Automatic Classifying Self-Admitted Technical Debt Using N-gram IDF

Supatsara Wattanakriengkrai*, Napat Srisermphoak*, Sahawat Sintoplertchaikul*,
Morakot Choetkiertikul*, Chaiyong Ragkhitwetsagul*, Thanwadee Sunetnanta*, Hideaki Hata†and Kenichi Matsumoto†
*Mahidol University, Thailand
†Nara Institute of Science and Technology, Japan
Email: {supatsara.wat, napat.sri, sahawat.sin}@student.mahidol.ac.th, {morakot.cho, chaiyong.rag, thanwadee.sun}@mahidol.ac.th
{hata, matumoto}@is.naist.jp

***Abstract*—Technical Debt (TD) introduces a quality problem and increases maintenance cost since it may require improvements in the future. Several studies show that it is possible to automatically detect TD from source code comments that developers intentionally created, so-called self-admitted technical debt (SATD). Those studies proposed to use binary classification technique to predict whether a comment shows SATD. However, SATD has different types (e.g. design SATD and requirement SATD). In this paper, we therefore propose an approach using N-gram Inverse Document Frequency (IDF) and employ a multi-class classification technique to build a model that can identify different types of SATD. From the empirical evaluation on 10 open-source projects, our approach outperforms alternative methods (e.g. using BOW and TF-IDF). Our approach also improves the prediction performance over the baseline benchmark by 33%.**

Keywords— Self-Admitted Technical Debt, N-Gram IDF, Multi-class Classification

I. Introduction

Technical Debt (TD) is a situation that software practitioners (e.g. developers) decide to achieve short-term goals (e.g. bug fixing) without an awareness of negative consequences that may take place in the future [1]. TD can highly impact software development cost and software quality (e.g. bug reopening) [2]. Specifically, Self-Admitted Technical Debt (SATD) refers to TD that intentionally stated in source code comments by a developer [3]. A number of studies focuses on supporting software practitioners by developing an approach to detect SATD (e.g. [4], [5]). Maldonado and Shihab [6] have categorized SATD into five different types: design, defect, test, requirement, and documentation. Each type has unique characteristics and impacts on a software project. Thus, it is crucial to identify a type of SATD, rather than whether it is SATD.

There are several studies that proposed models to classify whether comments in source code are SATD. For example, Maldonado et al. [7] proposed an approach using Natural Language Processing (NLP) and Stanford NLP classifier to identify SATD. Besides, Supatsara et al. [8] leverage source code comments using N-gram Inverse Document Frequency (IDF) to improve the performance of SATD prediction model. However, those existing approaches employ binary classification techniques (i.e. one model per one SATD type) which can cause many issues in order to maintain a number of models (e.g. tuning). A recent study [6] also shows that *design* and *requirement* are the most common types of SATD. Hence, in this paper, we focus on developing prediction models to determine whether a source code comment is design SATD, requirement SATD, or non-SATD (i.e. multi-class classification).

The approach in this paper enhances our previous work to develop a multi-class classification model based on textual feature extraction using n-gram IDF. We also employ instance hardness under-sampling to tackle the class imbalance and random forest as a classifier to classify *design* and *requirement* SATD. To the best of our knowledge, this is the first approach for SATD classification (especially for the most common types of SATD, design and requirement SATD). We conduct the empirical evaluation of our approach with baselines on publicly available data. The experimental results show that our approach consistently outperforms two traditional techniques in textual feature extraction: bag-of-words (BOW) and term frequency and inverse document frequency (TF-IDF) and one alternative (i.e., using Support Vector Machine). We then use the model proposed in [7] as a baseline benchmark. Our approach can improve the performance over the baseline in most cases. By analyzing the confusion metrics, our approach improves the prediction of requirement SATD over the baseline method by 33%.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes our approach. We discuss our experimental setting and present our results in Section 4. In Section 5, we present threats to validity and discuss the implications of our work. Section 6 presents our conclusions and future work.

II. Related work

A. Self-Admitted Technical Debt Detection (SATD)

A number of studies have focused on the detection of SATD. For example, Potdar and Shihab [4] pioneered the concept of self-admitted technical debt as a debt intentionally documented in source code comments by developers. They manually analyzed 100K source code comments to develop 62 patterns that indicate the presence of self-admitted technical debt. Maldonado and Shihab [6] leveraged these patterns to categorize self-admitted technical debt into 5 types: design, requirement, defect, test, and documentation debt. Farias et al. [9] utilized code tags and word classes to construct a contextualized vocabulary model for detecting technical debt via source code comments. Huang et al. [10] presented a text mining-based model to automatically identifying SATD comments. To determine SATD at the change-level, Yan et al. [11] made use of source code analysis and change history statistic. Flisar and Podgorelec [12] applied a word embedding technique to the feature selection method in order to increase the correct predictions of SATD. Most recently, Ren et al. [13] proposed a Convolutional Neural Network-based approach for SATD prediction.

