Deep Learning-Based Self-Admitted Technical Debt Detection Empirical Research

Yubin Qu[1,2,3], Tie Bao[1], Meng Yuan[1], Long Li[4]*

[1] College of Computer Science and Technology, Jilin University, China
[2] School of Information Engineering, Jiangsu College of Engineering and Technology, China
[3] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, China
[4] College of Cyber Security, Jinan University, China
yubinqu@icloud.com, baotie@jlu.edu.cn, wojade@gmail.com, lilong@guet.edu.cn

Abstract

Self-Admitted Technical Debt (SATD) is a workaround for current gains and subsequent software quality in software comments. Some studies have been conducted using NLP-based techniques or CNN-based classifiers. However, there exists a class imbalance problem in different software projects since the software code comments with SATD features are significantly less than those without Non-SATD. Therefore, to design a classification model with the ability of dealing with this class imbalance problem is necessary for SATD detection. We propose an improved loss function based on information entropy. Our proposed function is studied in a variety of application scenarios. Empirical research on 10 JAVA software projects is conducted to show the competitiveness of our new approach. We find our proposed approach can perform significantly better than state-of-the-art baselines.

Keywords: Deep learning, Convolutional neural network, Long short-term memory, Loss function, Class imbalance

1 Introduction

The software development process often requires a balance between quality and speed of development. Technical debt describes a sub-optimal choice in the software development process. This choice needs to be addressed by repayment at a later stage [1]. If we fail to pay off our technical debt in a timely manner, there is a high risk of potential harm to our software products [2-4]. We are more concerned with technical debt introduced at the initiative of the programmer, which is described through code comments, called Self-admitted technical debt (SATD) [5].

The prevalent phenomenon of SATD was first discovered by Potdar et al. The major difference between SATD and other technical debts is that SATD exists only in code comments. For example, the comment "// Experimental/ Testing, will be removed" from the project "apache-ant1.7.0" implies a defect, and we should pay this technical debt. Otherwise there is a high risk that such experimental code will cause unexpected production environment errors. This kind of experimental code is common in the software development process, and timely detection and correction of experimental code is an effective way to improve software quality. In recent years, much previous research has proved that although the ratio of SATD code in projects is not high, it will increase the complexity of the software and risk of software product downtime in production environments [6]. Detecting SATD and devoting more time and manpower to the software testing process can effectively reduce potential software risks.

To solve the problem of SATD detecting, techniques using machine learning have been studied. Some studies resorted to manually detect SATD patterns [5-7]. The biggest shortcomings of pattern recognition-based SATD detection methods are that only part of SATD patterns can be used in subsequent projects. Natural language processing (NLP) technology has been used to detect SATD [2]. Maximum entropy classifiers are trained in the dataset to identify two common types of technical debt, including design and requirement debt. Huang et al. proposed to treat SATD as text, and they introduced a text analyzing-based machine learning approach to detect SATD using feature selection and Naive Bayes Multinominal classifier [8]. Ren et al. proposed that convolutional neural networks could be employed to discover different semantic features of SATD through different convolutional kernels, thus ultimately enabling the detection of SATD [9-10]. Recently, deep learning techniques based on deep neural networks have been successfully applied in a wide range of different fields, including the field of graphic image processing for object recognition, natural language processing, speech recognition, and so on. In particular, convolutional neural networks use filters to extract key information from the local view and obtain the core characteristics of the research objective. Related previous studies can be summarized into the following categories: human-summarized patterns, NLP-based techniques, text-mining techniques, and CNN-based techniques. All these research methods are evaluated on open-source projects. For example, the experiment of text-mining techniques has been conducted in eight different areas of open source software projects [8]; an extensive set of experiments of neural network-based detection method is conducted for 10 open-source projects [10]. For these

*Corresponding Author: Tie Bao; E-mail: baotie@jlu.edu.cn
DOI: 10.53106/160792642023072404015

open-source projects, there exist a class imbalance problem in experimental datasets. A cross-entropy loss function that considers different types of weights is designed to cope with imbalances in data sets using CNN-based techniques [10-11]. The cross-entropy loss function that considers different types of weights was effective because of the feature extraction ability using one-dimension convolution kernels. For solving the class imbalance, the loss function using cross entropy that considers different types of weights can alleviate performance loss. This method can avoid the over-fitting problem and the performance loss problem caused by undersampling [12-14]. However, the classification difficulty of different instances is not considered. As shown in Lin's research [12], although the loss value of a large number of easy-to-classify instances is small, it will also affect the performance of the classification model through accumulation effects. Moreover, deep learning techniques have been extremely successful in several application areas and have achieved classification performance that even surpasses that of humans. When detecting SATD, DL can learn the intrinsic characteristics from labeled instances [10, 15]. Therefore, we propose a novel loss function based on information entropy for SATD using DL. In the novel loss function, instances that are difficult to be classified are given a higher penalty, and instances that are easy to be classified are given a lower penalty.

To be fair, we conducted experiments on 10 open source software projects coded in JAVA covering different domains to validate our proposed approach. The data set of source code comments contains a total of 259,229 comments [2, 10]. For investigating the differences of different technical debts, five different types of technical debts, are researched using our proposed focal loss compared with the maximum entropy classifier. For investigating class imbalance problems, within-project classification, cross-project classification, and performance differences of different neural network structures, two types of instances, SATD or NON-SATD are researched. Precision, recall, and F1-measure are computed. The Wilcoxon Signed-rank Statistical test is used to test if the differences of different metrics are statistically significant at the p-value <0.05. Cohen's d effect size is used to quantify the amount of difference.

We have summarized the contributions of the study as follows:

1. We present a novel loss function based on information entropy using DL for the class imbalance problem on SATD detection.
2. We are the first to investigate the impact of different technical debts using CNN-base classifiers. The experimental results show that our approach outperforms the NLP-based techniques.
3. The source code associated with the SATD project has been contributed to GitHub to facilitate comparative research by other researchers. (https://github.com/qyb156/SATD_EmpricalResearch_DL)

We have structured our paper in the following way. Section II gives a brief overview of the background of Neural Networks, the class imbalance problem, and SATD. Section III describes the proposed loss function in detail, including the design motivation and implementation details of the loss function that considers different types of weights. Section IV gives a brief description of the experimental design we used, in which describes the details of the basis of the experiment in detail, including the subjects, etc. Section V gives a detailed description of the results of our experiments. Section VI explores the experimental design in terms of internal and external threats. Section VII provides an overview of future research directions and offers research insights.

2 Background

In this subsection we would like to give a brief statement of the technical basis of the study on neural networks. Then we discuss the class imbalance problem. Finally, we analyze the related work for SATD.

