Source Code Comment Classification using Logistic Regression and Support Vector Machine

Soumen Paul[1,*,†]

[1]Indian Institute of Technology, Kharagpur, West Bengal, Kol-700129

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: logistic regression 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 0.688 and 0.684, respectively.

Keywords

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

1. Introduction

Software controls almost every sector of human essentials, such as finance, hospitals, transport, and many more. Many organizations constantly modify existing software or build new software based on human requirements. The amount of source code gradually increases with the increase of software functionality daily. Maintaining this large amount of source code is a crucial phase of *Software Development Life Cycle* (SDLC). Developers often need to fix bugs, develop new source code, or upgrade already deployed applications. Also, developers used to get a short period to comprehend a large code base. This increases the vulnerability of the developer and eventually leads to improper coding practices. In most cases, the documentation, such as requirement specification, high-level design, low-level design, etc., and change traces are outdated and incomplete, and the knowledge transfer process or help from the earlier developers is unobtainable.

Developers need to follow a proper process to tackle this type of scenario. New developers generally have the source code, sample test cases, requirement documents, and a debugger to implement new functionality. The developer has to understand the existing source code before modifying it. Thus, they are forced to repeatedly run the current application on the sample

[*]Forum for Information Retrieval Evaluation, December 9-13, 2022, India
*Corresponding author.

soumenpaul165@gmail.com (S. Paul)
0000-0003-3548-2663 (S. Paul)

test cases to identify execution patterns, understand the design, and comprehend the program. This process is time-taking, monotonous, effort-intensive, and often not manageable. These shortcomings force the developers to perform quick patches, further introducing errors that are sometimes difficult to filter. These limitations drastically dip the software quality and lower the developers' productivity. These types of situations demand a systematic quality-controlled development process for ease of use by the developers. Program comprehension is one such process of 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].

Comment within a program plays an essential role in understanding and assessing the program in a better way. Many developers are working on the same codebase and think differently while giving comments. These versatile approaches of inserting comments in the same codebase decrease the program's readability. Hence, a standard way of writing code and giving comments is necessary to increase the program's readability. But this approach does not help programmers in understanding an already registered code. An intuitive understanding of source code comments may be a good idea to solve the readability program. Researchers are working on this domain to develop such applications, which help new programmers understand existing code, increasing their programming efficiency.

In this paper, we propose a binary classification algorithm to understand the source code comments present in a program written in C language. We classify each comment into two classes - Useful and Not Useful. We try to use logistic regression and support vector machine (SVM) techniques for comment classification. We have a training data set of over 8000 samples and test data set of 1000 samples. We extract relevant structural features such as comment length, the position of comment within source code, and significant word ratio[2] and train the classification model on that. Five fold cross-validation is used for the model validation. We employ the hinge loss function for the SVM strategy and the cross-entropy loss function for the logistic regression strategy. Both models have achieved 68.6% as an average F1-score value.

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. 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.[3] 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.[4] 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[5] 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[6]. 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.[7] 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.[8] 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 [2] 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[9, 10], but still, automatic quality evaluation of source code comments is an important area and demands more research.

3. Task and Dataset Description

In this section, we have described 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 logistic regression and SVM can be 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 data is illustrated in table 1. The development dataset consists of 8000 instances, and the test dataset consists of 1000 instances.

4. Working Principle

We try two machine learning models - logistic regression and support vector machine (SVM) to implement the binary classification functionality. The system takes comments as well as

#	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 data instance

surrounding code snippets as input. We extract features such as comment length, the position of comment within source code, and significant word ratio[2] from the given input. The output of the feature extraction process is used to train both machine learning models. The training dataset consists of 8047 data instances along with their labels. Among them, 3710 data instances are labeled as *not useful* and 4337 data instances are marked as *useful*. We use a five-fold cross-validation process during training the model to deal with the unlikeliness of the training data. The description of each model is discussed in the following section.

4.1. Logistic Regression

We use logistic regression for the binary comment classification task which uses a logistic function to keep the regression output between 0 and 1. The logistic function is defined as follows:

$$Z = Ax + B \tag{1}$$

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

The output of the linear regression equation (refer to equation 1) is passed to the logistic function (see equation 2). The probability value generated by the logistic function is used for binary class prediction based on the acceptance threshold. We keep the threshold value of 0.6 in favor of the *useful* comment class. We have a three-dimensional input feature extracted from each training instance which is passed to the regression function. The Cross entropy loss function is used during training for the hyper-parameter tuning.

4.2. Support Vector Machine

We also implement a support vector machine model for our binary classification task. We take the output of the linear function (given in equation 1), and if the output is greater than 1, then we identify it with one class, and if the output is less than -1, then we classify it with another class. We train the SVM model using the hinge loss function, as shown below.

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

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 8GB RAM. We test our both models using our 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 logistic regression model has been tested on this dataset and achieved an F1-score value of 0.688. Similarly, the SVM model achieves a 0.684 F1-score value. The corresponding confusion matrix is shown in table 2 and 3. Both models achieve high recall values of 0.851 and 0.84, respectively. It shows that both models correctly predict useful comments in a better way. Both models achieve lower precision, such as 0.574 and 0.577, compared to the recall value. Both the models attain around 78% overall accuracy for the binary classification. Apart from this, our model is not using 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.

	Useful	Not Useful
Useful	240	42
Not Useful	178	541

Table 2
Confusion Matrix for Logistic Regression

	Useful	Not Useful
Useful	237	45
Not Useful	174	545

Table 3
Confusion Matrix for Support Vector Machine

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, logistic regression 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 8000 data instances. Cross entropy loss and hinge loss have been used during hyper-parameter tuning for both models, respectively. The models are tested on a test dataset of 1000 data instances. The logistic regression model achieved an F1-score value of 0.688, and the SVM model achieved an F1-score value of 0.684. Both models have achieved an overall accuracy of 78%. 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] 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. 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.

[3] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.

[4] 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.

[5] 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.

[6] 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.

[7] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.

[8] 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.

[9] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. D Clough, S. Chattopadhyay, P. Majumder, Overview of the IRSE track at FIRE 2022: Information Retrieval in Software Engineering, in: Forum for Information Retrieval Evaluation, ACM, 2022.

[10] 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.