B. N-gram IDF

N-gram represents a continuous sequence of n words from a given source text. The advantage of n-grams over isolated words is that an n-gram has more informative and practical. However, using all n-gram phrases will incur massive data and numerous useless features [14]. In order to figure out this problem, we utilize n-gram IDF, a

theoretical extension of inverse document frequency (IDF) proposed by Shirakawa et al. [15]. IDF assesses how important a word is, but cannot handle multi-word terms. N-gram IDF is capable of measure the significance of multiple words to extract useful n-gram phrases. A number of studies leveraged n-gram IDF in the field of text classification. For example, Terdchanakul et al. [16] proposed an n-gram IDF-based approach to automatically identify bug reports. To date, Maipradit et al. [17] presented a sentiment classification method leveraging n-gram IDF and automated machine learning.

C. Under-sampling

In our approach, we cope with the imbalanced dataset by instance hardness under-sampling, a technique to resample data by leveraging a threshold value to eliminate non-useful samples that are frequently misclassified. A previous study [18] demonstrated that instance hardness under-sampling is capable to reduce noisy data in order to improve the quality the of dataset. Recently, Verdikha et al. [19] presented a hate speech classification model based on TF-IDF weighting method and they also utilized instance hardness under-sampling to tackle the class imbalanced problem. Their experiment results showed that the under-sampling method can improve the performance of classifiers.

D. Random Forest

Breiman [20] pioneered random forest (RF), an ensemble machine learning algorithm composes of many decision trees. In random forest, each decision tree works independently with its own results and the final outcome of random forest is the decision with major voting of all decision trees. RF have become a popular technique in various activities such as speech recognition [21], image classification [22], and text categorization [23]. Klema et al. [23] presented an automated approach to classify and analyze fanaticism via emails using the combination of bag-of-words and random forest. Aphinyanaphongs et al. [24] trained a random forest classifier to automatically identify alcohol-related tweets. Liparas et al. [25] combined textual features with visual features to train random forest classifier for news articles categorization.

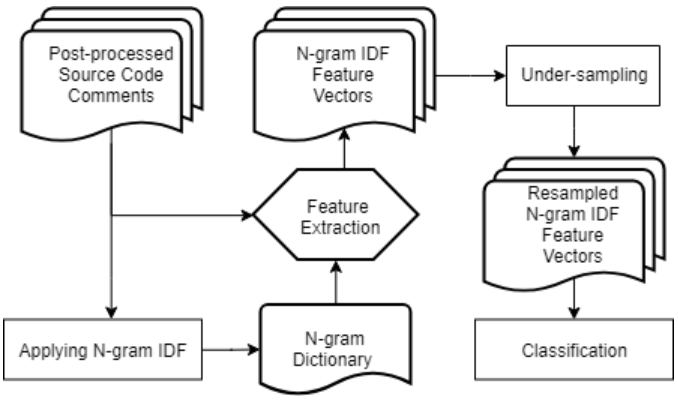

Fig. 1. Overview of our SATD classification approach.

III. Methodology

A. Overview

Figure 1 shows the overall structure of our classification model. To construct the model, we first pre-process source code comments. We then employ an n-gram IDF extraction tool to build our n-gram dictionary. After that, we filter ineffective features out of the dictionary by weight one score. In the next step, we count the raw frequency of each n-gram phase for each source code comment and then represent the frequency values as a series of member vectors. Instance hardness under-sampling is applied to a set of training vectors and the output of this process is resampled training vectors. Finally, we utilize the resampled one to train random forest classifier. We explain more details in the next sections.

B. Applying N-gram IDF

The previous study [8] shows that non-alphanumeric characters (e.g. `?`) yield a benefit to a classifier. We thus encode those with terms (e.g. `questionmark`) that applicable to employ the existing n-gram IDF extraction tool.

We use an n-gram weighting scheme tool namely Ngweight[1]. Note that our dictionary is only produced from a training set. The output after applying n-gram IDF tool is an n-gram dictionary, which contains all valid n-gram phrases and other information (e.g., n-gram id, length of n-gram, global term frequency, document frequency, and document frequency of a set of words composing n-gram). We use this information to find important features in the next step.

C. Feature Extraction

In our experiment, a training set includes approximately 60,000 comments. After applying an n-gram IDF tool to the training data, our n-gram dictionary consists of 60,000 to 70,000 n-gram phrases. On the ground of the large volume of data, we remove ineffective features from the dictionary. We first eliminate n-grams that appear only once in the whole document (global term frequency value equals to one). For each n-gram phrase, we calculate weight one score, a measure of how significant an n-gram phrase is. Noted that the higher weight one score, the more important n-gram is. Weight one score is defined as:

$$Weight1 = log(\frac{|D|}{sdf}) * gtf$$

Where $|D|$ is the number of total documents (in training set), *sdf* is document frequency of a set of words composing n-gram, and *gtf* is global term frequency of the term. We use top ten percents of n-gram phrases which have the highest score to build a new dictionary. The remaining n-grams in the new dictionary are 6,000 to 7,000 terms. We then leverage the filtered n-gram dictionary and pre-processed source code comments to create feature vectors. For each document, we count the raw frequency of each n-gram term exists in the source code comment and we then create a vector element based on the raw frequency values.

D. Under-sampling

Instance hardness under-sampling focuses on the usability of samples for the classification task. (i.e., excluding non-useful samples from the corpus). The under-sampling is available in imbalanced-learn[2] package. To calculate the instance hardness value of each sample, we utilize the same classifier and hyperparameters with the learning process, since both factors directly impact on the calculation of instance hardness [18].

