Detecting and Classifying Self-Admitted of Technical Debt with CNN-BiLSTM

Kuiyu Zhu*, Ming Yin¹ and Yizhen Li¹

¹ School of Software, Northwestern Polytechnical University, Xi’an, Shaanxi, 710129, China

*Corresponding author’s e-mail: zhuky@mail.nwpu.edu.cn

Abstract. Technical debt expresses the use of non-optimal solutions for short-term gains. Self-admitted technical debt is a technical debt that is deliberately introduced and recognized by developers in source code comments, including design debt, requirement debt, and defect debt, etc. Currently, many methods are proposed to detect SATD. However, these methods are limited to the identification of SATD or non-SATD. In this paper, we propose a CNN-BiLSTM method to detect and classify SATD. Through our cross-project experiments on 10 projects, it is shown that our method can not only effectively detect SATD, but also classify design debt, requirement debt, and defect debt in SATD.

1. Introduction

Technical debt was first proposed by Cunningham [1], expressing immature code can benefit in the short term, but it also accumulates debt, and the time spent on wrong code is like repaying interest. Potar and Shihab [2] defined technical debt intentionally introduced by developers as self-admitted technical debt, which is usually expressed in the form of source code comments, such as “Temporary until SimpleListModel is used for all”, “TODO This assumes we only get a SET notification, which isn't a good assumption”. In addition, their research shows that this kind of technical debt may frequently exist in software projects [2]. Over the long term, failure to repay debts in time may interrupt the development process, leading to great hidden dangers in software development and maintenance [3]. Sierra, Shihab and Kamei [4] pointed out that low-quality code, deadline pressure, etc. will cause developers to intentionally introduce technical debt. However, in the actual development process, it is very common to face these factors. Therefore, it is very important to study how to detect these self-admitted technical debts so that they can be paid in time for the maintenance of the whole development process.

Potar and Shihab [2] through exploratory research on four large-scale projects, they manually analyzed the comments extracted, and finally proposed the use of 62 patterns to identify SATD. However, this method not only requires a lot of manpower, but also is difficult to apply to large-scale cross-project identification, because unknown patterns cannot be summarized. Maldonado et al. [5] use NLP technology to automatically detect design and requirement self-admitted technical debt. After their cross-project experiments, this method is significantly better than the pattern-based method. Huang et al. [6] proposed a text mining method to detect SATD in comments by using feature selection and compound classifiers. However, some useful features may be filtered out during feature selection. Especially for the defect debt with very few samples, if the text mining method is used, it may be difficult to train a good classifier due to the lack of sufficient data. Ren et al. [7] proposed a Convolutional Neural Network-based method to detect SATD and non-SATD in code comments. In
addition, they also extracted some key phrases and patterns related to SATD detection to improve the interpretability of the method. Santos et al. [8] combined the long-term short-term memory network and the Word2vec model in order to identify the design and requirement self-admitted technical debt in the comments.

However, most of the current research is limited to the SATD and non-SATD in the detection code comments, and does not classify the existing design debt, requirement debt, and defect debt. Therefore, this paper proposes a CNN-BiLSTM method to detect SATD and classify three kinds of self-admitted technical debt in comments. Firstly, we use the data set provided by Maldonado [5], preprocess the data of 10 projects and then use Word2vec to express the comment text as a matrix, then train a CNN-BiLSTM model, and take turns to evaluate the 10 projects as the test set. According to our experimental results, our method can implement SATD detection and classification.

2. Related works
Many studies have focused on the detection of SATD in source code comments. Potar and Shihab [2] conducted exploratory research on four large open-source projects, and analyzed some examples of SATD to summarize the most likely SATD model. As a result, they summarized a total of 62 patterns and used a pattern-based approach can detect whether the comment is SATD. In addition, they also pointed out that self-admitted technical debt appears more frequently in software projects and the existence of SATD will have a negative impact on software quality. Farias et al. [9] developed a CVM-TD model to detect SATD through contextual vocabulary analysis.