2.1 Neural Networks

The term deep learning was first used and described in 2006 [16]. Artificial neural networks were regarded as having the ability to learn the essential characteristics from raw data. We learned this ability step by step through a multilayer neural network [17]. Different from feature engineering in traditional machine learning, this learning method was called end-to-end learning. In 2012, Deep Convolutional Neural Networks (CNN) were introduced for ImageNet classification. The results on the test data show that CNNs clearly outperform other methods [18]. Recurrent neural network (RNN) is another widely used neural network. LSTM was firstly proposed in 1997 [19] to avoid gradient vanishing problems. LSTM was good at handling events with long delays and time intervals in time series and is widely used by many scholars conducting research on natural language processing [20]. By introducing the Attention mechanism, the distance between any two positions can be reduced to a constant [21]. BERT (Bidirectional Encoder Representations from Transformers) was firstly proposed in 2018 and it was firstly used in Google [22]. Language representation model and bidirectional transformers are the core of BERT. Due to the great success of DL in image recognition and other fields, both academia and industry devoted great enthusiasm to various application fields. In software engineering, deep learning has gradually been popularized and applied in various research directions, such as requirements engineering [23-24], software design [25-27], software development [28-29], software code clone detection [30], software defect prediction [31-32], etc. However, based on the recent surveys [33], prior to 2015, deep learning was not valued and rarely widely adopted in software engineering. From 2015 to now, the studies of software engineering research based on deep learning have grown steadily. For SATD, some studies were using DL. For example, TextCNN was introduced to interpret the SATD pattern [10]. Attention-based Neural Networks were introduced to SATD detection [15]. They introduced word embedding techniques to identify SATD only in the language model construction phase [34].

2.2 Class Imbalance

Class imbalance exists in various fields. Though the ratio of the class with fewer data is low, the class with fewer data

had greater influence. For example, when checking lung cancer, most people tend to be healthy, but few people with potential lung cancer should be considered more. In software engineering, there is also the class imbalance for the SATD data set.

There are several traditional approaches for class imbalance [35], (1) oversampling methods. This method uses random sampling repeatedly for the minority class. For DL, this oversampling method may slow down the training speed and cause an overfitting problem. (2) undersampling method. For SATD, this method may exacerbate the problem of insufficient data and drop valuable instances. (3) cost-sensitive method. Classification models can be reformed, such as cost sensitive learning framework, cost sensitive data distribution method with Adaptive Boosting, cost-sensitive neural network [36]. We have designed different types of weights in our loss function to address the problems posed by different case distributions.

2.3 Self-Admitted Technical Debt

Cunningham firstly introduced technical debt to describe a scenario in which short-term metrics such as quality of software development and speed of software development delivery need to be considered together [1, 8]. Previous studies showed that technical debt was inevitable, and if it cannot be dealt with in time, it would reduce product quality and increase system risk [3, 6]. However, technical debt is not explicitly present in the code comments. In previous studies of technical debt, the static analysis of code has been studied in depth, yet the potential technical debt in code annotations has not been considered [37-38]. Recently Self-Admitted Technical Debt was proposed [5]. This type of technical debt is introduced by programmers on their own initiative, depending on the needs of the software project, etc. Its purpose is to provide sub-optimal technical solutions for the current software code, etc., and these codes may be optimized through software refactoring. Although the ratio of SATD in the entire software project is not very high, its impact should not be underestimated. Because rule-based detection could not be used for code comments due to its natural non-structural characteristics, source Code comments with SATD cannot be automatically detected by computers.

Based on the recent survey [39], six different approaches at the file level were introduced to identify SATD. These methods for detecting SATD can be broadly divided into two different types: (1) pattern-based approaches for textual patterns in comments; (2) machine learning-based approaches. In previous studies, SATD detection mostly used text mining methods [2, 8]. Software engineering has also made extensive use of deep learning techniques and has made extensive progress [33].

3 Our Proposed Approach

In this section, we briefly describe the SATD detection process using DL. Figure 1 shows SATD detection process in our study. Our proposed loss function based on information entropy is described in detail.

3.1 SATD Detection Process using DL

As shown in Figure 1, there are three main phases: collecting training data sets from open-source projects, optimizing loss function for DL, and SATD prediction. First of all, the code comment corpus was collected from 10 open-source projects. For different projects, there is a class imbalance problem [15]. In prior studies, the performance was evaluated in two aspects: within-project aspect, cross-project aspect [44]. For the within-project aspect, 10-fold cross-validation was performed to assess the classification performance. For the cross-project aspect, the experiments were conducted in ten rounds. In each round nine projects were used to train DL models and the remaining project was used to test their model. There is an obvious class imbalance problem in every project. The ratio of SATD ranges from 2.36% to 15.90%. The count of SATD ranges from 104 to 1,413. For example, In the case of ten fold cross validation, when 90% of the data is used as the training dataset, then the remaining 10% of the dataset is used to test the performance of the model. For the project EMF, there may be about 94 (104×0.9) instances used to train the DL model. This means that there will be a small number of instances used to train DL models. As we all know, the larger-scale data sets can help train the model. Therefore, an intuitive approach by merging multiple open-source data sets to a whole data set can be adopted. The intuitive basis for this approach is that in different projects, code comments have potentially consistent semantic features.

Secondly, the loss function was optimized in different neural networks. As shown in Figure 1, neural networks using specially designed loss function were validated in the validation data set to determine the best hyper-parameters.

CNN was adopted in this research using the same network described by Ren et al. [10]. This network was inspired by Kim's TextCNN [9]. Semantic analysis of code annotations by convolutional neural networks is obtained by operating on distributed vector representations of the input by different convolutional kernels. The final output is obtained through a number of different network layers, and based on the output values it is determined whether the code annotations contain SATD. Hyperparameters of our CNN network are the same as Ren's CNN since there were the best hyperparameters on 10 open-source projects. Word embeddings were used as word representations since the comments' rich semantic and syntactic features can be captured. To be consistent with GloVe [40], we set the dimension of the SATD vector as 200. Kim's CNN implementation and Ren's CNN are based on TensorFlow. Based on the same CNN structure, we reproduced our TextCNN using PyTorch. There are a number of key deep neural network hyperparameters to choose, including the dimensionality of the word embeddings for code annotations and the size and number of convolutional kernels for semantic analysis, and so on. Glove was used in our TextCNN and the dimension of the SATD vector is set to 200, which is different form Ren's word embedding dimension (300). Other hyperparameters discribed before in out TextCNN are same as Ren's TextCNN. The difference between our TextCNN and other architectures is that we devise a novel loss function to solve the class imbalance problem in the code annotation dataset. In addition to the

hyperparameter tuning, the termination of training CNN in time is a common means to avoid the over-fitting problem. Our designed model has excellent scalability and can be extended to model BERT at any time. In our experiment, the lstm model is based on PyTorch using neural network modules. In order to be consistent with TextCNN, GloVe is used to embed code comments and the word embedding dimension is 200. Similarly, in the training process of our lstm, we observed the trend of loss function reduction in the validation data set. When we found that the loss in the training data set of ten consecutive batches no longer decreased, we terminated lstm training and saved the network structure." In our experiment, the BERT pretrained model based on PyTorch is used. The pytorch_model module is downloaded locally and loaded using PyTorch. To be fair, the default parameters of the BERT pretrained model are used. Similarly, in the training process of our Bert, we observed the trend of loss function reduction in the validation data set. When we found that the loss in the training data set of ten consecutive batches no longer decreased, we terminated BERT training and saved the network structure.