E. Classification

We utilize the implementation of random forest machine learning as provided by scikit-learn[3], and train the classifier by the resampled training data in order to classify target source code comments into design, requirement, non-SATD. To optimize our classification model, we tuned a major hyperparameter of random forest, the number of trees (i.e., n-estimators), by conducting experiments with different values, while we used the default values for other hyperparameters. This will be explained in more details in the evaluation section.

[1]https://github.com/iwnsew/ngweight

[2]https://imbalanced-learn.readthedocs.io/en/stable/index.html

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

IV. Evaluation

We conduct an empirical evaluation in order to answer the following experiment questions.

- **EQ1: How well n-gram IDF deals with the multi-class SATD classification problem?**
  To substantiate that n-gram IDF can work well with the multi-class SATD classification, we performed a comparison between n-gram IDF and popular traditional methods to engineer features representing textual data, TF-IDF and BOW [26]. TF-IDF is a technique to calculate the weight of words in a textual document derived from the product of term frequency (TF) and inverse document frequency (IDF). On the other hand, bag-of-words represents a textual document as a sparse vector of word counts. [27]. For a fair comparison, we applied instance hardness under-sampling and random forest classifier (with its hyperparameters) to all settings.
- **EQ2: Is random forest more suitable for SATD categorization compared with support vector machine?**
  To answer this question, we compared random forest-based model with support vector machine-based model. Support vector machine (SVM) is the most successful linear classifier in the task of text categorization [28] [29] [30] [31]. To build the SVM-based model, we used the features derived from n-gram IDF, but did not apply the under-sampling method for the reason that SVM with instance threshold achieved down-and-out performance. In addition, SVM can handle only a binary decision. We therefore leveraged One vs. One strategy [32] in order to support SVM to address the multi-class classification problem.
- **EQ3: Can our approach outperform the baseline for self-admitted technical debt?**
  Besides our previous work, Maldonado et al. [7] also proposed a SATD classification model based on NLP and maximum entropy classifier. It is the most recent acceptable approach for identifying the most two common types of SATD in source code comments. By contrast, our proposed approach leverages a text mining algorithm, n-gram IDF, under-sampling, and ensemble machine learning algorithm to automatically classify design and requirement SATD. Unfortunately, we cannot directly compare our results with the baseline, for the reason that our approach is a multi-class classification and our dataset contains different SATD classes (i.e., design, requirement, and non-SATD). In order to compare the performance of our approach with them, we first constructed a new dataset containing design and requirement SATD comments labeled as a positive class and non-SATD comments labeled as a negative class. We also prepared sets to validate their approach using leave-one-out cross-project validation method, and we re-implemented their approach following their work to predict SATD comments.
- **EQ4: Can our approach classify design and requirement self-admitted technical debt more accurately than the baseline?**
  Our approach and Maldonado et al.'s work are different problem space (i.e., multi-class and binary classification). Previous work reported that there is a variance between the performance measures for binary classification and multi-class classification [33]. On the account of this problem, we compared the confusion metrics provided by our classification model against the confusion metrics produced by the baseline approach for design and requirement SATD classification. The confusion matrix is simple to understand and we also could know the raw information about predicted and actual classes done by a classification model. For this research question, we generated two new datasets. The first one contains design and without SATD comments. Another one consists of requirement and without SATD comments. Afterwards, we followed their methodology to construct two binary SATD classification models and then collected the confusion metrics of these.

TABLE I
Details of the studied dataset.

Project	Design SATD	Requirement SATD	Non-STAD
Ant	2.30%	0.30%	97.30%
JMeter	3.90%	0.30%	95.80%
ArgoUML	8.70%	4.40%	86.90%
Columba	2.00%	0.70%	97.30%
EMF	1.80%	0.40%	97.80%
Hibernate	12.20%	2.20%	85.60%
JEdit	1.90%	0.10%	98.00%
JFreeChart	4.20%	0.30%	95.50%
JRuby	7.30%	2.30%	90.40%
SQuirrel	2.90%	0.70%	96.40%

A. Dataset

We derived a publicly available data published in Maldonado et al. [7] which consists of 62K Java source code comments (i.e., design, requirement, defect, test, and documentation SATD) from ten open-source projects: Ant, ArgoUML, Columbia, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and Squirrel SQL. We made use of the dataset from the prior study to generate a new dataset composes of three types of SATD: design, requirement, non-SATD comments (approximately 60K) for the empirical evaluation. Table I shows the ratio of SATD comments in the new dataset. Our dataset is evidently imbalanced in which the majority class is non-SATD comments and the minority classes are design and requirement SATD comments.

B. Evaluation Setting

To evaluate the performance of our classification model, we separated the dataset into three sets, training set (8 projects), validation set (1 project), and testing set (1 project). The training set was employed to train the classifier, the validation set was used to tune hyperparameters of a classifier, and the testing set was applied to evaluate the model performance. The evaluation process was then iterated 10 times in which all projects served as the testing set once and one project acted as the validation set – see Section D.

