# Learning based and Context Aware Non-Informative Comment Detection

**Conference Paper** · September 2020

DOI: 10.1109/ICSME46990.2020.00115



**7 authors**, including:

Mingwei Liu
Fudan University
**9** PUBLICATIONS **64** CITATIONS
SEE PROFILE

Xin Peng
Fudan University
**118** PUBLICATIONS **1,225** CITATIONS
SEE PROFILE

Chong Wang
Fudan University
**2** PUBLICATIONS **13** CITATIONS
SEE PROFILE

# Learning based and Context Aware Non-Informative Comment Detection

Mingwei Liu*†, Yanjun Yang*†, Xin Peng*†, Chong Wang*†, Chengyuan Zhao*†, Xin Wang*†, Shuangshuang Xing*†
*School of Computer Science, Fudan University, Shanghai, China
†Shanghai Key Laboratory of Data Science, Fudan University, China
{19110240019, 20212010022, pengxin, 18212010030, 19110240027, 18212010029, 18212010042}@fudan.edu.com

***Abstract*—This report introduces the approach that we have designed and implemented for the DeClutter challenge of DocGen2, which detects non-informative code comments. The approach combines both comment based text classification and code context based prediction. Based on the approach, our "fduse" team achieved the best F1 score (0.847) in the competition.**

***Index Terms*—API Documentation, Text Classification, Deep Learning**

## I. Introduction

Code comments like "loop from 1 to N" convey no useful information for the readers. Moreover, they could drift when the corresponding code changes, for example the range of the loop change from $N$ to $N-1$. This kind of non-informative code comments are redundant or even harmful for maintenance. It is thus desired that non-informative code comments can be detected and improved to ensure the code quality. The DeClutter challenge of DocGen2 (the Second Software Documentation Generation Challenge)[1] calls for an automated tool that can identify non-informative code comments. We participated the challenge using the team name "fduse". This report introduces the approach that we have designed and implemented for the challenge and the results that we have achieved based on the data set of the challenge.

Our approach uses a combined prediction model to detect non-informative code comments. It combines both comment based text classification and code context based prediction. Comment based text classification uses a binary text classifier to determine whether a given code comment is informative or not. The text classifier is implemented using BERT (Bidirectional Encoder Representations from Transformers) [1], a pre-trained language model based on a large-scale corpus. Code context based prediction trains an AdaBoost classifier[2] using a set of features defined based on the similarity between code comments and their code context. These two classifiers are combined to produce the final prediction results based on the confidence of the classifiers and comment-code similarity.

## II. Approach

### A. Preprocessing

Among the 1,311 comments in the training set, 934 are annotated as informative and 377 are annotated as non-informative. We analyzed the data and identified some quality issues, and cleaned the data accordingly.

1) **Duplicate Comments**. Some data rows have exactly the same comments. For example, the comments in the data rows from SR201 to SR253 are all "Auto-generated method stub". For duplicated comments, we keep the data row that appears first in the data set and remove all the others. We found that the same comment may be annotated differently. For example, the data row FR761 and the row FR763 have the same comment "css: eye *" but different annotations. For these comments we determine the annotations based on the majority of them. If the annotations cannot be determined using the strategy, we remove all of them.

2) **Missing Comments**. Some data rows have no comments. For example, the data row FR866 is annotated as informative but its comment column is empty. For these rows, we try to complete the comment column by tracing to the code based on the "link_to_information" column. If the target comments do not exist in the project, we remove the rows.

After the above preprocessing, we obtained 1,194 annotated comments (303 non-informative and 891 informative). After that, we further removed special characters (*e.g.,* @, #) in the comments and turned the comments to lowercase.

### B. Comment based Text Classification

Comment based text classification takes a code comment as input and returns the classification result (informative or not). It can be treated as a standard binary text classification problem and supported by existing text classifiers such as FastText [2], TextCNN [3], and BERT [1]). We tried all these three text classifiers using their implementations: FastText[3], TextCNN[4], and BERT[5]. We did not apply extra preprocessing on the comments with the purpose of keeping their original features, *e.g.,* specific sentence patterns, tenses.