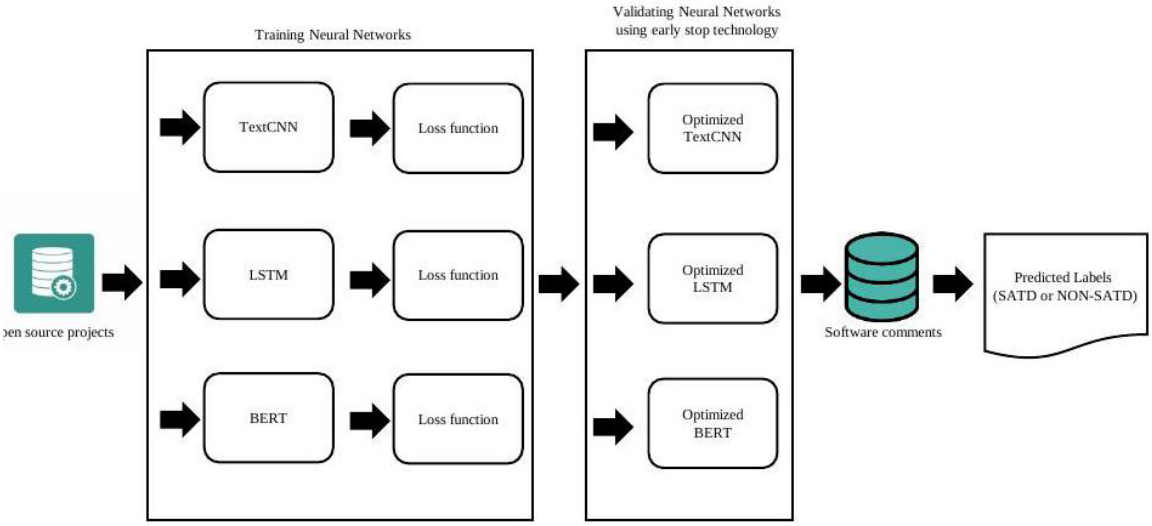

Figure 1. The flow of our proposed SATD detection approach

3.2 Our Improved Focal Loss based on Information Entropy

Cross entropy, from Shannon's information theory, measures the similarity of two different data distributions in terms of their similarity. This method is intuitive and continuous and can be used to optimize the loss value in deep neural networks. The cross entropy loss function was firstly proposed for finding the optimization results for specified domains [41].

y specifies the class of different SATD types, $y \in \{0, 1\}$, and q specifies the estimated probability with label y =1. If the label y =1, the corresponding source comment contains SATD.

$$Loss(y,q) = -1*\log(q) - 0*\log(q). \quad \textbf{(1)}$$

If the label y =0, there is NON-SATD in the corresponding source comment.

$$Loss(y,q) = -0*\log(q) - (1-0)*\log(1-q). \quad \textbf{(2)}$$

We redefine the cross-entropy loss function formally as Equation 3.

$$Loss(y,q) = \begin{cases} -\log(q) & if \ \ y=1 \\ -\log(1-q) & if \ \ y=0 \end{cases}. \quad \textbf{(3)}$$

For convenience, q_t is defined as:

$$q_t = \begin{cases} q & if \ \ y=1 \\ 1-q & if \ \ y=0 \end{cases}. \quad \textbf{(4)}$$

Equation 3 can be rewritten as Equation 5.

$$Loss(y,q) = -\log(q_t). \quad \textbf{(5)}$$

From Lin's research [12], a loss with non-trivial magnitude is incurred for instances, which can be easily classified. These instances are often in the majority category. If there is a serious class imbalance, the loss value of a large number of code annotations containing SATD accumulates and exceeds that of code annotations with a small number of SATD. Probably the neural networks using the loss function shown in Equation (5) tends to predict the class with fewer data as the wrong class.

The cost-sensitive approach introduces the design of weights so that class imbalances can be dealt with, which was used in Ren's study [10]. For example, y =1 corresponds

to the SATD instances with the size m and y =0 corresponds to the NONSATD instances with the size n. Equation 3 can be rewritten as Equation (6)

$$Loss(y,q)=\begin{cases}-\log(q)\times\dfrac{n}{m+n} & if\ \ y=1\\ -\log(1-q)\times(1-\dfrac{n}{m+n}) & if\ \ y=0\end{cases}. \quad \textbf{(6)}$$

For convenience, ratio t is defined as:

$$ratio_t=\begin{cases}\dfrac{n}{m+n} & if\ \ y=1\\ 1-\dfrac{n}{m+n} & if\ \ y=0\end{cases}. \quad \textbf{(7)}$$

Equation 5 can be rewritten as Equation 8.

$$Loss(y,q)=-\log(q_t)\times ratio_t. \quad \textbf{(8)}$$

The focal loss is proposed for dense detectors [12]. The intuition of this approach is that we want to reduce the weight of a large number of easily classifiable instances so that they have less influence in the loss function optimization process, thereby reversing the influence of a few classes. The Equation (8) can be rewritten as Equation (9).

$$Loss(y,q)=-\log(q_t)\times ratio_t\times(1-q_t)^{\gamma}. \quad \textbf{(9)}$$

This loss function tunes the focus using a modulating factor $(1\text{-}q)^{\gamma}$ with tunable focusing parameter $\gamma \in [0, 5]$. The problem for this kind of sepcified loss is that a hyperparameter is introduced. Hyperparameter optimization has always been a tedious and time-consuming process [42]. Through empirical research, γ = 2 can work best in their experiment for object detection [12].

