Co-Training and Multi-Level Semantic Extraction Based Code Debt Detection

Liang Li

Abstract— **Identifying Self-Admitted Technical Debt (SATD) plays an important role in maintaining software stability and improving software quality; SATDs need to be repaid in time, otherwise they may cause serious vulnerabilities or even crash the software. Identifying SATDs from large project code is a costly task, and although existing methods can detect SATDs, and researchers have identified design debt and requirements debt, there is still a lack of methods to achieve multi-label classification of SATDs. In this paper, we propose a CoTCapNet model based on a deep generative model and capsule network, for both recurrent neural networks and convolutional neural networks have the problems of insufficient textual feature extraction and easy to cause the loss of important feature information, first, we use a text generation model based on CoT co-training to generate new samples by learning the original SATD data, which can increase the number of small and medium SATD samples and reduce the data imbalance, then use graph convolutional neural network to encode syntactic dependency trees, construct multi-head attention to encode dependencies in text sequences, and finally merge with semantic information through capsule network. Experiments on crossitem recognition of 10 items show that our approach is more effective than existing methods such as CNN and text mining. The proposed CoTCapNet method has strong advantages, especially in the case of highly unbalanced data.**

Index Terms— **SATDs identification, Capsule network, Graph convolutional neural network, Syntactic Dependency Tree**

I. INTRODUCTION

Software development teams have a common goal when developing software products, which is to deliver high quality, bug-free software products within a specified (time or money) budget[1]. In order to achieve this goal, the development team needs to allocate development resources and develop a unified programming specification to ensure the quality of the delivered final product, but due to some irresistible factors, such as rapid delivery time and shortened budget, which make it difficult to achieve the current goal, the developer needs to take sub-optimal measures to achieve these goals. When developers continually use sub-optimal solutions to meet current goals, the accumulation of this behavior creates technical debt during the development process. Technical debt is common in software projects. For example, a developer chooses a technical framework to implement a feature in a software module, but that technical framework is deprecated in later releases, and test engineers perform only simple functional tests in the module, forgetting about complexity and stress tests. These technical debts are intentionally or unintentionally introduced into the software project, and in the future, as software versions and maintainers change, these unresolved technical debts are never tracked down and remain in the software program forever.

Technical debt can manifest itself:

Ad hoc solutions: To quickly implement features or fix bugs, developers may use ad hoc solutions instead of full design and implementation.

Future refactoring costs: Technical debt means that at some point in the future, the development team will have to spend additional time and resources refactoring code, improving architecture, or fixing bugs to undo the effects of an ill-conceived solution that was previously adopted.

Increased complexity: Ad hoc solutions can increase the complexity of a system, making it more difficult to maintain and expand in the future.

The key to managing technical debt is to identify it, quantify it, monitor it, and pay it off as early as possible. This may mean building time into the development cycle for refactoring, improving documentation, fixing defects, or adjusting the architecture to ensure the health and maintainability of the system. Neglecting technical debt can lead to reduced development efficiency, lower code quality, and an increased risk of long-term project failure.

Technical debt is not just technical debt, it is like a pile of garbage, if it is not dealt with for a long time, more garbage will be generated around it, so the "broken window effect" will have a great impact on the environment of the future project, and everyone will gradually lose confidence in maintaining the environment. Therefore, when we discuss technical debt, we are not only discussing the technical debt itself, but also the impact of technical debt on the team's confidence in the pursuit of quality and motivation to maintain a clean environment [2].

II. RELATED WORK

Currently, identifying technical debt through code annotations is a hot research topic in technical debt detection. Current research mainly focuses on solving the binary classification problem (i.e., categorizing code annotations as "SATD" or "non-SATD")[3] without considering the maintenance efficiency of different types of SATDs. In fact, identifying different types of technical debt is an important task that can help developers better understand technical debt.Maldonado[1] et al. manually categorized SATDs into

Manuscript received March 10, 2024

Liang Li, School of computer science and technology, Tiangong University, Tianjin, China

different types, such as design debt, requirements debt, defect debt, documentation debt, and test debt. They analyzed that different types of SATDs can lead to different unexpected behaviors[4] that need to be handled by different developers. In another empirical study [9], Maldonado and Shihab stated that identifying different types of technical debt can help developers to better understand technical debt and is complementary to existing research related to technical debt detection. Thus, identifying different types of technical debt is an important addition to the existing research. However, there are challenges in identifying these three types of technical debt. First, through a survey of code annotations, we found that the number of SATD annotations is less than the number in the overall framework of our approach. NonSATD annotations in projects. The number of different types of SATD annotations in the same project also varies greatly. For example, in the JFreeChart[5] project, the number of design debt comments is about 20 times higher than the number of defect debt comments and about 15 times higher than the number of realization debt comments. Second, different types of technical debt may share some of the same characteristics. Learning useful semantic information for different types of technical debt comments is difficult.

