An Ensemble Approach to Detect Code Comment Inconsistencies using Topic Modeling

Fazle Rabbi, Md. Nazmul Haque, Md. Eusha Kadir, Md. Saeed Siddik and Ahmedul Kabir
Institute of Information Technology
University of Dhaka, Bangladesh
Email: {bsse0725, bsse0635, bsse0708, saeed.siddik, and kabir}@iit.du.ac.bd

***Abstract*—In modern era, the size of software is increasing, as a result a large number of software developers are assigned into software projects. To have a better understanding about source codes these developers are highly dependent on code comments. However, comments and source codes are often inconsistent in a software project because keeping comments up-to-date is often neglected. Since these comments are written in natural language and consist of context related topics from source codes, manual inspection is needed to ensure the quality of the comment associated with the corresponding code. Existing approaches consider entire texts as feature, which fail to capture dominant topics to build the bridge between comments and its corresponding code. In this paper, an effective approach has been proposed to automatically extract dominant topics as well as to identify the consistency between a code snippet and its corresponding comment. This approach is evaluated with a benchmark dataset containing 2.8K Java code-comment pairs, which showed that proposed approach has achieved better performance with respect to the several evaluation metrics than the existing state-of-the-art Support Vector Machine on vector space model.**

***Index Terms*—Source Code, Code Comment, Topic Modeling, Software Artifact Analysis**

I. Introduction

Code comments with its corresponding source code are the main artifact of any software systems. For the management of software evolution and maintenance, developers provide comments with a code fragment which give insightful information about a software system. Comments are very important as they are more natural, descriptive and easy to understand than source code [1], [2]. In large projects, new developers are highly dependent on code comments to understand its corresponding source codes. Researchers found that code and comments evolve over time [3] and this evolved codes and comments become inconsistent to each other. Because of changing codes frequently and keeping corresponding comments same, comments become invalid or inconsistent with corresponding source code.

Tracking the inconsistency of source code and its comment, several diverse approaches have been proposed. Where most of the approaches apply Information Retrieval (IR) techniques to collect lexical information with the assumption that the textual information of source code and comment are same. However, that assumption can be violated [4] in several cases, for example, the vocabulary developers use to write source code can be different from the vocabulary of comment (e.g. synonym). Nevertheless, there is no sufficiently rich literature to track this inconsistency because of lacking standard datasets. A benchmark dataset has been provided [5] with a proposal to measure the coherence between source code and comment. Lexical similarity has been collected by using Vector Space Model to classify the text using tf-idf [6] and finally the code-comment inconsistency is measured using Support Vector Machine (SVM). However, this approach uses all of the vocabulary as features which can take a huge execution time.

DOI reference number: 10.18293/SEKE2020-062.

By analyzing existing literature, some insights of source code and comments have been found, which are concluded below as the research direction in this domain.

- A single word (topic) is more important than a large number of similar words (features). For example, if a bag of words is found from a java method like, “dropdown”, “chrome”, “menu”, “http” or “browser”, a topic related to “browser” can represent these words.
- The size of comments is less than the size of source code. So, the source code and comment need to be represented into a fixed-sized common topic.
- Synonymous words have been chosen by developers while writing comment with respect to source code. So, to capture the semantic information between source code and comment, the vocabulary information needs to be incorporated.

To capture these insightful information, several Research Questions (RQ) have been raised to propose an efficient inconsistency detection approach, which are listed below.

- **RQ1:** How to comprehend the insight meaning of a code and comment pair?
- **RQ2:** How to measure the relation between the code and comment pair?

We focused on the above research questions as our objectives and tried to answer them throughout the newly proposed code comment inconsistency detection technique. This paper proposes an automated approach to identify the inconsistency of source code with its respective comments. The breakdown of the contributions of this paper are listed as follows.

- Datasets are pre-processed to capture more meaningful information about source code and comments, e.g., developers defined simple name.

- Latent Dirichlet Allocation (LDA) has been used for representing the similar words into topics.
- A fusion approach (ensemble Random Forest) has been proposed for measuring the probability of the inconsistency between code and comments, where SVM is used to discriminate consistent and inconsistent comments.
- The proposed approach has been compared with state of the art baseline classification approaches and it is evident that this approach performs better in terms of Accuracy and Area Under Precision-Recall Curve (AUCPR).

In section II, an overview of the methods are briefed which are important to understand the proposed method. Dataset parsing and pre-processing is discussed in section III-A. Proposed method is explained in section III-B. The dataset description and experimental results are presented in section IV. Related works are reviewed in section V, and finally this work has been concluded in section VII.

