

Source Code Comment Classification using machine learning algorithms

Trisha Datta^{1,*†}

¹Sri Ramaswamy Memorial Institute of Science and Technology, Kattankulathur

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 three machine learning models: logistic regression, support vector machine, and multinomial naive Bayes which are trained using an initial seed dataset. These classifier will classify each comment into two categories - *useful* and *not useful*. The initial trained models achieves accuracy of 83.42%, 84.72%, and 50.45% respectively. 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). All three models are again trained with the augmented dataset. The new models are tested with the same test dataset. We observe that all three models generates a little lower accuracy which demonstrates the inclusion of noise and biasness due to the LLM generated dataset.

Keywords

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

1. Introduction

In today's technology-driven world, programming has a crucial role in all aspects of life, be it medicine, communication, or transportation. As the number of applications increases, the quantity of source code also becomes more. Revisions to pre-existing code also contribute to this. Managing and debugging a huge bulk of code in a limited time is difficult. Trying to achieve such a task can lead to poor documentation of the code. This makes recycling this old code to make something new and improved a nearly impossible task.


The programmer must execute the source code repeatedly in order to understand its working. This is a slow, boring process that takes a lot of effort. So, programmers may end up trying to speed up the process, which leads to errors. This reduces the functionality of the new code or software. One way of tackling this issue is the comments used in coding. They provide a shortcut to understanding the original programmer's thought process while developing the code. However, all programmers have different personal styles of writing these comments,


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

† Corresponding author.

✉ trishadatta2003@gmail.com (T. Datta)

ORCID 0000-0000-0000-0000 (T. Datta)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

 CEUR Workshop Proceedings (CEUR-WS.org)

which makes it confusing. Therefore, it is essential to follow certain rules while programming, including the comments.

But in the case of already existing source code with no proper comments or documentation, we need a different solution. Building software that helps improve the readability of such existing source code is an area of interest for researchers right now. Such efforts can help programmers save time and develop improved applications.

In this paper, we implement a classification framework to categorize a code-comment pair into two classes - useful and not useful. A seed dataset with almost 10,000 code-comments pairs written in C language is used for training the framework. Three machine learning models, such as support vector machine, logistic regression, and multinomial naive bayes are tried out to implement the binary classification framework. Also, we pull out a set of code comment pairs from internet resources, which is written in C language. A large language model, chatGPT 4, is used to label them into two classes. The seed dataset is augmented using the new data and again trains the classification models. The new F1 score and accuracy are compared with the previous results. This comparison illuminates the advantages and drawbacks of the new dataset for our classification model.

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

Understanding a program automatically is a well-known research area among people working in the software domain. A significant number of tools [1, 2, 3, 4, 5, 6] have been proposed to aid in extracting knowledge from software metadata [7] like runtime traces or structural attributes of codes. New programmers generally check for existing comments to understand a code flow. Although, every comment is not helpful for program comprehension, which demands a relevancy check of source code comments beforehand. Many researchers worked on the automatic classification of source code comments in terms of quality evaluation. For example, Omal et al.[8] discussed that the factors influencing software maintainability can be organized into hierarchical structures. The author defined measurable attributes in the form of metrics for each factor which helps measure software characteristics, and those metrics can be combined into a single index of software maintainability. Fluri et al.[9] 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[10] 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[11]. 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.[12] 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.[13] 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 [14] 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[14, 15, 16, 17], but still, automatic quality evaluation of source code comments is an important area and demands more research.

With the advent of large language models [18], 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 [19] extends the approach proposed in [14] 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.

3. Task and Dataset Description

This section describes the task addressed in this paper. The task can be addressed in three modules - classification framework design with seed dataset, augmenting the seed dataset using a large language model, and training the same classification framework with the new dataset. We aim to implement a binary classification framework to classify source code and comments pairs into two classes - *useful* and *not useful*. The procedure takes a comment description with associated lines of code as input. The output will be a label such as *useful* or *not useful* for the corresponding comment. This framework is designed to help developers comprehend the associated code. Classical machine learning algorithms such as naive Bayes, logistic regression, and SVM are employed 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.