C. Performance Measures

Most of the studies on SATD [7] [10] [12] utilized Precision, Recall, and F1-score in evaluating the accuracy of a SATD classification model. Intuitively, Precision, Recall, and F1-score measure how well a classification model detects SATD comments. Noted that these performance measures do not focus on certain types of SATD in this work. To compute Precision, Recall, and F1-score, we first coped with the confusion matrix. Since the confusion matrix does not take into account a multi-class probabilistic classification, we reduced classified source code comments into two binary classes: SATD and non-SATD using the following formula:

$$C_i = \begin{cases} \textit{SATD}, \text{if } P(i, Des) + P(i, Req) > P(i, Non) \\ \textit{non-SATD}, \text{otherwise} \end{cases} \quad (1)$$

Where C_i is the binary classification of source code comment *i*, and *P(i,Des)*, *P(i,Req)*, an *P(i,Non)* are the probabilities of source code comment *i* predicted as design, requirement, and non-SATD class respectively. If the sum probability of it being predicted as design and requirement SATD is greater than non-SATD, the source code comment will be classified into SATD; otherwise, it will be classified into non-SATD. Then, the values collected in the confusion matrix (i.e., true positive, true negative, false positive, and false negative) are used to calculate Precision, Recall, and F1-score for SATD comments.

Since the confusion matrix does not consider the multi-class probabilistic classification and cost of each SATD class, we utilize a performance measure namely Macro-Averaged Mean Cost-Error

or MMCE proposed in [34]. MMCE assesses how close predicted class probabilities are to the actual classes or MMCE measures how effectively a classification model classify types of SATD. The smaller value is better since it indicates that the distance between them is small. The macro-averaged mean cost-error is defined as:

$$MMCE = \frac{1}{3}\sum_{j=1}^{k}\frac{1}{n_j}\sum_{i=1}^{n}|\hat{y}_i - y_i| \quad (2)$$

Where $\hat{y}_i$ is the probability of class k_{th} of sample i, y_i is the true class of class k_{th} of sample i, k is the number of classes where k = [1,2,3]. There are 3 classes in our classification: non-SATD, design SATD, and requirement SATD respectively. n is the number of samples, and n_j is the number of samples of class k_{th}.

We evaluate the outcomes achieved by classification models using Precision, Recall, F1-score, and MMCE. To compare the performance of two classification models, we test the statistical significance of cost error achieved with them by utilizing a non-parametric statistical test namely Wilcoxon Signed Rank Test [35]. The Wilcoxon test is a safe test that does not assume the underlying data distribution. We set a null hypothesis as "the cost errors presented by a classification model are not significantly different compared to the cost errors provided by another classification model" and set the confidence limit at 0.05 for rejecting the null hypothesis.

The Wilcoxon test indicates whether two classification models are significantly different. On the other hand, to assess the magnitude of the difference that does not demonstrate in the Wilcoxon test, we utilize a non-parametric effect size measure namely the Correlated Samples Case of the Vargha and Delaneys $\hat{A}_{xy}$ Statistic [36]. Also, $\hat{A}_{xy}$ measure does not concern with the underlying data distribution of our dataset. We utilize the cost error provided by two models to calculate the effect size. With the given performance measure, the $\hat{A}_{xy}$ measure assesses the superiority of the classification performance between the two approaches using the following formula:

$$\hat{A}_{XY} = \frac{[\#(X < Y) + (0.5 * \#(X = Y))]}{n} \quad (3)$$

Where *#(X <Y)* is the number of source code comments that the cost error achieved by *X* less than *Y*, *#(X = Y)* is the number of source code comments that the cost error achieved by *X* equal to *Y*, and *n* is the number of source code comments. If the $\hat{A}_{xy}$ measure is more than 0.5, the classification performance of the model *X* is significantly outperform the model *Y*.

D. Hyperparameter Setting

We focused on tuning a major hyperparameter: the number of trees (i.e., n-estimators) of random forest classifier to mitigate the risk of overfitting and improve the performance of our approach, since increasing the number of trees does not increase the probability to encounter the overfitting problem [20]. Oshiro et. al. [37] suggested that the best number of trees is between 64 to 128, and also Khoshgoftaar et. al. [38] recommended to use 100 trees as a default value when the dataset is imbalanced. However, their studies are different from our work (e.g., medical, image processing and transaction). Therefore, we minimized the risk by setting the number of trees in the range of suggestions. To do so, we increased the number of trees from 50 to 350 in which added by 50, while other hyperparameters are fixed. We then observed the F1-score measure to determine the best hyperparameter of the classifier. Noted that the tuning process was done using the validation set.

To select the validation set, we settled projects which have the highest ratio of the positive classes (i.e., the sum of design and requirement SATD comments). As demonstrated in figure 2, the gap between negative and positive classes in the Hibernate project is the smallest, followed by the ArgoUML project. We therefore set the Hibernate project as the validation set, but when the Hibernate project was utilized to evaluate our approach, the ArgoUML project took over the role instead.

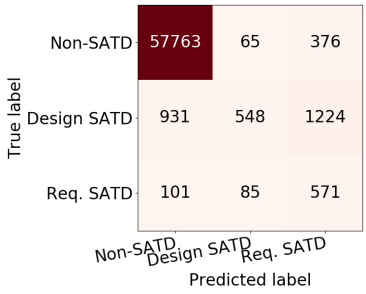

Fig. 2. Confusion matrix of our approach for requirement and design SATD classification.

