An Empirical Study on the Impact of Class Overlap in Just-in-Time Software Defect Prediction

Minyang Yi*, Guisheng Fan*✉, Huiqun Yu*‡✉, Xingguang Yang*†
*Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China
†Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China
‡Shanghai Engineering Research Center of Smart Energy, Shanghai, China

***Abstract*—Just-in-time software defect prediction (JIT-SDP) is an active research topic in the field of software engineering, aiming at identifying defect-inducing code changes. Most of the current JIT-SDP work focused on model construction. It is often ignored that the performance of classifiers often depends on high quality data. In this paper, we first investigate the impact of the class overlap problem on the performance of the classifiers in JIT-SDP, and propose a new effective preprocessing method (IKMCCA-TL) combining improved K-Means clustering cleaning approach and Tomek-link method. In order to objectively estimate the impact of class overlap on the classifiers in JIT-SDP, we conduct a large-scale empirical study on the data sets of six open source projects and compare the performance of LR, RF and KNN classifiers by using IKMCCA or KMCCA or NCL and without cleaning data. Experimental results show that after removing overlapping instances, the performance of the classifiers is significantly improved in terms of balance, recall and AUC and our proposed method achieves the best performance.**

***Index Terms*—Just-in-time software defect prediction, class overlap, K-Means clustering, Tomek-link**

I. Introduction

Software defect prediction technology is one of the most popular research topics among academic and industrial organizations [1]. In the development process of large-scale software, developers cannot avoid software defects. Defects in the software cause great harm and loss to users, customers and enterprises. In order to minimize the defects in the software system, developers will put a lot of effort into testing the software.

Just-in-time software defect prediction (JIT-SDP) is a special software defect prediction (SDP), which is at the software change level instead of the module level (for example, function, file, or class level) and it refers to the technology that predicts whether there are defects in each code change submitted by the developer [2]. Once the software change that caused the defect is implemented, it will be identified in JIT-SDP.

In SDP, researchers have done a lot of research on class imbalance and noise cleaning, and they have begun to study the problem of class overlap. Tang et al. [3] proposed a K-Means clustering cleanup method to clean noise instances by calculating the noise factor value (NF_t) of each instance for each cluster, and deleting the top p%. Chen et al. [4] applied Neighbor Clean Learning (NCL) rules to remove overlapping instances for SDP. Gong et al. [5] proposed an improved K-Means clustering cleanup method (IKMCCA) to solve the problem of class overlap and class imbalance in SDP.

The main research work in JIT-SDP includes model construction and feature selection. In order to investigate the impact of the class overlap problem in JIT-SDP, we firstly uses the NCL, KMCCA and IKMCCA methods to process the data, and check whether the performance of the three classifiers is affected. Considering that overlap usually occurs near the decision boundary and removing important boundary instances will reduce the learning process, we propose an effective cleaning approach (IKMCCA-TL) combining IKMCCA method and Tomek-link pair method.

The rest of this article is organized as follows. Section II introduces related work. Section III introduces the proposed algorithm IKMCCA-TL. Experimental setup is described in section IV. Experimental results and discussion are presented in section V. Section VI describes the threats to validity. The conclusion and future work are presented in section VII.

II. Related Work

A. Just-in-time Software Defect Prediction

The idea of JIT-SDP was proposed by Mockus et al. [6] and scores of change metrics to predict whether changes are defect-inducing or clean was designed by them. Kamei et al. [2] performed a large-scale empirical study in JIT-SDP. They collected eleven data sets which include six open-source projects and five commercial projects.

Subsequently, various methods were proposed to improve the performance of the prediction model for JIT-SDP. Yang et al. [7] validated the availability of progressive sampling in the JIT-SDP issue which can reduce the size of the defect data sets and reduce the cost of data sets acquisition. Chen et al. [8] proposed a JIT-SDP model MULTI based on multi-objective optimization algorithm NSGA-II to generate optimal solutions. The two goals of MULTI optimization are designed through the benefit-cost analysis. Yang et al. [9] proposed three optimal solutions selection strategies: benefit priority (BP), cost priority (CP), and a compromise between cost

Corresponding Authors: Guisheng Fan (gsfan@ecust.edu.cn), Huiqun Yu (yhq@ecust.edu.cn) DOI reference number: 10.18293/SEKE2021-076

and benefit (CCB) to improve the performance of MULTI. Yang et al. [10] proposed a differential evolution (DE) based supervised method DEJIT to build JIT-SDP models which can significantly improve the effort-aware prediction performance in the three evaluation scenarios.