II. Background

To understand the proposed approach, knowledge about Topic Modeling, Random Forest (RF) and Support Vector Machine (SVM) is needed which are briefly discussed here.

A. Topic Modeling

Topic Modeling is a subfield of Machine Learning and Natural Language Processing. It is one type of statistical model which follows unsupervised machine learning technique to provide abstract topics for a given document. Latent Dirichlet Allocation (LDA) is a type of topic modeling which is used in this paper. LDA is trained using a set of documents and with a given number of topics. It provides a probability distribution of words for a topic and a probability distribution of topics for a document as output.

B. Random Forest

Random Forest [7] is an ensemble learning approach for classifying data. In training time, it builds a multitude of decision trees. Each decision tree predicts a class label of a new input data pattern and RF merges them together to get a more accurate and stable prediction. RF is a fast, simple and flexible machine learning algorithm. In this paper RF receives the topic distribution gained from LDA as input and produces output for the next procedure.

C. Support Vector Machine

A Support Vector Machine [8] is a discriminative classifier formally defined by a separating hyperplane. While training this classifier, it finds the maximum-margin hyperplane that separates the group of data points into two classes. A new incoming pattern is classified in the class according to the side of the hyperplane.

III. Proposed Approach

The proposed ensemble approach to detect code comment inconsistency is described in this section thoroughly. Before training, the way of pre-processing code and comment pairs is also discussed here.

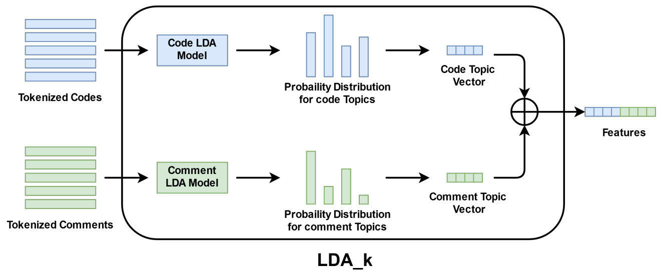

Fig. 1. Topic Modeling (LDA).

A. Code-Comment Parsing

The raw code comment pairs need to be processed to create features for training. The two steps which are followed to make the features during pre-processing are described next.

*1) **Process Code and Comment Pairs**:* The pairs of codes and comments need to be parsed into tokens to execute the next steps. At the beginning, all newlines, tabs and special characters like braces, semicolons, full-stops are removed. Extra whitespaces are also removed from the remaining code and comments. Words are also split based on camel cases. After tokenizing, every code and comment is turned into a bag of word tokens. Finally each of the words are lemmatized into root words.

*2) **Create vectors from code and comments**:* After processing the code-comment pairs into bag of word tokens, two corpora for codes and comments are built. Each of these corpora turns a code/comment into an index vector. These index vectors of code/comment is passed into its corresponded Latent Dirichlet Allocation (LDA) model. There are two identical LDA models for training code and comment index vectors separately. For every k number of topic, these LDA models provide two separate probability distributions for a pair of code and comment which are concatenated to produce feature vector using Eq. (1).

$$concatenatedfeature_k = LDA_k(code) \oplus LDA_k(comment) \quad (1)$$

Here, k is the number of topics and $\oplus$ is used for concatenating vectors. This features vector is now ready to be used for Inconsistency Detector described in the next section. The overview of preparing feature vector is illustrated in Fig. 1.

B. Inconsistency Detector

As discussed above, the extracted feature vector of k number of topics is fed into a random forest as input. Based on this input features, the random forest model produces a consistency score for a code-comment pair. A random forest built from k topics, RF_k, provides a consistency score, $score_k$ derived in Eq. (2).

$$score_k = RF_k(concatenatedfeature_k) \quad (2)$$

As the number of topics for both comment and code can be varied, it is needed to incorporate different number of topics to find the informative and effective features. For different