To solve the problem of hyperparameter optimization of the modulating factor, we propose an improved loss function based on information entropy. Information entropy can be formulated for binary classification as Equation (10).

$$Entropy(p)=\sum_{i=0}^{1}-p_i(x)\log(p_i(x)). \quad \textbf{(10)}$$

$p_i(x)$ indicates the estimated probability of the i-th class. Information entropy can be used to measure the amount of information contained in the current predicted instance. The greater the amount of information, the lower the certainty that the current instance is predicted, and the instance should be considered easy to classify; conversely, an instance with a lower amount of information should be considered hard to classify. To amplify the influence of information entropy on hard-to-classify instances, the influence factor θ is shown in Equation (11).

$$\theta=(2+Entropy(p))^3. \quad \textbf{(11)}$$

In this subsection, we first introduce the motivation of using information entropy to modify the modulating factor. Then we describe the computing process of information entropy. In active learning, uncertainty sampling is considered to be the simplest and most commonly used framework [43-44]. There are some strategies to implement uncertainty sampling, in which the most popular sampling strategy uses entropy as an uncertainty measure [45] shown in Equation (10). Using entropy as an uncertainty measure is investigated in our prior work [46]. The reason that Entropy can be used for uncertainty sampling in active learning or in our approach as shown in Equation (11) shows that this is a metric centered on the amount of information, which can represent the amount of information needed to "encode" a distribution. This method can be directly used for machine learning models based on statistical probabilities [47]. Then in our experiment, one source comment from the test data set is firstly embedded. The output word-embedding result is inputted into our trained TextCNN model. For this instance, $p(x)$ is computed and the Entropy according to Equation (10) is computed. If $p(x)$ is closest to 0.5, the Entropy may be close to 1 and θ can be amplified; If $p(x)$ is closest to 0 or 1, the Entropy may be close to 0 and the θ almost cannot be changed. The process of computing Entropy is shown in Figure 2.

Equation 9 can be rewritten as Equation 12.

$$Loss(y,q)=-\log(q_t)\times ratio_t\times\theta. \quad \textbf{(12)}$$

Compared with Equation 9, some important features are considered in Equation 12. First, the class imbalance problem is considered by using the factor ratio t . This cost sensitive approach, which can deal with the class imbalance problem, is used in previous studies [10, 12]. Second, we are still using cross-entropy to calculate the loss values directly from different data distributions. Third, there is still a modulating factor in Equation 12.

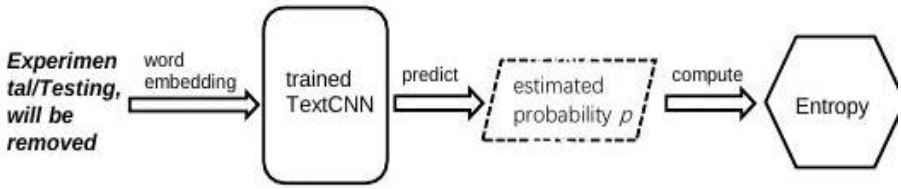

Figure 2. The process of computing entropy

Table 1. Comparison of impact factor amplification

	$p = 0.01$	$p = 0.1$
Impact factor of focal loss ($(1-q_t)^2$)	0.9801	0.81
Impact factor of our loss function (θ)	7.0691	3.588

However, there is an obvious difference that the loss function is designed according to the difficulty of classification. In Equation 9, there is a hyperparameter γ in the modulating factor. In Equation 12, there is not any hyperparameter and the modulating factor, which is replaced by the θ. Information entropy is used to measure the difficulty and the modulating factor is amplified. As shown in Table 1, for example, the estimated probability p is from $p = 0.01$ to $p = 0.1$. If the estimated probability $p = 0.01$ representing an instance with SATD, this indicates that the classification difficulty has increased tenfold compared with the estimated probability $p = 0.1$. For focal loss, there is a little change in magnification from 0.81 to 0.98. But for our proposed loss function, the magnification is from 3.588 to 7.069. Using our approach, the loss function has been significantly enlarged and strengthened for cases that are not easy to classify.

4 Experimental Setup

In this section, we first briefly describe the motivation for our study. Based on this research question, we then describe our entire experimental procedure from various perspectives, including experimental design.

4.1 Research Questions

The detection performance of SATD is our primary concern when conducting research, so we must first address the question of whether our method is optimal. Research questions (RQs) are designed for our empirical study:

RQ1: Whether better classification performance than NLP can be achieved using our designed method?

The motivation of RQ1 is that we compare our proposed approach with NLP techniques between the 10 selected projects for different technical debt types. For design and requirement debt, the maximum entropy classifier can achieve better performance using NLP techniques [2].

RQ2: Can our proposed approach of improved focal loss solve the class imbalance problem for SATD detection?

The motivation of RQ2 is that, in the training data set, the class imbalance problem is widespread. We propose to use an improved focal loss to solve this problem. To verify the effectiveness of our proposed approach, we need to compare it with the normal cross-entropy loss function.

RQ3: Is it more effective using our improved focal loss than weighted loss for within-project classification?

The motivation of RQ3 is that whether our proposed approach can outperform the weighted loss approach. Convolutional neural networks are used to extract features from code comments [10]. For different projects with different SATD characteristics, we need to investigate which models are able to learn the most features from the historical dataset.

4.2 Dataset Description

We conducted our experiments on 10 public open-source projects [2], including Apache Ant, and so on. These datasets are used by other comparison strategies and they allow the performance of the different strategies to be fully evaluated [8, 10]. The dataset description is shown in Table 2. Since the ratio of SATD is from 2.36% to 15.90%, there is an obvious class imbalance in every project. The ten datasets were collected from different types of projects, making these data highly representative. Also because of the different sizes of these software projects, it makes the evaluation of the strategies more credible. Most of these projects are coded in Java or for Java platforms.

Table 2. Statistics of SATD comments in our benchmark gathered from open-source projects

Project	Release	Comments	SATD	% of SATD
Ant	1.7.0	4,137	131	3.17%
ArgoUML	0.34	9,548	1,413	14.80%
JMeter	2.10	8,162	374	4.58%
Columba	1.4	6,478	204	3.15%
EMF	2.4.1	4,401	104	2.36%
Hibernate	3.3.2	2,968	472	15.90%
JEdit	4.2	10,322	256	2.48%
JFreeChart	1.0.19	4423	209	4.73%
JRuby	1.4.0	4897	662	13.52%
SQuirrel	3.0.3	7230	285	3.94%

This data was labelled by Maldonado et al. They followed the following steps for data collection as shown in Figure 3, including pre-processing the data, cleaning the data etc. [2].

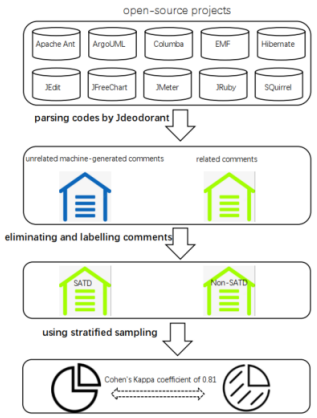

Figure 3. The process of data collection

Firstly, they used a plug-in called JDeodorant to collect raw code comments from projects coded in JAVA, and the total number of data cleaned was 25,923.