The seed 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 data is illustrated in table 1. We augment this dataset with another set of code-comment pairs collected from different online resources. The set of code-comment pairs is categorized into two above-mentioned classes using a large language model. The generated subset is then adjoined with the seed dataset.

This augmented dataset is used to train the classification model another time to understand the effect of augmentation. We analyze noise inclusion, distribution of dataset, and many other

#	Comment	Code	Label
1	/*test 529*/	<pre> -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 </pre>	Not Useful
2	/*cr to cr,nul*/	<pre> -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; </pre>	Not Useful
3	/*convert minor status code (underlying routine error) to text*/	<pre> -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 </pre>	Useful

Table 1
Sample data instance

factors that affect the change in accuracy while training with the augmented dataset.

4. Working Principle

We try to train three machine learning models- Logistic Regression, Support Vector Machine, and Multinomial Naïve Bayes to implement the binary classification functionality. Note that we do not implement any of the deep learning frameworks due to the constraint of the task criteria.

The system takes the comments and corresponding code segments as input. The comment is lemmatized and tokenized using English word tokenizer before transforming it to a numerical vector space. We assume that all comments are written in English language. We use TF-IDF vectorizer to transform each English keyword into numerical vector. It generates a TF-IDF matrix for all keywords present in the comment set. This matrix is considered as a feature set in our classification framework. The entire dataset is divided into two disjoint subsets - training set (90%) and test set (10%). We extract features from the training data using above mentioned techniques. The TF-IDF matrix along with class levels are fed into machine learning models for training. These models are used to automatically assign each code-comment pair into two classes. We will now briefly discuss about three machine learning classification models in the subsequent subsections.

4.1. Logistic Regression

Logistic Regression is used for the binary comment classification. We use a function to keep regression output between 0 and 1. The logistic function is defined as given below:

$$Z = Ax + B \quad (1)$$

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

The output of the linear regression equation (refer equation 1) is passed to the logistic function (refer equation 2). The probability value generated by the logistic function is used to predict the binary class based on the acceptance threshold. The threshold value of 0.6 is kept 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. The Cross-Entropy Loss function is used during training for the Logistic Regression hyper-parameter tuning.

4.2. Support Vector Machine

We next use a Support Vector Machine model to do the binary classification. We take the output of the linear function, and if the output is greater than 1, we classify it with one class and if the output is less than -1, we classify it with the other class. We train the SVM model using the Hinge Loss function, as given below:

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

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

4.3. Multinomial Naive Bayes

We also use Multinomial Naïve Bayes model, which is used mostly in text classification, to implement the binary classification. This model uses the Bayes' theorem mentioned as below:

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

where,

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

$P(X|y)$ is the likelihood, representing the probability of observing features X given class y .

$P(y)$ is the prior probability of class y .

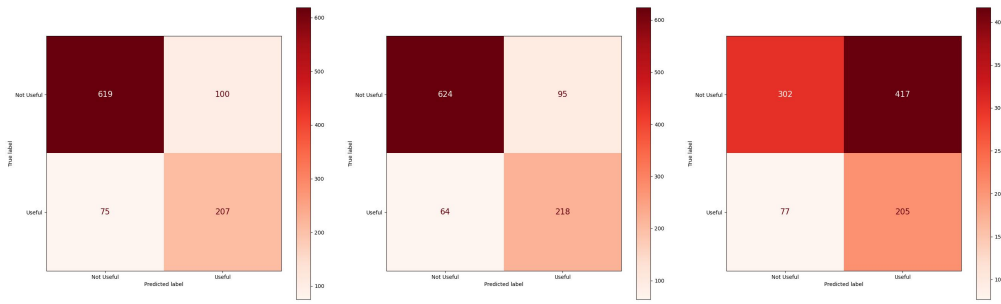
$P(X)$ is the probability of observing features X , which acts as a normalization constant.