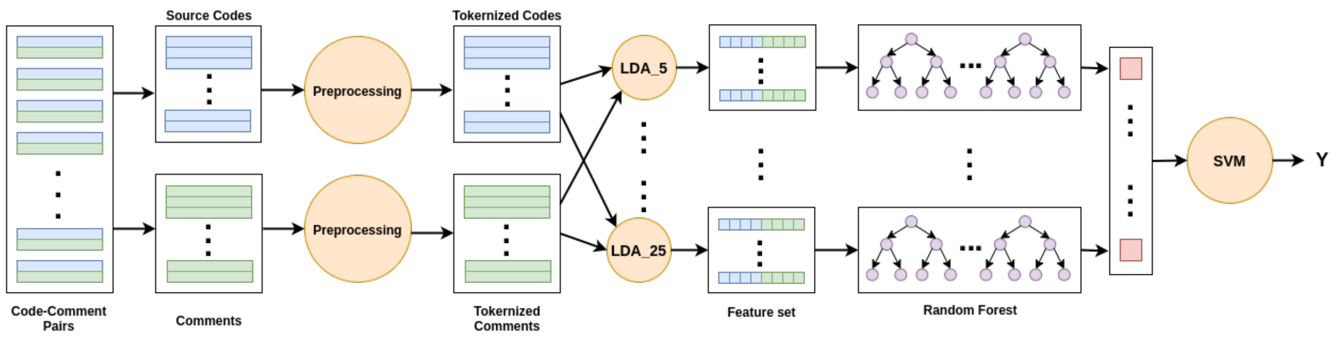

Fig. 2. Overall process of the proposed method.

TABLE I
DESCRIPTIVE STATISTICS OF THE DATASET

Application	Files	Classes	Methods	Methods with comments	Coherent	Non-Coherent	Total	Not Included
CoffeeMaker	7	7	51	47 (92%)	27	20	47	0
JFreeChart-0.6.0	82	83	617	485 (79%)	406	55	461	24
JFreeChart-0.7.1	124	127	807	624 (77%)	520	68	588	36
JHotDraw-7.4.1	575	692	6414	2480 (39%)	762	1025	1787	693
All	788	909	7889	3636 (46%)	1715	1168	2883	753

number of topics ranging from i to j on interval 1, different random forest models are built. Each random forest model returns a consistency score for a code-comment pair. These consistency scores derived from m different random forests are needed to be fused. Here, SVM is used to fuse the consistency scores provided by m random forests and predict the outcome using Eq. (3).

$$\hat{Y} = SVM(\oplus_{k=i}^{j} score_k) \tag{3}$$

Fig. 2 describes the overall procedure of proposed method.

IV. EXPERIMENTS & RESULT ANALYSIS

We used a dataset provided by Corazza et al. [9] in this experiment. Four versions of three java projects CoffeeMaker[1], JFreeChart[2] and JHotDraw[3] are used in this dataset. Some descriptive statistics of these projects are reported in Table I.

We split the dataset randomly into 90% training and 10% testing set. After training finished, we run our model on testing set. The result is evaluated based on two evaluation metrics namely Accuracy and Area Under Precision-Recall Curve (AUCPR). Table II and III report the performance comparison of our proposed method with other methods in terms of accuracy and AUCPR. We report the average performance by applying 10 fold cross-validation. To find the outputs of the existing approach, we re-implemented it.

From Table II, it can be observed that, the accuracy of the proposed method is better than the existing approach [5] for all of the projects except JHotDraw-7.4.1. At first a single Random Forest having 10 features (RF_10) is used to classify consistent and inconsistent code-comment pairs. As the result of this approach was not satisfactory, an ensemble RF with SVM (Proposed) is used. By using this, the accuracy increases for all of the projects and looks very promising.

TABLE II
PERFORMANCE COMPARISON IN TERMS OF ACCURACY

Dataset	RF-10	Coherence	Proposed
CoffeeMaker	0.830	0.873	0.895
JFreeChart-0.6.0	0.879	0.835	0.918
JFreeChart-0.7.1	0.876	0.875	0.898
JHotDraw-7.4.1	0.743	0.811	0.801
All	0.803	0.837	0.841

In Table III, the result is showed based on AUCPR which also denotes that, the result of the proposed approach performs better than the existing one. It can also be observed that, AUCPR values are improved when using the ensemble of RFs for different number of topics instead of a single Random Forest with constant number of topics (e.g. RF_10).

TABLE III
PERFORMANCE COMPARISON IN TERMS OF AUCPR

Dataset	RF-10	Coherence	Proposed
CoffeeMaker	0.878	0.943	0.975
JFreeChart-0.6.0	0.938	0.909	0.941
JFreeChart-0.7.1	0.924	0.941	0.962
JHotDraw-7.4.1	0.837	0.882	0.855
All	0.888	0.888	0.912

We also found that the proposed approach is promising while comparing the training time with state-of-the-art method. For small projects training time is almost same in both cases. However, the proposed approach trains faster than the existing approach for large projects.