Based on previous studies, Bavota and Russo [10] conducted a study on 159 software projects, aiming to investigate changes in self-admitted technical debt and its relationship with software quality. Maldonado and Shihab [11] used four filtering heuristics to deal with comments that are unlikely to be technical debt, and then by reading a large number of comments, they showed that self-admitted technical debt can be specifically divided into five types, and in the entire software the design debt accounted for the highest proportion of the project. Maldonado et al. [5] used the data of 10 projects to train a maximum entropy classifier to identify design and requirement self-admitted technical debt. Compared with the pattern-based method, this method reduces manual participation and improves detection efficiency. However, training a maximum entropy classifier often requires a lot of data, so for defect debt, this method cannot be detected. Huang et al. [6] proposed an automatic SATD detection method. They first performed text preprocessing on multiple projects, and then used the feature selection method to select the top 10% features with the highest information gain score, and then used these features to Train sub-classifiers to construct a composite classifier, and use the composite classifier to predict the SATD in the comments. Ren et al. [7] proposed a convolutional neural network method to detect SATD. In addition, in order to improve interpretability, they extracted some key words used to indicate SATD, and showed that the use of CNN method can effectively extract comment text feature.

Our work is different from them. We focus on using the CNN-BiLSTM model to achieve SATD classification, aiming to not only complete the identification of SATD and non-SATD, but also classify the design, requirements, and self-admitted of defects in software project comments.

3. Methodology and experiments

3.1. The framework of our approach
Figure 1 shows the overall framework of our approach. Our experiment is divided into two stages: train phase and prediction phase. In the train phase, first we train 9 projects each time and perform text preprocessing, and then use the pre-trained Word2vec word embedding to represent our text comments, then use the convolution and pooling layers in CNN to extract local relevant features, BiLSTM layer extracts context-sensitive semantic information, then uses Softmax for classification.
3.2. Data Preprocessing
The data we use is collated by Maldonado et al. [5], the data is divided into 10 projects, a total of 61,664 source code comments, most of the projects include defect, requirement, design, documentation and test self-admitted technical debt. This data set has been publicly used to advance research on self-admitted of technical debt.

Table I shows the main categories of SATD in our data set and the number of them on each project. We can know that the three types of SATDs total 3,932 by calculation. They accounted for 95.67% of the total SATD. Therefore, we focused on these three SATDs to detect and classify defect debt, design debt, and requirement debt. In addition, there is a serious imbalance in the data. From the perspective of the two categories of SATD and non-SATD, we find that the total amount of non-SATD is 14 times the total amount of SATD. Please note that there are other categories of self-admitted technical debt, such as test debt and document debt, which we did not list because they have very little data and are not enough to train deep learning models.

| Projects  | Defect | Design | Requirement | SATD  | Non-SATD |
|-----------|--------|--------|-------------|-------|----------|
| Ant       | 13     | 95     | 13          | 131   | 3967     |
| ArgoUML   | 127    | 801    | 411         | 1413  | 8039     |
| Columba   | 13     | 126    | 43          | 204   | 6264     |
| EMF       | 8      | 78     | 16          | 104   | 4286     |
| Hibernate | 52     | 355    | 64          | 472   | 2496     |
| JEdit     | 43     | 196    | 14          | 256   | 10066    |
| JFreeChart| 9      | 184    | 15          | 209   | 4199     |
| JMeter    | 22     | 316    | 21          | 374   | 7683     |
| JRuby     | 161    | 343    | 110         | 662   | 4275     |
| SQuirrel  | 24     | 209    | 50          | 285   | 6929     |
| Total     | 472    | 2703   | 757         | 4110  | 58204    |

3.3. Word embedding
Word2vec [12] uses a layer of neural network to map sequence text to low-dimensional space in a distributed manner. Words with similar semantics or words with relatively large correlation can be mapped to similar spaces, which can express text characteristics well. Word2vec trains word vectors according to the appearance relationship between contexts. It has two models Skip-gram and CBOW, as shown in Figure 2. Among them, Skip-gram predicts the context based on the target word, CBOW predicts the target word based on the context, and finally uses part of the model’s parameters as the word vector.
3.4. Training the CNN-BiLSTM Model
We use a convolutional layer to extract features of different sizes. In order to better extract features, multiple different types of kernels need to be used at the same time, and there can be multiple kernels of each size. Maximum pooling layer, this layer is to splice the comment text features obtained after convolution. We use the max-pooling layer to extract the n-gram features extracted by convolution to extract the features with the most activation. BiLSTM layer, BiLSTM is a combination of forward LSTM and reverse LSTM. Figure 3 shows the structure of the BiLSTM network.