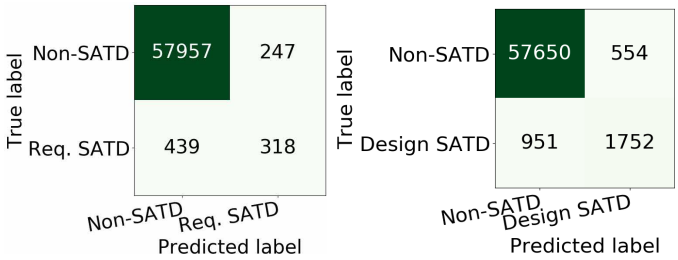

Fig. 3. Confusion metrics of Maldonado et al. for requirement and design SATD classification.

E. Results

In this sub-section, We provide results for answering the experiment questions EQ 1-4.

EQ1: How well n-gram IDF deals with the multi-class SATD classification problem?

Table II shows the comparison of Precision, Recall, F1-score, and MMCE for n-gram IDF, BOW and TF-IDF-based models. We find that for 7 projects, the F1-score values achieved by n-gram IDF-based model are higher than BOW-based model, and for 8 projects, the MMCE values achieved by n-gram IDF-based model are higher than BOW-based model. Moreover, n-gram IDF-based model surpasses TF-IDF-based model in many cases (i.e., 6 projects in F1-score values and 9 projects in MMCE values). Table III presents the results of Wilcoxon test together with corresponding $\hat{A}_{xy}$ measure to assess the statistical significance and effect size of the improved performance brought by n-gram IDF-based model over traditional technique-based models. As shown, n-gram IDF significantly performs better than BOW and TF-IDF in 9 projects (except the JRuby project) with the effect sizes greater than 0.5. Especially, the large effect size ($\hat{A}_{xy}$ >0.7) of the improvement of n-gram IDF over TF-IDF is obtained in the ArgoUML, Hibernate, and SQuirrel projects.

EQ2: Is random forest classifier more suitable for SATD categorization compared with support vector machine?

Table IV shows the comparison of the performance between our approach using random forest (n-gram IDF + RF) against using Support Machine Vector (n-gram IDF + SVM). Using RF consistently outperforms SVM with respect to the F1-score and MMCE measures across all 10 projects. Moreover, Table V shows the results for the Wilcoxon test and $\hat{A}_{xy}$ measure of comparing RF against SVM. The improvement of our approach using RF over SVM is significant with the effect sizes greater than 0.5 in all projects and also the $\hat{A}_{xy}$ measure being greater than 0.7 in 7 projects.

TABLE II
EVALUATION RESULTS OF N-GRAM IDF, BOW, AND TF-IDF-BASED APPROACHES.

Project	N-gram IDF				Bag-of-Words				TF-IDF			
	Precision	Recall	F1	MMCE	Precision	Recall	F1	MMCE	Precision	Recall	F1	MMCE
Ant	0.710	0.204	0.317	0.518	0.472	0.231	0.311	0.506	0.456	0.287	0.352	0.531
JMeter	0.817	0.730	0.771	0.434	0.751	0.662	0.703	0.463	0.773	0.736	0.754	0.461
ArgoUML	0.831	0.922	0.874	0.348	0.834	0.919	0.874	0.387	0.807	0.925	0.862	0.400
Columba	0.851	0.846	0.849	0.347	0.838	0.793	0.815	0.371	0.768	0.864	0.813	0.373
EMF	0.829	0.309	0.450	0.493	0.392	0.213	0.276	0.557	0.594	0.404	0.481	0.519
Hibernate	0.950	0.723	0.821	0.332	0.919	0.735	0.817	0.408	0.876	0.773	0.821	0.404
JEdit	0.737	0.200	0.315	0.535	0.581	0.257	0.356	0.506	0.724	0.300	0.424	0.495
JFreeChart	0.672	0.462	0.548	0.414	0.677	0.452	0.542	0.432	0.708	0.427	0.533	0.461
JRuby	0.853	0.810	0.831	0.388	0.852	0.815	0.833	0.417	0.867	0.795	0.829	0.437
SQuirrel	0.714	0.683	0.698	0.359	0.481	0.676	0.562	0.410	0.637	0.691	0.663	0.428

TABLE III
COMPARISON BETWEEN N-GRAM IDF AGAINST BOW AND TF-IDF USING WILCOXON TEST AND $\hat{A}_{xy}$ EFFECT SIZE.

Project	N-gram IDF VS.	
	Bag-of-Words	TF-IDF
Ant	<0.001 [0.67]	<0.001 [0.68]
JMeter	<0.001 [0.59]	<0.001 [0.60]
ArgoUML	<0.001 [0.53]	0.0 [0.73]
Columba	<0.001 [0.53]	<0.001 [0.66]
EMF	<0.001 [0.65]	<0.001 [0.62]
Hibernate	<0.001 [0.67]	<0.001 [0.75]
JEdit	<0.001 [0.55]	0.0 [0.68]
JFreeChart	<0.001 [0.56]	<0.001 [0.58]
JRuby	0.17 [0.48]	<0.001 [0.50]
SQuirrel	<0.001 [0.60]	0.0 [0.73]

TABLE IV
EVALUATION RESULTS OF RANDOM FOREST AND SUPPORT VECTOR MACHINE-BASED APPROACHES.