Secondly, by introducing several heuristic data cleaning strategies, they obtained a synthesis of 62,566 training data. These heuristic strategies were effective in reducing the workload and improving the quality of the dataset.

Finally, to address the issue of possible bias in the labelled data, they conducted statistical tests by means of stratified sampling. The results of the test showed that the sampling results are highly consistent across different user groups.

4.3 Performance Evaluation Measures

To investigate the impact of different loss functions based on different deep learning models, we consider the following performance measures: precision, recall, and F1-measure. These performance evaluation measures have been used in previous studies for evaluating the results of experiments on the imbalanced datasets [2, 8, 10].

4.4 Experimental Design

The experiment is repeatedly performed m times (m=10). The data is sampled using the sampling method described earlier.

The experimental environment is a workstation computer equipped with Nvidia RTX 2070 GPU, Intel(R) Core i7-10700K CPU and 64GB RAM. The experimental environment runs on the Windows 10 operating system.

5 Experimental Results

In this section, we report and analyze experimental results to answer related research questions.

5.1 Result Analysis for RQ1

Approach. For this research question, we would like to investigate how effectively we can classify self-admitted technical debt comments using our proposed approach. We train the TextCNN using our improved focal loss using the selected 10 projects. Each source comment has a category If there is no SATD in the code comment, it is marked as without-technical-debt; if the code comment contains a defect, it is marked as defect-debt. In keeping with previous research, nine software projects were randomly selected and the data from each project were randomly sampled and finally combined as a training dataset. The remaining one software project was also used as a test dataset for model performance evaluation. We use the final remaining dataset to evaluate the performance of our model by the TextCNN with our improved focal loss. Using comments from 9 out of 10 projects to create the training dataset is reasonable. The training comments from different domains of applications can make convolutional neural networks learn more deep features.

The classifier used for comparison is a maximum entropy classifier. We implement the classifier based on the Stanford Classifier, which is a Java implementation. A series of data items are input to the maximum entropy classifier, and the features of the data items are automatically learned by the maximum entropy classifier. The classifier uses the maximum posterior decision rule and selects the category with the highest probability. In order to fairly compare the performance of the different strategies, the results mentioned in the paper were used directly when evaluating two specific technical debts: design debt and requirement debt [2].

Results - design debt. As shown in Table 3, for the design type of SATD, our proposed method outperforms the NLP-based detection method in most cases in terms of F1 metrics. The range of values for the indicator F1-measure is [0.54, 0.98], while we obtain a mean value of 0.6953 for the F1-measure indicator. In contrast, the natural language processing-based classifier, whose F1-measure takes values in the range [0.47, 0.814], has a mean value of 0.62. The average value has increased by 12.13%. However, we also see that in these two projects: ArgoUML and Hibernate, our classifier performance is weaker than the NLP-based classifier. In addition, we see that on the three test projects (Columba, EMF, and JEdit) with a higher-class imbalance rate, our classifier has a larger performance improvement than the NLP-based classifier. In particular, in the JEdit project, the performance has been improved by 90%. This is probably related to our classifier's ability to focus on class imbalance problems in the training dataset.

Table 3. Comparison of F1-measure between our approach and NLP-based approach for design debt

Project	SATD	Sample count	% of SATD	Our	NLP	Imp. Over NLP
Ant	95	4098	2.32%	0.554	0.517	7.16%
JMeter	316	8057	3.92%	0.78	0.731	6.70%
ArgoUML	801	9452	8.47%	0.659	0.814	−19.04%
Columba	126	6468	1.95%	0.65	0.601	8.15%
EMF	78	4390	1.78%	0.663	0.47	41.06%
Hibernate	355	2968	11.96%	0.658	0.744	−11.56%
JEdit	196	10322	1.90%	0.98	0.509	92.53%
JFreeChart	184	4408	4.17%	0.54	0.492	9.76%
JRuby	343	4897	7.00%	0.8	0.783	2.17%
SQuirrel	209	7215	2.90%	0.669	0.54	23.89%

Results - requirement debt. As shown in Table 4, for the requirement type of SATD, our proposed method outperforms the NLP-based detection method in most cases in terms of F1 metrics. We see that for most of the projects, the F1-measure achieved by our approach is higher than the NLP-based approach. Using our proposed method, the F1-measure metric

has a range of [0.447, 0.998] and a mean value of 0.6577, in contrast to the natural language processing classifier, which has a range of [0.091, 0.804] and a mean value of 0.403. The average value has increased by 63.04%. Compared with the experimental results for the design debt, we see that our classifier has a large performance improvement on all projects (except for the Columba project). In addition, we see that on the three test projects (Ant, JMeter, and JEdit) with a higher-class imbalance rate, our classifier has a larger performance improvement than the NLP-based classifier. In particular, in the JEdit project, the performance has been improved by 997%.

Results - defect debt, documentation debt, and test debt. Table 5, Table 6 and Table 7 present the F1-measure of the two approaches, as well as the improvement achieved by our approach compared to the NLP-based approach. From the experimental data we can intuitively see that our approach outperforms the natural language processing based classifier significantly. From the experimental results, we can see that for the serious class imbalance problems, the performance using NLP-based approach is poor, and the recall rate is almost 0. The classifier we proposed obviously has a very strong generalization and it can deal with class imbalance problem.

Table 4. Comparison of F1-measure between our approach and NLP-based approach for requirement debt

Project	SATD	Sample count	of SATD	Our	NLP	Imp. Over NLP
Ant	13	4098	0.32%	0.784	0.154	409.09%
JMeter	21	8057	0.26%	0.447	0.237	88.61%
ArgoUML	411	9452	4.35%	0.631	0.595	6.05%
Columba	43	6468	0.66%	0.564	0.804	−29.85%
EMF	16	4390	0.36%	0.908	0.381	138.32%
Hibernate	64	2968	2.16%	0.52	0.476	9.24%
JEdit	14	10322	0.14%	0.998	0.091	996.70%
JFreeChart	15	4408	0.34%	0.487	0.321	51.71%
JRuby	110	4897	2.25%	0.5	0.435	14.94%
SQuirrel	50	7215	0.69%	0.738	0.54	36.67%

Table 5. Comparison of F1-measure between our approach and NLP-based approach for defect debt

Project	SATD	Sample count	of SATD	Our	NLP	Imp. Over NLP
Ant	13	4098	0.32%	0.753	0.075	904.00%
JMeter	22	8057	0.27%	0.471	0.065	624.62%
ArgoUML	127	9452	1.34%	0.407	0.12	239.17%
Columba	13	6468	0.20%	0.717	0	+∞
EMF	8	4390	0.18%	0.918	0	+∞
Hibernate	52	2968	1.75%	0.422	0.16	163.75%
JEdit	43	10322	0.42%	0.994	0.09	1004.44%
JFreeChart	9	4408	0.20%	0.667	0	+∞
JRuby	161	4897	3.29%	0.187	0.15	24.67%
SQuirrel	24	7215	0.33%	0.8	0.041	1851.22%