Deep generative models (e.g., GAN and VAE)[6] are effective for data augmentation mainly because these models learn the underlying structure and distribution of the data, thus generating new data with richness and realism. These newly generated data can help improve the performance of downstream tasks for the following main reasons.

Data diversity: Deep generative models can generate new data samples in the learned latent space that not only have similar characteristics to the training data, but also have some diversity. By introducing these new data samples, the diversity of the training data set can be increased, which helps to improve the generalization ability of the model.

Insufficient data situation: In the case of limited training data, data augmentation using deep generative models can effectively expand the data set, which helps the model to better learn the distribution of the data. In this case, the generated data can help the model better adapt to the insufficient data domain.

Introduction of noise and transformations: Data generated by deep generative models (especially GANs) usually come with a certain amount of noise and transformations, which can be considered as perturbations of the original data. These perturbations help train the model to be more robust to noise and transformations, and thus perform better in real-world applications.

Generate task-relevant data: By conditioning the deep generative model (e.g. conditional GAN), it is possible to generate data of a specific class or with specific attributes. This approach allows the generation of targeted data based on the needs of the downstream[7] task, thus improving the performance of the model on a specific task.

In this paper, we propose a capsule generative network text classification model CoTCapNet for the task of selfrecognition technical debt classification. In the preprocessing stage, the original data distribution is learned by a deep generative model with co-training approach to reduce the class imbalance of small samples, and then a graphical convolutional neural network is used as a sub-module to encode the syntactic dependency tree to extract the syntactic information in the text, which is further integrated with the sequence information and dependency fusion to improve the effect of text classification. Through the model classification effect validation experiments, grammar module validation experiments, and module ablation experiments, the effect of this paper's model on text categorization and multi-label text categorization tasks is verified, the function of the grammar module is argued, and the combined effect proves the principle of graph convolutional neural network, capsule network, and multi-head attention. Future work will further optimize the model for other downstream text categorization tasks, such as the technical debt repayment model.

III. METHOD

A. CoT Training

For continuous discrete data with tractable density, such as natural language, generative models are mainly optimized by maximum likelihood estimation (MLE), which inevitably introduces an exposure bias.This leads to the fact that, given a finite set of observations, the parameters of the model that are optimally trained by MLE do not correspond to those that give the best generation quality. Specifically, the model is trained on the input data distribution and tested on a different input distribution (i.e., the learning distribution). This discrepancy means that the model is never exposed to errors during the training phase, and thus errors made along the way will accumulate rapidly during the testing phase.
Because gradient computation requires backpropagation through the output of the generator (i.e., the data), GANs can only model distributions of continuous variables, making them unsuitable for generating discrete sequences such as natural language. Researchers then proposed Sequential Generative Adversarial Networks (SeqGAN), which uses a model-free policy gradient algorithm to optimize the original GAN objective. With SeqGAN, the expected JSD between the current discrete data distribution and the target discrete data distribution is minimized when training is perfect.SeqGAN shows significant improvement in many tasks. Since then, many variants of SeqGAN have been proposed to improve its performance. Nevertheless, according to a previous investigation, SeqGAN is not an ideal algorithm to solve the problem and current algorithms based on it cannot show stable, reliable and observable improvements covering all scenarios.

According to Cooperative Training (CoT)[8], a new algorithm for training likelihood-based generative models on discrete data by directly optimizing the well-estimated Jensen-Shannon scatter, proposed by Lu et al. CoT coordinates the training of a generative module, G, and an auxiliary predictor module, M (called the mediator), which is used to guide G in a cooperative manner.

At each iteration, a number of samples are taken from G and an equal number of samples are randomly selected from the training data[9], and the two are mixed and used to train M.