[1]agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker
[2]www.jfree.org/jfreechart
[3]www.jhotdraw.org

V. Related Works

There have been some previous works in this field related to code comment relation. The earliest work related to code comment inconsistency that we studied is “iComment: Bugs or Bad Comments” [10]. In this work, authors proposed an approach to detect code comment inconsistency in locking and calling mechanism. They limited their scope to the comments related to programmers' assumptions and requirements.

Another work for testing Javadoc comments was proposed by Tan et al. to detect comment-code inconsistencies called @TCOMMENT [11]. The authors considered method properties for null values and related exceptions. They set some rules for @param tags in javadoc comments and null pararamer exception statement to detect inconsistencies between codes and comments using Natural Language Processing. The scope of their work is limited to only comments related to null reference and throwing exceptions.

Ratol et al. proposed an approach to detect invalid comments while renaming identifiers in source code [12]. The authors created guidelines to link comments and their responsive codes and defined the scope of comments in a project to link identifiers.

While the above works are related to detect inconsistencies between code and comment, there are some other works to measure the code comments quality. Steidl et al. presented a semi automatic approach for quality analysis and assessment of code comments [13]. Their focus was to evaluate comments quality to improve the readability of source codes. They used machine learning technique to classify comments into categories and based on these categories they developed a comment quality model.

Corazza et al. published a benchmark dataset of java method-comment pairs with corresponding coherent values which they inspected manually [5]. Later they investigated if it is possible to predict whether a code-comment pair is coherent or not. They used Vector Space Model to represent a method or comment based on their tf-idf score. Initially they used lexical similarity to measure the coherence value and later they trained a SVM with grid search algorithm to adjust parameters.

Wen et al. presented a large scale study [14] on code comments and found that, code and comments co-evolve over time. Besides, some approaches are proposed to generate natural language summary or comments from source codes [15], [16]. As per our knowledge, none of the approaches use the insight dominating topics to detect if a code and comment pair conveys the same meaning or not.

VI. Threats to Validity

The most important threat is external validity which is related to the software applications considered in the dataset. All the applications in experimented dataset were implemented in Java which could bias the results. For example, Java is more verbose than other programming languages (e.g., C, C++ etc.) and then the developers of the applications in the dataset could have paid inadequate attention on commenting methods.

VII. Conclusion

This paper proposed a new ensemble approach to measure the effectiveness of detecting code comment inconsistencies. In this approach, features are extracted from codes and comments using topic modeling. Afterwards, proposed model fuses the coherence scores obtained by different sources (Random Forests) to provide the probability of inconsistency between a code and comment. This approach was evaluated in a benchmark dataset of java projects and the result was prominent and satisfactory. This approach can also be applied to detect inconsistencies between code comment pairs of other languages as a future work.

References

[1] T. Tenny, “Program readability: Procedures versus comments,” *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1271–1279, 1988.

[2] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, “The effect of modularization and comments on program comprehension,” in *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 1981, pp. 215–223.

[3] B. Fluri, M. Wursch, and 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.

[4] D. Lawrie, D. Binkley, and C. Morrell, “Normalizing source code vocabulary,” in *2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 3–12.

[5] A. Corazza, V. Maggio, and G. Scanniello, “Coherence of comments and method implementations: a dataset and an empirical investigation,” *Software Quality Journal*, vol. 26, no. 2, pp. 751–777, 2018.

[6] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[7] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[9] A. Corazza, V. Maggio, and G. Scanniello, “On the coherence between comments and implementations in source code,” in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2015, pp. 76–83.

[10] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, “/* icomment: Bugs or bad comments?*/,” in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 145–158.

[11] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, “@ tcomment: Testing javadoc comments to detect comment-code inconsistencies,” in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 260–269.

[12] I. K. Ratol and M. P. Robillard, “Detecting fragile comments,” in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 112–122.

[13] D. Steidl, B. Hummel, and E. Juergens, “Quality analysis of source code comments,” in *2013 21st International Conference on Program Comprehension (ICPC)*. Ieee, 2013, pp. 83–92.

[14] F. Wen, C. Nagy, G. Bavota, and M. Lanza, “A large-scale empirical study on code-comment inconsistencies,” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 53–64.

[15] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, “Summarizing source code using a neural attention model,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 2073–2083.

[16] E. Wong, T. Liu, and L. Tan, “Clocom: Mining existing source code for automatic comment generation,” in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 380–389.