We can see that the Forward layer and the Backward layer are jointly connected to the output layer, which contains the shared weights w1-w6. In the Forward layer, the forward calculation is performed from time 1 to time t, and the output of the forward hidden layer at each time is obtained and saved. In the Backward layer, the backward calculation is performed from time t to time 1, and the output of the backward hidden layer at each time is obtained and saved. Finally, at each moment, the final output is obtained by combining the output results of the Forward layer and the Backward layer at the corresponding time.

3.5. Detecting and classifying self-admitted technical debt
We have separately tested and classified self-admitted technical debt. The detection is SATD and non-SATD, and the classification is to specifically divide SATD into defect, design, and requirement self-admitted technical debts. It is worth noting that we are conducting cross-project experiments, that is, we train 9 projects and test 1 project each time.

4. Results
In our experiment, we conducted two-class detection of SATD and non-SATD. At the same time, we also conducted three classifications of defect, design, and self-admitted technical debt. So, our experimental results are divided into two parts, the first part detects the results of SATD, and the second part classifies the results of SATD. Since our data set is unbalanced, it is unreasonable to use the indicator Accuracy to evaluate our method, because the high accuracy of non-SATD samples will mask the low accuracy of SATD samples. We use F1 to evaluate.

4.1. The results of detection of SATD
The following are the four baseline methods that we use for SATD detection.
• Patterns: The pattern-based method is to read and analyze a large number of source code comments, and summarize 62 patterns that indicate self-recognition of technical debt [2].

• NLP: Maldonado et al. [5] proposed to use training data to train the maximum entropy classifier to achieve the automatic recognition design and demand SATD.

• TM: This is a method proposed by Huang et al. [6] to automatically detect SATD using an integrated text mining method.

Table II shows the F1 score results of our method and the other four baseline methods in cross-project experiments on 10 projects. The average F1 of our proposed CNN-BiLSTM method on 10 projects reached 0.695, which was significantly higher than the 0.664 of the CNN method and increased by 4.67%. It is worth noting that the TM method has excellent performance on JFreeChart and EMF projects, which is significantly better than other methods. It shows that the combination of feature selection and compound classifiers used in the TM method is more efficient in some scenarios. But there are still limitations.

| Projects  | Patterns | NLP  | TM  | CNN  | CNN-BiLSTM |
|-----------|----------|------|-----|------|-------------|
| Ant       | 0.132    | 0.429| 0.388| 0.564| 0.520       |
| ArgoUML   | 0.054    | 0.802| 0.884| 0.832| 0.856       |
| Columba   | 0.129    | 0.682| 0.829| 0.853| 0.847       |
| EMF       | 0.103    | 0.574| 0.549| 0.510| 0.550       |
| Hibernate | 0.133    | 0.587| 0.819| 0.756| 0.821       |
| JEdit     | 0.304    | 0.459| 0.397| 0.408| 0.549       |
| JFreeChart| 0.093    | 0.5   | 0.638| 0.601| 0.558       |
| JMeter    | 0.102    | 0.557| 0.841| 0.736| 0.806       |
| JRuby     | 0.073    | 0.584| 0.674| 0.872| 0.877       |
| SQuirrel  | 0.123    | 0.435| 0.768| 0.503| 0.566       |
| Avg.      | 0.125    | 0.561| 0.679| 0.664| 0.695       |