Since in this case we are only concerned with the likelihood estimation of M with respect to the given samples, our use of MLE in training M does not give rise to the kinds of problems that arise in the general sense. After training M, for a set of samples s from G, the estimate M(s) given by M is used instead of the true value M*(s) to obtain an approximate estimate of the JSD. Convergence to the target distribution is achieved by minimizing this approximate estimate as G is trained. With some derivations, we can give the objective function of each of the two modules in this algorithm [10]:

Generator:

$$\nabla_\theta J_\beta^{0.0}(\theta) = \sum_{t=0}^{n-1} \mathbb{E}_{s_t \sim G_\theta}\left[\nabla_\theta \pi_g(s_t)^\top \left(\log \frac{\pi_m(s_t)}{\pi_g(s_t)}\right)\right] \quad (1)$$

Moderator:

$$J_m(\phi) = \frac{1}{2}\left(\mathbb{E}_{s\sim G_\theta}\left[-\log\left(M_\phi(s)\right)\right] + \mathbb{E}_{s\sim p_{data}}\left[-\log\left(M_\phi(s)\right)\right]\right) \quad (2)$$

For CoT, the final optimization problem can be written as:

$$\max_\theta \max_\phi \mathbb{E}_{s\sim p_{data}}\left[\log\left(M_\phi(s)\right)\right] + \mathbb{E}_{s\sim G_\theta}\left[\log\left(M_\phi(s)\right)\right] \quad (3)$$

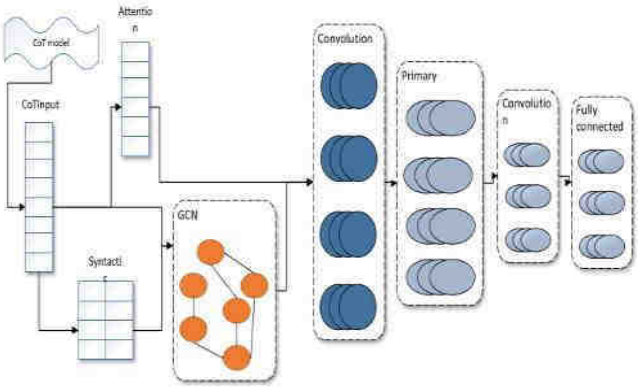
Advantages over previous methods CoT have several practical advantages over previous methods, including MLE, scheduled sampling (SS), and adversarial methods such as SeqGAN. First, although the goal of both CoT and GAN is to optimize the estimated JSD, CoT is more stable than GAN. This is because the two modules, i.e., the generator and the mediator, have similar tasks, i.e., they deal with the same data distributions[11] as the generative and predictive models, respectively.The superiority of CoT over incoherent methods such as predetermined sampling is reliable because CoT has a systematic theoretical explanation for its behavior[12]. CoT is less computationally expensive than methods such as SeqGAN, which require pre-training to reduce variance. In particular, the computational complexity of CoT is the same as that of MLE under the recommended settings.

Moreover, CoT works independently. In fact, it does not require pre-training of the model by traditional methods such as MLE. This is an important property of unsupervised learning algorithms for continuous discrete data, as it eliminates the need to use supervised approximation to reduce variance or complex smoothing.

B. Capsule

The following classification model consists of four modules, as shown in Figure 1.

Fig. 1. Capnet model structure diagram



Data preprocessing module: the preprocessing process consists of data filtering, altered record deletion, tokenization, deactivated word removal, and word shape reduction.

Attention module: it consists of an attention layer that uses multiple attention heads[13]. It encodes the dependencies between words and important word information in a text sequence to form a textual representation.

Grammar module: consists of GCN[14]. It encodes the grammatical dependency tree and extracts the grammatical information in the text to form the textual representation.

Capsule network module: it is a 5-layer capsule network. Based on the text representation output from the Attention Module and the Grammar Module, it further extracts the text semantic and structural information to categorize the text.

IV. EXPERIMENTAL RESULTS

A. *Dataset*

The dataset is derived from a review of 10 projects compiled by Maldonado, including Ant, ArgoUML, Columba, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and Squirrel[15].These data are publicly available and accessible to researchers. Extreme imbalances are observed in the data. For example, in the case of deficient debt, there are 472 deficient debts across the 10 programs, compared to 58,204 non-SATDs, which is approximately 125 times the number of deficient debts. Only 6.50% of the code comments are SATD comments, and the number of non-SATD comments is about 14.38 times the number of SATD comments. In addition, 11.84%, 67.81%, and 18.99% of the technical debts are defect debts, design debts, and realization debts, while only 2.13% and 1.4% of the technical debts are test debts and documentation debts. The number of comments belonging to different types of SATD is highly unbalanced compared to non-SATD comments. This extreme data imbalance can make categorization very difficult. In addition, they categorized self-identified technical debt into five categories, including design debt, defect debt, documentation debt, requirements debt, and test debt.