Table 6. Comparison of F1-measure between our approach and NLP-based approach for documentation debt

Project	SATD	Sample count	% of SATD	Our	NLP	Imp. Over NLP
Ant	0	4098	0.00%	0	0	
JMeter	3	8057	0.04%	0.95	0.5	90.00%
ArgoUML	30	9452	0.32%	0.694	0.044	1477.27%
Columba	16	6468	0.25%	0.68	0.027	2418.52%
EMF	0	4390	0.00%	0	0	−
Hibernate	1	2968	0.03%	0.96	0	+∞
JEdit	0	10322	0.00%	0	0	−
JFreeChart	0	4408	0.00%	0	0	−
JRuby	2	4897	0.04%	0.9	0	+∞
SQuirrel	2	7215	0.03%	0.998	0.056	1682.14%

Table 7. Comparison of F1-measure between our approach and NLP-based approach for test debt

Project	SATD	Sample count	% of SATD	Our	NLP	Imp. Over NLP
Ant	10	4098	0.24%	0.836	0.143	484.62%
JMeter	12	8057	0.15%	0.746	0.14	432.86%
ArgoUML	44	9452	0.47%	0.661	0.01	6510.00%
Columba	6	6468	0.09%	0.88	0.29	203.45%
EMF	2	4390	0.05%	0.986	0	+∞
Hibernate	0	2968	0.00%	0	0	–
JEdit	3	10322	0.03%	0.999	0.8	24.88%
JFreeChart	1	4408	0.02%	0.952	0	+∞
JRuby	6	4897	0.12%	0.75	0	+∞
SQuirrel	1	7215	0.01%	0.994	0.5	98.80%

5.2 Result Analysis for RQ2

Approach. Ren's TextCNN model [10] is trained using two kinds of loss functions. While the cross-entropy loss function is still used to calculate the loss values due to different distributions, we have redesigned and implemented an information entropy-based loss function based on our proposed loss function. In order to be consistent with the previous study for subsequent comparisons, the hyperparameters of the deep neural network layers are the same as the model of Ren et al. We reimplement the TextCNN model using Pytorch. Nine of the project comments are used to train the classification model and the remaining project is used for the test data set. Therefore, there are 10 projects to be computed. Precision, recall, and F1-measure are used to evaluate different loss functions.

Results. The comparison results are shown for different loss functions in terms of precision, recall, and F1-measure as Table 8. It is very intuitive to see that our proposed method has various degrees of performance improvement over the original method. The precision value of our approach increases by 9.34% on average; The recall value of our approach increases by 15.78% on average; The F1-measure value of our approach increases by 13.20% on average. The SATD ratio of the Ant project is 3.17%. We can see the precision value increases by 27.62% and the recall value increases by 17.19%. The SATD ratio of the EMF project is 2.36%. We can see the recall value increases by 42.10% and the F1-measure value increases by 16.80%. Therefore, we believe that our approach can effectively deal with the class imbalance problem for SATD detection.

Table 8. Precision, recall, F1-measure of our approach with improved focal loss versus normal loss (NL)

	Precision			Recall			F1-measure		
Project	NL	Our	Improv.	NL	Our	Improv.	NL	Our	Improv.
Ant	0.478	0.61	27.62%	0.64	0.75	17.19	0.547	0.673	23.00%
ArgoUML	0.81	0.85	4.94%	0.912	0.919	0.77	0.858	0.883	2.90%
Columba	0.799	0.85	6.39%	0.799	0.948	18.65	0.799	0.896	12.10%
EMF	0.788	0.799	1.40%	0.553	0.786	42.10	0.65	0.792	21.80%
Hibernate	0.901	0.942	4.55%	0.728	0.831	14.10	0.805	0.883	9.70%
JEdit	0.78	0.99	26.90%	0.6	0.97	61.67	0.678	0.98	44.50%
JFreeChart	0.655	0.695	6.10%	0.75	0.8	6.67	0.699	0.744	6.40%
JMeter	0.806	0.896	11.16%	0.788	0.83	5.33	0.797	0.862	8.20%
JRuby	0.8	0.87	8.75%	0.874	0.91	4.12	0.835	0.89	6.60%
SQuirrel	0.778	0.802	3.08%	0.688	0.745	8.28	0.73	0.772	5.80%
Average	0.7595	0.8304	9.34%	0.7332	0.8489	15.78%	0.74	0.838	13.20%

5.3 Result Analysis for RQ3

Approach. 10 projects belong to different domain fields, and there are different class imbalance ratios in different projects. For investigating different SATD characteristics, each project is studied individually. Therefore, a total of ten projects need to be investigated. For each project, 80% of the data is randomly sampled using a stratified way as training data, 10% of the data is randomly sampled as verification data, and the remaining 10% is used as the test data set. On the test data set, the change values of metrics such as precision, recall, and F1-measure, are calculated to compare our approach with Ren's approach [10]. The Wilcoxon Signed-rank test was introduced into our data testing process. We use the p-value variation to verify that our classification method maintains a high classification performance on different items with different evaluation metrics. Finally,

Cohen's d effect size is used to quantify the amount of difference. For different effect sizes, it represents the diversity between related methods. The difference is considered Small (*d = 0.2*), *medium* (*d = 0.5*), *or large* (*d = 0.8*).

Results. The results of 10 projects are shown in Table 9. Our approach is compared in terms of precision, recall, and F1-measure with Ren's weighted loss approach. Overall, our approach has greatly improved in all performance metrics. The results of the Wilcoxon Signed-rank test show that our method has a strong generalisation capability and is highly robust. For Fl-measure, our approach increases by 16.70% on average. Cohen's d effect size in F1-measure is 0.857, which shows that there is a large difference between the two approaches. For recall, our approach increases by 5.61% on average. Cohen's d effect size in the recall value is 0.475, which shows that there is a big difference between the two approaches. A higher recall rate means that more minority classes with SATD are correctly classified. Therefore, our proposed approach can improve the classifier performance between different individual projects.