Project	Random Forest				Support Vector Machine			
	P	R	F1	M	P	R	F1	M
Ant	<0.71	0.20	0.32	0.52	0.50	0.13	0.20	0.59
JMeter	<0.82	0.73	0.77	0.43	0.87	0.51	0.65	0.50
ArgoUML	<0.83	0.92	0.87	0.35	0.83	0.87	0.85	0.38
Columba	<0.85	0.85	0.85	0.35	0.91	0.66	0.76	0.47
EMF	<0.83	0.31	0.45	0.49	0.87	0.21	0.34	0.61
Hibernate	<0.90	0.72	0.82	0.33	0.93	0.71	0.80	0.44
JEdit	<0.74	0.20	0.32	0.54	0.91	0.05	0.09	0.63
JFreeChart	<0.67	0.46	0.55	0.41	0.98	0.26	0.41	0.54
JRuby	<0.85	0.81	0.83	0.39	0.89	0.44	0.59	0.51
SQuirrel	<0.71	0.68	0.70	0.36	0.94	0.50	0.66	0.46

TABLE V
COMPARISON OF RANDOM FOREST AND SUPPORT VECTOR MACHINE USING WILCOXON TEST AND $\hat{A}_{xy}$ EFFECT SIZE.

Project	Random Forest VS. Support Vector Machine
Ant	<0.001 [0.80]
JMeter	<0.001 [0.84]
ArgoUML	<0.001 [0.73]
Columba	<0.001 [0.86]
EMF	<0.001 [0.77]
Hibernate	<0.001 [0.69]
JEdit	<0.001 [0.92]
JFreeChart	<0.001 [0.79]
JRuby	<0.001 [0.69]
SQuirrel	<0.001 [0.51]

TABLE VI
COMPARISON OF OUR APPROACH AND BASELINE FOR SATD USING WILCOXON TEST AND $\hat{A}_{xy}$ EFFECT SIZE.

Project	Our Approach VS. Maldonado et al.
Ant	<0.001 [0.75]
JMeter	<0.001 [0.82]
ArgoUML	<0.001 [0.64]
Columba	<0.001 [0.75]
EMF	<0.001 [0.71]
Hibernate	<0.001 [0.71]
JEdit	<0.001 [0.88]
JFreeChart	<0.001 [0.72]
JRuby	<0.001 [0.57]
SQuirrel	0.0 [0.43]

TABLE VII
COMPARISON OF F1-SCORE BETWEEN OUR APPROACH AND BASELINE FOR SELF-ADMITTED TECHNICAL DEBT.

Project	Our Approach	Maldonado et al.
Ant	0.317	0.512
JMeter	0.771	0.715
ArgoUML	0.874	0.819
Columba	0.849	0.750
EMF	0.450	0.462
Hibernate	0.821	0.763
JEdit	0.315	0.461
JFreeChart	0.548	0.513
JRuby	0.831	0.773
SQuirrel	0.698	0.593

EQ3: Can our approach outperform the baseline for self-admitted technical debt?

Table VII demonstrates the F1-score values provided by our approach and the baseline. We find that our approach outperforms Maldonado et al. in most cases (7 projects) when detecting self-admitted technical debt. The ArgoUML project achieves the highest F1-score value of 0.874. Besides, we compare the outcomes provided by our approach and the baseline utilizing the Wilcoxon test and $\hat{A}_{xy}$ measure. Table VI shows the results of the Wilcoxon test with the corresponding $\hat{A}_{xy}$ effect size of our approach against the baseline. The differences are statistically significant (p-value <0.05) in all cases. Indeed, the improvement of our approach over the baseline is significant with the effect sizes greater than 0.5 in 9 projects. The effect sizes of 7 out of 10 projects are considered as large ($\hat{A}_{xy}$ >0.7). The largest effect size is 0.88 accomplishing in the JEdit project.

EQ4: Can our approach classify design and requirement self-admitted technical debt more accurately than the baseline?

As demonstrated in the figures 3 and 2, our multi-class classifi-

TABLE VIII
EVALUATION RESULTS OF OUR APPROACH USING UNDER-SAMPLING AND WITHOUT UNDER-SAMPLING.

Project	Our Approach with Under-sampling				Our Approach without Under-sampling			
	P	R	F1	M	P	R	F1	M
Ant	0.71	0.20	0.32	0.52	0.74	0.19	0.30	0.55
JMeter	0.82	0.73	0.77	0.43	0.78	0.66	0.72	0.49
ArgoUML	0.83	0.92	0.87	0.35	0.85	0.92	0.88	0.36
Columba	0.85	0.85	0.85	0.35	0.78	0.68	0.73	0.42
EMF	0.83	0.31	0.45	0.49	0.84	0.29	0.43	0.55
Hibernate	0.95	0.72	0.82	0.33	0.95	0.73	0.83	0.41
JEdit	0.74	0.20	0.32	0.54	0.96	0.21	0.34	0.56
JFreeChart	0.67	0.46	0.55	0.41	0.68	0.44	0.54	0.48
JRuby	0.85	0.81	0.83	0.39	0.92	0.63	0.74	0.47
SQuirrel	0.71	0.68	0.70	0.36	0.89	0.54	0.67	0.45

cation model improves the baseline model by 33% when classifying requirement SATD with the true positive value of requirement SATD class exceeds Maldonado's work by 253. Nevertheless, for design SATD classification, our model achieves the true positive value of design SATD of 548. We assume that the multi-class classification is more challenging than the binary classification; therefore, we need to apply advanced techniques to enhance the performance of our approach in the future. Moreover, a prior study manifests that the detection of design SATD requires many training data as design SATD comments are less similar to each other compared to requirement SATD comments [7]. Under these circumstances, our multi-class classification model finds more difficult to classify design SATD comments.