4.2. The results of classification of SATD

Table III shows the Marco-F1 results of SATD multi-classification of our method and common deep learning models on ten projects. The average Marco-F1 of our method on 10 projects is 0.451, which is 10.81%, 6.62% and 6.37% higher than CNN, LSTM, and BiLSTM, respectively. In addition, we can also see that the methods have achieved the minimum value on the EMF project. According to our investigation on this project, we found that the data of the EMF project is very unbalanced, which brings great difficulties to classification. Even so, our method is still 10.81% better than CNN.

| Projects  | CNN   | LSTM  | BiLSTM | CNN-BiLSTM |
|-----------|-------|-------|--------|-------------|
| Ant       | 0.424 | 0.383 | 0.384  | 0.458       |
| ArgoUML   | 0.404 | 0.422 | 0.408  | 0.427       |
| Columba   | 0.486 | 0.438 | 0.479  | 0.493       |
| EMF       | 0.329 | 0.331 | 0.369  | 0.377       |
| Hibernate | 0.474 | 0.539 | 0.517  | 0.508       |
| JEdit     | 0.323 | 0.380 | 0.372  | 0.430       |
| JFreeChart| 0.362 | 0.385 | 0.412  | 0.469       |
| JMeter    | 0.429 | 0.429 | 0.469  | 0.441       |
| JRuby     | 0.425 | 0.492 | 0.416  | 0.453       |
| SQuirrel  | 0.412 | 0.426 | 0.411  | 0.450       |
| Avg.      | 0.407 | 0.423 | 0.424  | 0.451       |
5. Conclusion
In this article, we propose a CNN-BiLSTM model to detect and classify self-admitted technical debt. Through our experiments, the results show that our method is effective. There has been a significant increase in the detection and classification of self-admitted technical debt. In addition, due to the differences of projects, the results of various methods in each project also have certain differences, which also brings difficulties to classification. However, we also found that for the classification of self-admitted technical debt, the performance of the methods adopted at this stage is limited, which is subject to serious imbalances in data. Therefore, the next work can improve the classification performance by solving the serious data imbalance problem faced by SATD.

References
[1] Cunningham, W. (1993) The WyCash portfolio management system. In: Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), pp. 29-30.
[2] Potdar, A., Shihab, E. (2014) An Exploratory Study on Self-Admitted Technical Debt. In: IEEE International Conference on Software Maintenance and Evolution, Victoria. pp. 91-100.
[3] Zazworka, N., Shaw, M. A., Shull F., Seaman C. B. (2011) Investigating the impact of design debt on software quality. In: Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki. pp. 17-23.
[4] Sierra, G., Shihab, E., Kamei, Y. (2019) A survey of self-admitted technical debt. Journal of Systems and Software., 152: 70-82.
[5] Maldonado, E. D. S., Shihab E., Tsantalis N. (2017) Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. IEEE Trans. Software Eng, 43(11): 1044-1062.
[6] Huang, Q., Shihab, E., Xia, X., Lo, D., Li, Shanping. (2018) Identifying self-admitted technical debt in open source projects using text mining. Empir. Softw. Eng, 23(1): 418-451.
[7] Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., Grundy, J. (2019) Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability. ACM Trans. Softw. Eng. Methodol, 28(3): 15.
[8] Santos, R. M., Santos I. M., Júnior M. C. R., Neto, M. G. D. M. (2020) Long Term-short Memory Neural Networks and Word2vec for Self-admitted Technical Debt Detection. In: Proceedings of the 22nd International Conference on Enterprise Information Systems, Prague. pp: 157-165.
[9] Farias, M. A. D. F., Neto, M. G. D. M., Silva, A. B. D., Spinola, R. O. (2015) A Contextualized Vocabulary Model for identifying technical debt on code comments. In: IEEE International Workshop on Managing Technical Debt, Bremen. pp: 25-32.
[10] Bavota, G., Russo, B. (2016) A large-scale empirical study on self-admitted technical debt. In: Proceedings of the 13th International Conference on Mining Software Repositories, Austin. pp: 315-326.
[11] Maldonado, E. D. S., Shihab, E. (2015) Detecting and quantifying different types of self-admitted technical Debt. In: IEEE International Workshop on Managing Technical Debt, Bremen. pp: 9-15.
[12] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space. In: International Conference on Learning Representations. Scottsdale.