Next, we want to make a separate and specific analysis of the performance changes of each project. The project with the highest performance improvement in terms of the F1 measure is Ant. the Fl-measure increases by 67.50%, the recall value increases by 15.87%, and the precision value increases by 137.86%. In terms of F1-measure growth rate, the second-ranked project is the JEdit project, which has a performance increase of 36.60%. The precision value increases by 35.45%. The recall value increases by 5.32%. In terms of F1-measure growth rate, the third-ranked project is EMF. Its F1-measure value has increased by 15%. Similar to the previous two projects, we can see that the performance of the precision and the recall value has been greatly improved. In terms of F1-measure growth rate, the fourth-ranked project is SQuirrel. Its performance increases by 9.30%. From the Table 2, in the JEdit project, the Ant project, the EMF project and the SQuirrel project, the ratio of SATD is 4.73%, 3.17%, 2.36% and 3.94% respectively. Therefore, we believe that our proposed approach can better solve the problem of class imbalance, thereby improving the classification performance of the CNN-based classifier. For the remaining other projects, the performance has been slightly improved. It shows that our proposed approach still has a strong generalization ability for different application fields.

Table 9. Within-project precision, recall, and F1-measure of our approach and weighted loss approach (WL)

	Precision			Recall			F1-measure		
Project	WL	Our	Improv.	WL	Our	Improv.	WL	Our	Improv.
Ant	0.42	0.999	137.86%	0.63	0.73	15.87%	0.504	0.844	67.50%
ArgoUml	0.84	0.91	8.33%	0.95	0.97	2.11%	0.892	0.939	5.30%
Columba	0.71	0.81	14.08%	0.945	0.97	2.65%	0.811	0.883	8.90%
EMF	0.38	0.45	18.42%	0.75	0.81	8%	0.504	0.579	15%
Hibernate	0.8	0.88	10%	0.93	0.98	5.38%	0.86	0.927	7.80%
JEdit	0.55	0.91	65.45%	0.94	0.99	5.32%	0.694	0.948	36.60%
JFreeChart	0.69	0.74	7.25%	0.87	0.91	4.60%	0.77	0.816	6.00%
JMeter	0.84	0.855	1.79%	0.93	0.95	2.15%	0.883	0.9	1.90%
JRuby	0.85	0.928	9.18%	0.91	0.98	7.69%	0.879	0.953	8.40%
SQuirrel	0.79	0.92	16.46%	0.87	0.89	2.30%	0.828	0.905	9.30%
Average	0.687	0.8402	28.88%	0.8725	0.918	5.61%	0.763	0.869	16.70%

6 Threats to Validity

In this section, we explore the factors that may have an impact on the classification performance of our model. Threats to construct validity refer to the hyperparameters of different neural networks and loss functions. Threats to internal validity relate to personal bias in data labeling [2]. The 10 open-source projects are labeled by the author and other independent people. The external threat mainly comes from the application area of the dataset. Although we have used a large number of datasets to cover as many application areas as possible, there are still a large number of application scenarios where our approach cannot be validated. To ensure a sufficient number of samples in this study, we conducted model training on a total of 62,566 samples from 10 open-source projects. These open-source projects cover multiple application areas. We have not validated our model on commercial source projects.

7 Conclusion

SATD is a workaround for current gains and subsequent software quality in software engineering, such as describing potential code defects in software comments. Previous research has adopted pattern recognition, natural language processing, text processing, and other technologies to study SATD detection. In this research, we put forward our consideration from the view point of the class imbalance that exists in the dataset and classifying difficulty for different instances. We propose a new loss function based on information entropy. Then we conduct empirical research on 10 JAVA software projects. The results of empirical studies on data sets from multiple domains show that: (1) The CNN-based classifier using our improved focal loss can outperform the NLP-based classifier for different technical debt types; (2) Our proposed approach can deal with the class imbalance problem; (3) For within-project classification, the CNN-based classifier using our improved focal loss achieves better

performance. In future studies, we will continue to focus on class imbalance in SATD research. Moreover, we will continue to conduct in-depth research on other classification models and other data sets.

Acknowledgments

The authors are particularly grateful to the reviewers who provided review comments. This work was partially supported by Nantong Science and Technology Project (JC2021124), Guangxi Key Laboratory of Trusted Software (kx202046). The Fifth Research Project on Teaching Reform of Vocational Education in Jiangsu Province (ZYB685), Special Foundation for Excellent Young Teachers, Principals Program of Jiangsu Province, and the Qing Lan Project of Jiangsu province.

References

[1] W. Cunningham, The wycash portfolio management system, *ACM SIGPLAN OOPS Messenger*, Vol. 4, No. 2, pp. 29-30, April, 1993.

[2] E. da Silva Maldonado, E. Shihab, N. Tsantalis, Using natural language processing to automatically detect self-admitted technical debt, *IEEE Transactions on Software Engineering,* Vol. 43, No. 11, pp. 1044-1062, November, 2017.

[3] E. Lim, N. Taksande, C. Seaman, A balancing act: What software practitioners have to say about technical debt, *IEEE software*, Vol. 29, No. 6, pp. 22-27, November-December, 2012.

[4] N. Zazworka, M. A. Shaw, F. Shull, C. Seaman, Investigating the impact of design debt on software quality, *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, Hawaii, USA, 2011, pp. 17-23.

[5] A. Potdar, E. Shihab, An exploratory study on self-admitted technical debt, *2014 IEEE International Conference on Software Maintenance and Evolution*, Victoria, British Columbia, Canada, 2014, pp. 91-100.

[6] S. Wehaibi, E. Shihab, L. Guerrouj, Examining the impact of self-admitted technical debt on software quality, *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, Vol. 1, Suita, Osaka, Japan, 2016, pp. 179-188.

[7] G. Bavota, B. Russo, A large-scale empirical study on self-admitted technical debt, *Proceedings of the 13th international conference on mining software repositories*, Austin, Texas, USA, 2016, pp. 315-326.

[8] Q. Huang, E. Shihab, X. Xia, D. Lo, S. Li, Identifying self-admitted technical debt in open source projects using text mining, *Empirical Software Engineering*, Vol. 23, No. 1, pp. 418-451, February, 2018.

[9] Y. Kim, Convolutional neural networks for sentence classification, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, 2014, pp. 1746-1751.

[10] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, J. Grundy, Neural network-based detection of self-admitted technical debt: From performance to explainability, *ACM transactions on software engineering and methodology (TOSEM)*, Vol. 28, No. 3, pp. 1-45, July, 2019.

[11] N. Tran, H. Chen, J. Jiang, J. Bhuyan, J. Ding, Effect of class imbalance on the performance of machine learning-based network intrusion detection, *International Journal of Performability Engineering*, Vol. 17, No. 9, pp. 741-755, September, 2021.

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, Focal loss for dense object detection, *Proceedings of the IEEE international conference on computer vision*, Venice, Italy, 2017, pp. 2980-2988.

[13] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, S. Belongie, Class-balanced loss based on effective number of samples, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Long Beach, CA, USA, 2019, pp. 9268-9277.