B. *Evaluation Indicators*

In this paper, we use four commonly used metrics, i.e., precision, recall, and F-measure, to measure the performance of the method. If the predicted category matches the true category, it is a correct classification result, such as true positive (TP) and true negative (TN). Similarly, if there is a mismatch, it is a misclassification result, such as false positive (FP) and false negative (FN)[16]. TP means that the predicted result belongs to SATD and the true result belongs to SATD. TN means that the predicted result does not belong to SATD and the final true result does not belong to SATD. FP means that the predicted result is not SATD, but the final true result is SATD.

FN indicates that the predicted result belongs to SATD, but the real result does not belong to SATD. different classifiers have different experimental results. Precision indicates the proportion of samples predicted as positive by the model in which the true result is also positive, as shown in equation (4).

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

Recall represents the ratio of samples with positive model predictions to those with positive predictions, as shown in equation (5).

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

Since the checking accuracy and the checking rate cannot fully evaluate the performance, F-Measure is supplemented and introduced. as shown in Eq. From equation (6), it can be seen that the reconciled mean value of the check rate and the check rate tends to be close to the smaller value, so a higher value of F-Measure can indicate that both the check rate and the check rate are higher.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (6)$$

C. *Experiment Result*

The experimental parameters of our work are as follows. A 300-dimensional word2vec word vector is input to the model, the attention module uses two attention heads, the first layer of the capsule network module uses 32 convolutional filters with a window size of 3, the second layer uses 32 transformation matrices with 16-dimensional capsule vectors, and the third layer uses 16 transformation matrices with a window size of 3. The last layer uses 9 capsule vectors to represent the 9 categories. For model training, a mini-batch with a batch size of 25 is used, the training batch is controlled to be 20, and the learning rate is set to 0.001. For model testing, for single-label classification tasks, the category label corresponding to the capsule vector with the largest module length is taken. For multi-label classification tasks, the category label corresponding to the capsule vector with a module length greater than 0.5 is taken

Table.1. Precision against other models

Projects	CNN	GRU	SGRU	BiLSTM	ours
Ant	0.571	0.33	0.429	0.416	0.51
ArgUML	0.42	0.4	0.457	0.51	0.63
Columba	0.7	0.75	0.79	0.66	0.83
EMF	0.4	0.85	1.0	0.541	1.0
Jedit	0.571	0.575	0.583	0.4	0.699
Avg	0.563	0.55	0.692	0.61	0.71
JRuby	0.63	0.69	0.904	0.718	0.85
JMeter	0.65	0.655	0.67	0.674	0.742
Squirrel	0.5	0.53	0.529	0.55	0.57
Hbernate	0.878	0.7	0.854	0.76	0.869

Table.2.Recall against other models

Projects	CNN	GRU	SGRU	BiLSTM	ours
Ant	0.308	0.23	0.462	0.4	0.51
ArgUML	0.638	0.64	0.669	0.51	0.63
Columba	0.538	0.57	0.692	0.66	0.86
EMF	0.25	0.23	0.25	0.241	0.27
Jedit	0.093	0.07	0.163	0.04	0.199
Avg	0.348	0.31	0.348	0.361	0.37
JRuby	0.056	0.09	0.584	0.318	0.61
JMeter	0.5	0.49	0.545	0.14	0.53
Squirrel	0.208	0.31	0.375	0.35	0.41
Hbernate	0.558	0.68	0.673	0.46	0.687

Table.3. F-Measure against other models

Projects	CNN	GRU	SGRU	BiLSTM	ours
Ant	0.4	0.33	0.44	0.46	0.48
ArgUML	0.506	0.54	0.543	0.51	0.6
Columba	0.609	0.75	0.72	0.6	0.73
EMF	0.308	0.39	0.4	0.341	0.44
Jedit	0.16	0.15	0.255	0.14	0.199
Avg	0.396	0.33	0.557	0.561	0.61
JRuby	0.102	0.17	0.709	0.718	0.785
JMeter	0.55	0.655	0.6	0.64	0.681
Squirrel	0.294	0.53	0.439	0.55	0.57
Hbernate	0.682	0.7	0.753	0.67	0.869

The process of our CoTCapNet method is divided into two parts. The first part is based on oversampled SATD data from CoT. Then, to validate the effectiveness of SCGRU, we compare the F-measure of SCGRU with the other four methods, i.e., CNN, GRU, SCGRU, and BiLSTM, for five types of technical debt (i.e., defect debt, test debt, document debt, design debt, and requirements debt). Tables 1, 2, and 3 show the precision, recall, and F-measures of the five methods for identifying defect debt, respectively. The Fmeasure of our method outperforms the other four methods on 10 items and improves significantly over Columba. Our method outperforms the other methods on several items. We can see that CNNs have a hard time detecting test debts. Most importantly, none of the six methods can detect across items on JFreeChart and Squirrel due to lack of training data. However, our method can successfully detect them. For document debt, the F metric of our method significantly outperforms the other methods on ArgoUML and Hibernate, and the results on JMeter, JRuby, and Squirrel improve with both GRU and SCGRU.