B. Class overlap

Class overlap means that some instances of different classes in the training data are close or even overlapped in the distribution space and often results in poor class boundaries and affects the performance of the learner. In the current SDP research, the problem of class overlap is mainly regarded as data quality or noise detection. Tang et al. [3] proposed a cluster-based noise detection method, which uses the outlier detection method to calculate the noise factor (NF), and removed the top $p\%$ of NF. Chen et al. [4] proposed Neighborhood Clean Learning (NCL) to solve the problems of class overlap and class imbalance. The experimental results show that, compared with the existing learning methods, the new learning model can obtain the best values in terms of G-mean and AUC. In order to take into account the effects of class overlap and class imbalance, Gong et al. [5] proposed an improved K-Means clustering cleanup method (IKMCCA) to solve the problem of class overlap and class imbalance in SDP. Experiments have proved that compared to KMCCA and NCL methods, the IKMCCA method can obtain the best performance.

III. IKMCCA-TL

The evaluation model of IKMCCA-TL is shown in Fig 1. In the following section, we elaborate on the details of IKMCCA-TL.

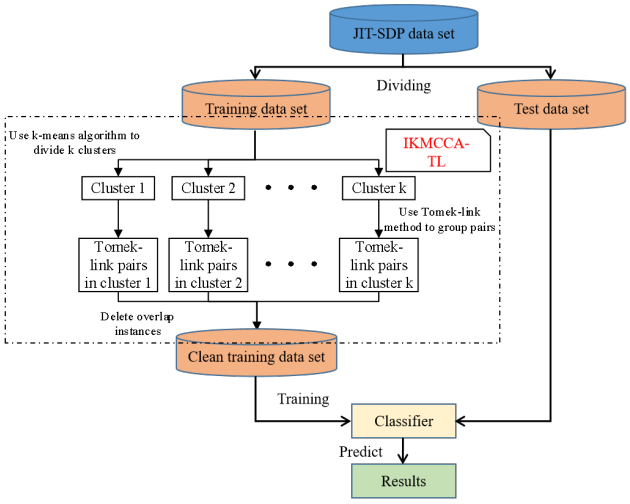

Fig. 1. The evaluation model of IKMCCA-TL

A. IKMCCA

In order to consider both the impact of class overlap and class imbalance at the same time, Gong et al. [5] proposed an improved K-Means clustering cleanup method (IKMCCA) to solve the problem of class overlap and class imbalance in traditional defect prediction. In the step of deleting overlapping instances in the IKMCCA algorithm, if the percentage of defective instances in the cluster i is lower than $p\%$, delete the defective instances in the cluster. On the contrary, instances without defects in this cluster will be deleted.

B. Tomek-link pair

The Tomek-link undersampling algorithm is used to eliminate boundary instances [11] . Given two instances $t_i \in T$ and $t_j \in T$ belonging to different classes, let $\text{distance}(t_i, t_j)$ be the distance between them. If a pair of instances (t_i, t_j) does not exist $\text{distance}(t_i, t_l) < \text{distance}(t_i, t_j)$ or $\text{distance}(t_l, t_j) < \text{distance}(t_i, t_j)$ for any other instance $t_l \in T$, then this pair of instances is called a Tomek-link pair.

C. IKMCCA-TL

The overlap often occurs near the decision boundary in the case of class overlap. In this case, the excessive elimination of boundary instances will drift the decision boundary between the minority class and the majority class, which in turn will reduce the learning process. Therefore, we improve the IKMCCA algorithm and combines the Tomek-link pair to make it possible to balance the data distribution, without distorting the decision boundary, and remove only unimportant boundary instances and unimportant redundant most instances. The pseudo code of IKMCCA-TL is shown as Algorithm 1.

IV. EXPERIMENTAL SETUP

In this paper, the problem of class overlap is studied in JIT-SDP, and the following two research questions are designed.

- RQ1: How does class overlap influence the prediction performance of the basic classifiers in JIT-SDP?
- RQ2: Why our proposed method (IKMCCA-TL) is more effective in reducing the impact of class overlap on the classifiers?