Table I shows the performance of different text classifiers with the training set and test set. When testing with the training set, we used five-fold cross validation based on the training data. When testing with the test set, we used the whole training set to train a classifier and submitted its prediction results on

[1]https://dysdoc.github.io/docgen2/index.html
[2]https://scikit-learn.org/stable/index.html
[3]https://github.com/facebookresearch/fastText
[4]https://github.com/yongfengxuemei/NLP/tree/master/CNN_ChineseText-BinaryClassify
[5]https://github.com/google-research/bert

TABLE I
PERFORMANCE OF DIFFERENT TEXT CLASSIFIERS

| Classifier | Training Set | | | Test Set |
|---|---|---|---|---|
| | Precision | Recall | F1 | F1 |
| FastText | 0.722 | 0.690 | 0.702 | 0.777 |
| TextCNN | 0.720 | 0.678 | 0.693 | 0.746 |
| BERT | 0.713 | 0.714 | **0.713** | **0.823** |

the test set to the competition website to get the feedback on F1 score. We can see BERT achieves the best performance with both the training set and test set.

As the training set is small, we have even tried a data augmentation technique called EDA (easy data augmentation) [4], which generates new training data by text mutation (*e.g.,* synonym replacement, random insertion, random swap, and random deletion). We used the EDA implementation[6] to generate new training data with the hyperparameter num_aug set to 1, *i.e.,* for each existing training sample generating a new one with the same annotation. We used the expanded training set to train a BERT model. The new BERT model achieved an F1 score of **0.807** on the test set, which is lower than the corresponding F1 score shown in Table I (*i.e.,* **0.823**). The reason may be that EDA is more suitable for long text which can better tolerate text mutation while code comments are usually very short. We have also tried to combine the three text classifiers using a majority voting strategy. The resulted F1 score on the test set is 0.808, which is lower than that of the BERT model (*i.e.,* **0.823**).

### C. Code Context based Prediction

Code context based prediction is designed based on the following features about code comments and their code context. Currently, we only consider the next line of code of a given comment as its code context. The calculation of similarity features is based on the bags of words of code comments and code context produced by text preprocessing, which includes splitting identifiers by camel case and removing stop words.

1) **Semantic Similarity**. The semantic similarity between a code comment and its code context is the cosine similarity (ranging between -1 and 1) between their sentence vectors, which are produced by averaging their word vectors. The word vectors are obtained using Word2Vec with Spacy[7].

2) **Lexical Similarity**. The semantic similarity between a code comment and its code context is the percentage of the words of the comment that are shared with the code context.

3) **Distance**. The number of lines between a code comment and its code context. Note that there may be multiple lines of comments before a code line and their relationships with the code line may be different.

4) **Comment Length**. The number of words of the comment.

5) **Comment Type**. The type of a code comment is 1 for JavaDoc, 2 for line comment, and 3 for block comment.

We trained an AdaBoost classifier using scikit-learn[8] with the above five features on the training set. We tested the AdaBoost classifier on the test set and the F1 score is 0.777. This result shows the effectiveness of the code context based prediction based on manually defined features.

[6]https://github.com/jasonwei20/eda_nlp

[7]https://spacy.io/

[8]https://scikit-learn.org/stable/index.html

### D. Prediction Model Combination

Our analysis on the results shows that the BERT classifier and the AdaBoost classifier can be complementary. The BERT classifier performs better in most cases, but in some other cases the AdaBoost classifier performs better. We found that the complementarity is relevant to the prediction confidence of the BERT classifier and the comment-code similarity. Based on our findings, we decided to use the following strategy to combine the two classifiers.

The prediction results of the AdaBoost classifier are chosen as the final results if one of the three conditions is met:

1) the prediction confidence of the BERT classifier is between 0.5-0.8, indicating that the classifier is uncertain about the prediction result;

2) the lexical similarity between the comment and its code context is between 0.7-1.0, indicating that the comment may be a simple repetition of the code;

3) the semantic similarity between the comment and its code context is less than zero, indicating that the comment may be irrelevant to the code, *e.g.,* TODO statement or obsolete code that is commented out.

In these three cases, it may be difficult to make the prediction based only on the content of the comment, so the AdaBoost classifier is used. In all the other cases, the prediction results of the BERT classifier are chosen as the final results.

## III. CONCLUSION

For the DeClutter challenge of DocGen2, we designed and implemented a learning based and context aware non-informative comment detection approach. The approach combines both comment based text classification and code context based prediction. Based on the approach, our "fduse" team achieved the best F1 score (0.847) on the private leaderboard. The result confirms the benefit of the combination of comment based text classification and manually defined code context features.


## ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under Grant No. 61972098.


## REFERENCES


[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT 2019*, pp. 4171–4186.

[2] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *ACL*, 2017, pp. 427–431.

[3] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP 2014*, A. Moschitti, B. Pang, and W. Daelemans, Eds., pp. 1746–1751.

[4] J. W. Wei and K. Zou, "EDA: easy data augmentation techniques for boosting performance on text classification tasks," in *EMNLP-IJCNLP 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., pp. 6381–6387.