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

5. Results

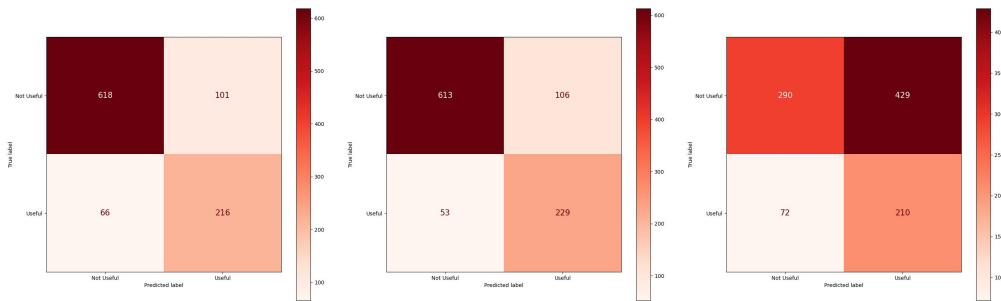
The task has been implemented in a system with an Intel i5 processor and 32 GB RAM configuration. As mentioned earlier in 3, the whole task has three steps. Initially, the seed dataset is randomly divided into two parts - training data (90%) and validation data (10%). Three classification model such as logistic regression, support vector machine, and multinomial naive Bayes are trained using the same training dataset. The test dataset consists of 1001 data instances, among them, 719 instances are labeled as *not useful* and 282 instances are *useful*. We test all three models on this test dataset and achieves an overall accuracy of 83.42%, 84.72%, and 50.45% respectively. The relevant confusion matrix are displayed in figure 1. It is evident that the naive Bayes algorithm does not predict *not useful* data well. This leads to low overall accuracy for the naive Bayes unlike the other two algorithms.

We augment the seed data with a large language model generated data consisting of 309 *useful* samples and 25 *not useful* samples. The overall dataset is then divided into training and validation dataset. Same classification models are again trained with the new training and validation dataset. The newly trained models are tested with the same test data. These models achieve overall accuracy of 83.32%, 84.12%, and 49.95% respectively. The individual confusion matrix are also displayed in figure 2. We notice that the models trained from the augmented dataset experience a small decrease in accuracy from the accuracy achieved for seed data. The evaluation matrix for all three models are illustrated in table 2. This demonstrates that the large language models introduce some noises in the seed data which affects the overall accuracy of all three models. This noises are generated because of the imperfection of large language model such as chatGPT 4 in our case. Still we can argue that the augmented dataset is well-balanced for machine learning model training.



(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 has developed three binary classification models in the domain of code and comment classification. The classification models are trained based on a seed data having two different classes. All source data present in the dataset are written in C language and all comments are written in English language. Each comment are tokenized and further vectorized using the TF-IDF vectorizer. This numerical vector are used as feature in our classification models. Also, the initial seed dataset is augmented using new data extracted from online sources. The newly extracted data are then classified into same two classes using a large language model, chatGPT. All three classification models are again trained with the augmented dataset. We have observed that models trained with augmented data are producing a little lower accuracy than those models trained with initial seed data. This exemplify for noisy data introduced as part of the LLM generated dataset. We have also done a comparative analysis on both results. We can argue that the biasness and noise present in the augmented dataset are degrading the accuracy

Table 2

Experimental results of three classification model on the test dataset

Dataset	Models	Precision	Recall	F1-score	Accuracy (%)
Seed data	Logistic regression	0.8102	0.7936	0.8009	83.42
	Support vector machine	0.8343	0.8086	0.8191	84.72
	Naive Bayes	0.5775	0.5675	0.5009	50.45
Seed data + LLM generated data	Logistic regression	0.8127	0.7924	0.8011	83.32
	Support vector machine	0.8323	0.802	0.8138	84.12
	Naive Bayes	0.574	0.5649	0.4963	49.95

of our classification models.

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] P. Oman, J. Hagemester, Metrics for assessing a software system’s maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.
- [9] 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.
- [10] 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.

- [11] 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.
- [12] T. Tenny, Program readability: Procedures versus comments, *IEEE Transactions on Software Engineering* 14 (1988) 1271.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [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.