V. DISCUSSION

A. Threats to Validity

1) Internal threat: We applied Maldonado et al. [7] as a baseline in our experiment. However, they did not explicitly provide the implementation of their model. We therefore needed to re-implement their classification model by strictly following their methodology and we concede that our implementation may not be identical to all implementations in their work. We minimize this threat by constructing their approach using the dataset published in their work and then comparing the results achieved by our own version of their classification model with the results provided in their work. We found that our results are almost consistent with their results.

We utilized an n-gram extraction tool namely Ngweight to construct our n-gram dictionary without modifying any parameters in this work. If the n-gram IDF tool receives a different parameter, it will create a different dictionary from ours. Also, constructing an n-gram dictionary by other algorithms could produce a different one.

Besides, we applied an instance hardness under-sampling library, which automatically computes the threshold value for each learning algorithm, to deal with the imbalanced dataset. Thus, changing the algorithm or adjusting the threshold value could provide distinct outcomes.

2) External threat: The dataset was leveraged from Maldonado et al. [7] consists of JAVA source code comments of 10 open source projects from different domains. However, our work may not generalize to other open-source projects, commercial projects, projects from different areas, projects are written in other languages (e.g., Python), and projects are not written in the English language. Also, projects contain low source code comments, or no source code comments are not suitable for our work since this study mainly focuses on classifying SATD through source code comments.

B. Is under-sampling a good decision for dealing with the imbalanced dataset?

To investigate the effectiveness of instance hardness under-sampling for imbalanced data problem, we conducted the comparison between our approach applying under-sampling and without under-sampling in which derived features from n-gram IDF and trained random forest to classify design and requirement SATD. We set the evaluation by splitting the dataset into a training set (8 projects), testing set (1 project), and validation set (1 project). After 10 rounds of the evaluation, the performance measures achieved by two approaches were reported.

Table VIII compares the classification performance of our approach using under-sampling against without under-sampling. Our approach (using under-sampling) performs better than the comparing approach (without under-sampling) with respect to F1-score and MMCE measures in most cases. Based on the experiment results, we assume that the under-sampling technique supports our approach to tackle the imbalanced data problem for improving the performance of our classification model.

VI. CONCLUSION

Our work focuses on design and requirement SATD classification through the use of n-gram IDF, instance hardness under-sampling, and random forest classifier. The proposed approach aims to address the limitations of binary models (i.e., misclassification) in order to support development teams to develop a payback plan for design and requirement SATD management, the most frequent types of SATD. In our experiment, we use source code comments of 10 open-source software projects to build the multi-class classification model to automatically predict design and requirement SATD in a new target project.

The proposed approach consistently outperforms traditional techniques (i.e. BOW and TF-IDF) and one alternative (i.e., using SVM) according to the experiment results. We also outperform Maldonado et al. for SATD identification with the highest F1-score values of 0.874. Especially, when classifying requirement SATD, our approach has exceeded the baseline by 33%. Resembling our previous study, the use of n-gram IDF (with under-sampling, and ensemble classifier) has improved our predictive performance over the traditional techniques since n-gram IDF is capable of capture useful phrases or words indicates the presence of design and requirement SATD in source code comments.

For future work, we plan to improve the performance of our approach, especially classifying design SATD by using advanced algorithms (i.e. word embedding and deep learning methods). Besides open-source software projects, We plan to run our multi-class classification model on commercial software projects. Moreover, we plan to extend our work to classify all types of SATD including the types which are not common in software projects (e.g., defect, test, and documentation debt). We also plan to develop our approach to detect SATD comments in other programming languages.

ACKNOWLEDGMENTS

This work has been supported by JSPS KAKENHI (Grant Number 16H05857 and 17H00731).

REFERENCES

[1] W. Cunningham, "The wycash portfolio management system," in Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), ser. OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30.

[2] A. Martini, T. Besker, and J. Bosch, "The introduction of technical debt tracking in large companies," in 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Dec 2016, pp. 161–168.

[3] S. Wehaibi, E. Shihab, and L. Guerrouj, "Examining the impact of self-admitted technical debt on software quality," in 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1, March 2016, pp. 179–188.

[4] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in 2014 IEEE International Conference on Software Maintenance and Evolution, Sep. 2014, pp. 91–100.

[5] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), May 2016, pp. 315–326.

[6] E. d. S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Oct 2015, pp. 9–15.

[7] E. d. S. Maldonado, E. Shihab, and 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, Nov 2017.

[8] S. Wattanakriengkrai, R. Maipradit, H. Hata, M. Choetkiertikul, T. Sunetnanta, and K. Matsumoto, "Identifying design and requirement self-admitted technical debt using n-gram idf," in 2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP), Dec 2018, pp. 7–12.