[14] K. Dwivedi, R. Lakshmanan, R. Regunathan, K-means under-sampling for hypertension prediction using NHANES DATASET, *International Journal of Performability Engineering*, Vol. 17, No. 8, pp. 733-740, August, 2021.

[15] X. Wang, J. Liu, L. Li, X. Chen, X. Liu, H. Wu, Detecting and explaining self-admitted technical debts with attention-based neural networks, *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Australia, 2020, pp. 871-882.

[16] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, *Neural computation*, Vol. 18, No. 7, pp. 1527-1554, July, 2006.

[17] W. E. Wong, Y. Shi, Y. Qi, R. Golden, Using an RBF neural network to locate program bugs, *Proceedings of the 19th International Symposium on Software Reliability Engineering*, Seattle, Washington, USA, 2008, pp. 27-36.

[18] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Advances in neural information processing systems* 25, Lake Tahoe, Nevada, USA, 2012, pp. 1-9.

[19] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural computation*, Vol. 9, No. 8, pp. 1735-1780, November, 1997.

[20] S. Wang, J. Jiang, *Learning natural language inference with LSTM*, December, 2015, https://arxiv.org/abs/1512.08849.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, *Attention is all you need*, June, 2017, https://arxiv.org/abs/1706.03762.

[22] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, October, 2018, https://arxiv.org/abs/1810.04805 .

[23] A. Al-Hroob, A. T. Imam, R. Al-Heisa, The use of artificial neural networks for extracting actions and actors from requirements document, *Information and Software Technology*, Vol. 101, pp. 1-15, September, 2018.

[24] K. Wiegers, J. Beatty, *Software requirements*, Pearson Education, 2013

[25] F. Pudlitz, F. Brokhausen, A. Vogelsang, Extraction of system states from natural language requirements, *IEEE 27th International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, 2019, pp. 211-222.

[26] H. Thaller, L. Linsbauer, A. Egyed, Feature maps: A comprehensible software representation for design pattern detection, *2019 IEEE 26th international conference on software analysis, evolution and reengineering*, Hangzhou, China, 2019, pp. 207-217.

[27] A. Mahadi, K. Tongay, N. A. Ernst, Cross-dataset design discussion mining, *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*, London, ON, Canada, 2020, pp. 149-160.

[28] M. Bisi, N. K. Goyal, Software development efforts prediction using artificial neural network, *IET Software*, Vol. 10, No. 3, pp. 63-71, June, 2016.

[29] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, S.-W. Lin, A neural model for method name generation from functional description, *IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, Hangzhou, China, 2019, pp. 414-421.

[30] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, K. A. Schneider, Clcdsa: cross language code clone detection using syntactical features and api documentation, *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego California, USA, 2019, pp. 1026-1037.

[31] S. Wang, T. Liu, L. Tan, Automatically learning semantic features for defect prediction, *38th International Conference on Software Engineering (ICSE)*, Austin, Texas, USA, 2016, pp. 297-308.

[32] Y. Qu, X. Chen, L. Li, Cross-version software defect prediction method for relieving class overlap problem, *Journal of Jilin University (Science Edition)*, Vol. 59, No. 2, pp. 372-378, March, 2021.

[33] Y. Yang, X. Xia, D. Lo, J. Grundy, A survey on deep learning for software engineering, November, 2020, https://arxiv.org/abs/2011.14597.

[34] J. Flisar, V. Podgorelec, Identification of self-admitted technical debt using enhanced feature selection based on word embedding, *IEEE Access*, Vol. 7, pp. 106475-106494, August, 2019.

[35] H. He, E. A. Garcia, Learning from imbalanced data, *IEEE Transactions on knowledge and data engineering*, Vol. 21, No. 9, pp. 1263-1284, September, 2009.

[36] M. Kukar, I. Kononenko, Cost-sensitive learning with neural networks, *13th European Conference on Artificial Intelligence*, Brighton, UK, 1998, pp. 445-449.

[37] R. Marinescu, Detection strategies: Metrics-based rules for detecting design flaws, *20th IEEE International Conference on Software Maintenance*, Chicago, IL, USA, 2004, pp. 350-359.

[38] R. Marinescu, G. Ganea, I. Verebi, Incode: Continuous quality assessment and improvement, *14th European Conference on Software Maintenance and Reengineering*, Madrid, Spain, 2010, pp. 274-275.

[39] G. Sierra, E. Shihab, and Y. Kamei, A survey of self-admitted technical debt, *Journal of Systems and Software*, Vol. 152, pp. 70-82, June, 2019.

[40] J. Pennington, R. Socher, C. D. Manning, Glove: Global vectors for word representation, *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, Doha, Qatar, 2014, pp. 1532-1543.

[41] R. Rubinstein, The cross-entropy method for combinatorial and continuous optimization, *Methodology and computing in applied probability*, Vol. 1, No. 2, pp. 127-190, September, 1999.

[42] H. Shaziya, R. Zaheer, Impact of hyperparameters on model development in deep learning, *Proceedings of International Conference on Computational Intelligence and Data Engineering*, Hyderabad, India, 2021, pp. 57-67.

[43] B. Settles, *Active learning literature survey*, 2009, https://burrsettles.com/pub/settles.activelearning.pdf.

[44] D. D. Lewis, W. A. Gale, A sequential algorithm for training text classifiers, *SIGIR'94 Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994, pp. 3-12.

[45] C. E. Shannon, A mathematical theory of communication, *ACM SIGMOBILE mobile computing and communications review*, Vol. 5, No. 1, pp. 3-55, January, 2001.

[46] Y. Qu, X. Chen, R. Chen, X. Ju, J. Guo, Active learning using uncertainty sampling and query-by-committee for software defect prediction, *International Journal of Performability Engineering*, Vol. 15, No. 10, pp. 2701-2708, October, 2019.

[47] F. Li, Y. Qu, X. Chen, L. Li, F. Yang, A sentiment analysis method based on class imbalance learning, *Journal of Jilin University (Science Edition)*, Vol. 59, No. 4, pp. 929-935, July, 2021.

Biographies

Yubin Qu was born in Nanyang, China in 1981. He received the B.S. and M.S. degrees in Computer Science and Technology from Henan Polytechnic University in China in 2004 and 2008. His research interests include software maintenance, software testing, and machine learning.

Tie Bao was born in 1978. He received the Ph.D. degree from Jilin University in 2007. He is now a professor in College of Computer Science and Technology, Jilin University. His research area covers software quality analysis, data analysis, etc.

Meng Yuan is currently a master student in the School of Computer Science and Technology of Jilin University. Her main interest is the application of deep learning and the interpretability of deep learning.

Long Li received his Ph.D. degree from Guilin University of Electronic Technology, Guilin, China in 2018. He is now a lecturer at the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His research interests include cryptographic protocols, privacy-preserving technologies in big data and IoT.