V. CONCLUSION

In this paper, we propose an approach called CoTCapNet for the identification of multiple classes of SATDs. We use a CoT-based deep text generation model to deal with extreme data imbalance, and use a combination of an attention mechanism and a capsule network model to identify multiple classes of SATDs. Five types of technical debt are identified, namely, defect debt, test debt, document debt, design debt, and requirements debt. Cross-project experiments show that our approach significantly outperforms existing methods, especially when the data is extremely imbalanced. Our proposed method provides new ideas for the practice of SATD identification. When performing SATD prediction for software projects, we effectively solve some problems where SATD cannot be identified due to extreme data imbalance or lack of sufficient data. In addition, the proposed method successfully identifies multiple categories of SATDs and helps to achieve accurate debt localization. In addition, the identification of multiple types of SATDs helps in conducting other studies on SATDs, such as SATD removal and management. If there is an extreme lack of technical debt for a particular item, such as fewer than five training samples, even if we sample some items for text generation, the generated samples may not be sufficient for the classifier to learn features due to the small number of samples. Technical data volume. In the future, we would like to obtain more data to supplement our learning, generate more diverse and valuable

technical debt samples, and provide more opportunities to validate the effectiveness of our method.

REFERENCES

1) E. da S. Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt", IEEE Trans. Softw. Eng., vol. 43, no. 11, pp. 1044–1062, Nov. 2017.
2) R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in Proc. 20th Int. Conf. Softw. Maintenance, Chicago, IL, USA, 2004, pp. 350–359.
3) C. Nacimento, S. Matalonga, and J. C. R. Hauck, "Identifying technical debt cost factors in reflection activities of an agile projects," in Proc. XL Latin Amer. Comput. Conf., Montevideo, Uruguay, 2014, pp. 1– 11.
4) K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," Comput. Linguistics, vol. 16, no. 1, pp. 22–29, 1990.
5) W. Liu, S. Wang, X. Chen, and H. Jiang, "Predicting the severity of bug reports based on feature selection," Int. J. Softw. Eng. Knowl. Eng., vol. 28, no. 4, pp. 537–558, 2018.
6) Information explaining API types using text classification," in Proc. 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, 2015, vol. 1, pp. 869–879
7) X. Chen et al., "A systemic framework for crowdsourced test report quality assessment," Empirical Softw. Eng., vol. 25, no. 2, pp. 1382– 1418, 2020.
8) N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou, "JDeodorant: Identification and removal of type-checking bad smells," in Proc. 12th Eur. Conf. Softw. Maintenance Reengineering, Athens, Greece, 2008, pp. 329–331.
9) G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescap, "Multiclassification approaches for classifying mobile app traffic," J. Netw. Comput. Appl., vol. 103, pp. 131–145, 2018.
10) F. Sebastiani, "Machine learning in automated text categorization," ACM Comput. Surv., vol. 34, no. 1, pp. 1–47, 2002.
11) A. Mccallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in Proc. AAAI-98 Workshop Learn. Text Categorization, 1998, pp. 41–48.
12) N. Beringer, "Fast and effective retraining on contrastive vocal characteristics with bidirectional long short-term memory nets," in Proc. 9th Int. Conf. Spoken
13) Alaparthi S, Mishra M. 2020. Bidirectional Encoder Representations from Transformers (BERT): a sentiment analysis odyssey. Available at http://arxiv.org/abs/2007.01127.
14) Bastings J, Titov I, Aziz W, Marcheggiani D, Sima'an K. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Copenhagen, Denmark, 1957–1967.
15) Eriguchi A, Tsuruoka Y, Cho K. 2017. Learning to parse and translate improves neural machine translation. In: 55th Annual Meeting of the Association for Computational Linguistics. Vancouver, Canada, 72– 78.
16) Lewis DD. 1992. An evaluation of phrasal and clustered representations on a text categorization task. In: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieva. Copenhagen, Denmark, 37–50.
17) Marcheggiani D, Titov I. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In: The 2017 Conference on Empirical Methods in Natural Language Processing. 1506–1515.