[9] M. A. d. F. Farias, M. G. d. M. Neto, A. B. d. Silva, and R. O. Spnola, "A contextualized vocabulary model for identifying technical debt on code comments," in 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Oct 2015, pp. 25–32.

[10] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," Empirical Softw. Engg., vol. 23, no. 1, pp. 418–451, Feb. 2018.

[11] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating change-level self-admitted technical debt determination," IEEE Transactions on Software Engineering, pp. 1–1, 2018.

[12] J. Flisar and V. Podgorelec, "Enhanced feature selection using word embeddings for self-admitted technical debt identification," in 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Aug 2018, pp. 230–233.

[13] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network based detection of self-admitted technical debt: From performance to explainability," ACM Transactions on Software Engineering and Methodology (TOSEM), mar 2019.

[14] D. Bespalov, B. Bai, Y. Qi, and A. Shokoufandeh, "Sentiment classification based on supervised latent n-gram analysis," in Proceedings of the 20th ACM International Conference on Information and Knowledge Management, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 375–382.

[15] M. Shirakawa, T. Hara, and S. Nishio, "N-gram idf: A global term weighting scheme based on information distance," in Proceedings of the 24th International Conference on World Wide Web, ser. WWW '15, 2015, pp. 960–970.

[16] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? bug report classification using n-gram idf," in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Sept 2017, pp. 534–538.

[17] R. Maipradit, H. Hata, and K. Matsumoto, "Sentiment classification using n-gram idf and automated machine learning," IEEE Software, pp. 1–1, 2019.

[18] M. R. Smith, T. Martinez, and C. Giraud-Carrier, "An instance level analysis of data complexity," Machine Learning, vol. 95, no. 2, pp. 225–256, May 2014.

[19] N. Verdikha, T. Adji, and A. Permanasari, "Study of undersampling method: Instance hardness threshold with various estimators for hate speech classification," IJITEE (International Journal of Information Technology and Electrical Engineering), vol. 2, 12 2018.

[20] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, Oct 2001.

[21] C. Galvn Tejada, J. Galvn Tejada, J. Celaya Padilla, J. R. Delgado Contreras, R. Magallanes-Quintanar, M. L. Martinez-Fierro, I. Garza-Veloz, Y. Lopez, and H. Gamboa-Rosales, "An analysis of audio features to develop a human activity recognition model using genetic algorithms, random forests, and neural networks," Mobile Information Systems, vol. 2016, pp. 1–10, 01 2016.

[22] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in 2007 IEEE 11th International Conference on Computer Vision, Oct 2007, pp. 1–8.

[23] J. Klema and A. Almonayyes, "Automatic categorization of fanatic text using random forests," Kuwait Journal of Science and Engineering, vol. 33, pp. 1–18, 12 2006.

[24] Y. Aphinyanaphongs, B. Ray, A. Statnikov, and P. Krebs, "Text classification for automatic detection of alcohol use-related tweets: A feasibility study," in Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), Aug 2014, pp. 93–97.

[25] D. Liparas, Y. HaCohen-Kerner, A. Moumtzidou, S. Vrochidis, and I. Kompatsiaris, "News articles classification using random forests and weighted multimodal features," in Multidisciplinary Information Retrieval, D. Lamas and P. Buitelaar, Eds. Cham: Springer International Publishing, 2014, pp. 63–75.

[26] A. Alahmadi, A. Joorabchi, and A. E. Mahdi, "A new text representation scheme combining bag-of-words and bag-of-concepts approaches for automatic text classification," in 2013 7th IEEE GCC Conference and Exhibition (GCC), Nov 2013, pp. 108–113.

[27] F. Sebastiani, "Machine learning in automated text categorization," ACM Comput. Surv., vol. 34, no. 1, pp. 1–47, Mar. 2002.

[28] A. Basu, C. Walters, and M. Shepherd, "Support vector machines for text categorization," in 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, Jan 2003, pp. 7 pp.–.

[29] N. Rajvanshi and K. R. Chowdhary, "Comparison of svm and nave bayes text classification algorithms using weka," International Journal of Engineering Research and, vol. V6, 09 2017.

[30] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in Machine Learning: ECML-98, C. Nédellec and C. Rouveirol, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142.

[31] P. Y. Pawar and S. H. Gawande, "A comparative study on different types of approaches to text categorization," 2012.

[32] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," ACM Trans. Intell. Syst. Technol., vol. 2, no. 3, pp. 27:1–27:27, May 2011.

[33] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," Information Processing & Management, vol. 45, no. 4, pp. 427 – 437, 2009.

[34] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Characterization and prediction of issue-related risks in software projects," in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, May 2015, pp. 280–291.

[35] K. Muller, "Statistical power analysis for the behavioral sciences," Technometrics, vol. 31, no. 4, pp. 499–500, 1989.

[36] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," Journal of Educational and Behavioral Statistics, vol. 25, no. 2, pp. 101–132, 2000.

[37] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in Machine Learning and Data Mining in Pattern Recognition, P. Perner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 154–168.

[38] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse, "An empirical study of learning from imbalanced data using random forest," in 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), vol. 2, Oct 2007, pp. 310–317.