponding class for the new set are generated using chatGPT large language model (LLM). The newly generated augmented dataset is again used to train all the models. It is observed that the new augmented dataset drops down the accuracy for all three models due to inclusion of noise and biasness owing to the LLM generated dataset.

References

[1] M. Berón, P. R. Henriques, M. J. Varanda Pereira, R. Uzal, G. A. Montejano, A language processing tool for program comprehension, in: XII Congreso Argentino de Ciencias de la Computación, 2006.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[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] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[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, 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.

[21] 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.