The experimental hardware environment is *Intel(R) Core(TM) i7-10875 CPU@ 2.30GHz; RAM 16.00GB*. The experimental code is written in Python.

A. Data Sets

The experiment considers the data sets of six open source projects shared by Kamei et al. [2], which have been widely used in JIT-SDP studies. The data set comes from 6 open source projects, which is shown in the Table I.

The data sets contains 14 change metrics. These characteristics are briefly introduced in [2], which involve five dimensions: diffusion, size, purpose, history, and experience.

B. Performance Indicators

In order to explore the influence of class overlap on the learner, we use three performance measures including $Balance$, $Recall$ and AUC.

Algorithm 1: IKMCCA-TL

Input: training set: T, the parameter m.
// m is used to determine the number of clusters
Output: a clean training set: T_{new}

```
begin
    n = the number of instances in T
    d = the number of defective instances in T
    p = d/n
    k = [n/m]
    using K-means algorithm to divide T into k clusters
    for i = 1 → k do
        find the instance pairs that are the Tomke-link pairs in cluster i
        compute the ratio r of of defective instances to all instances in cluster i
        if r > p then
            delete the non-defective instances of the Tomke-link pairs in cluster i
        else
            delete the defective instances of the Tomke-link pairs in cluster i
    end
    T_new is the set combining the remaining instances in each cluster
end
```

TABLE I
THE INFORMATION OF DATA SETS

Project	Period	#defective changes	#changes	%defect rate
BUG	1998/08/26∼2006/12/16	1696	4620	36.71%
COL	2002/11/25∼2006/07/27	1361	4455	30.55%
JDT	2001/05/02∼2007/12/31	5089	35386	14.38%
MOZ	2000/01/01∼2006/12/17	5149	98275	5.24%
PLA	2001/05/02∼2007/12/31	9452	64250	14.71%
POS	1996/07/09∼2010/05/15	5119	20431	25.06%

C. Parameter Setting

In IKMCCA and IKMCCA-TL methods, we set the percentage $p\%$ to the ratio of defective instances, and the parameter m is set as 20, which is the same as the references [5]. The parameter settings of KMCCA and NCL methods are the same as references [3] and [4]. 85% of the instances are randomly selected as training data, and rest instances are used as test data. For eliminating the randomness of the experiment, the experiment is repeated 20 times.

V. Experimental Results and Discussion

This section answers the questions raised in Section IV through experiments. And data processing is the same as [2].

A. Analysis for RQ1

In order to test the degree of impact of overlapping instances in software defect data sets on the performance of the classifiers, we adopt RF, KNN and LR classifiers.

As shown in the table II, through the experimental results, we can find that after using the NCL method to remove overlapping instances, the value of $Recall$ increased by 2.9%, 4.6% and 5% respectively on LR, RF and KNN classifiers. The value of AUC increased by 1%, 1.8% and 2.2% respectively. The value of Bal increased by 1.9%, 3.1% and 3.4% respectively. KMCCA method which only remove the noise instances doesn't achieve much improvement. The IKMCCA method which solves the problem of class overlap and class imbalance achieves better performance than the NCL method.

As shown in experimental result, the class overlap problem will have a serious impact on the performance of the classifier in JIT-SDP. When the overlapping instances of the class are removed, the performance of the classifier will be greatly improved. Considering the problem of class overlap and class imbalance at the same time, the classifier will perform better.

B. Analysis for RQ2

The existence of important boundary instances is also important for accurately defining the decision boundary. In the case of class overlap, the overlap often occurs near the decision boundary. In this case, the excessive elimination of boundary instances will drift the decision boundary between the minority class and the majority class, which in turn will reduce the learning process. In order to solve the above problems, we propose IKMCCA-TL method which only removes unimportant boundary instances and unimportant redundant instances. Compared with the IKMCCA method, the $Recall$ value of the IKMCCA-TL method is increased by 5.5%, 5.4% and 5.4% respectively on LR, RF and KNN classifiers. The value of AUC is increased by 0.7%, 0.8% and 0.6% respectively. The value of bal is increased by 1.9%, 2.5% and 1.4% respectively.

As shown in experimental result, when we only delete unimportant overlap instances of the decision boundary instances, the performance of classifiers can be better.

VI. Threats to Validity

External validity. The results of the experiment can't be guaranteed to apply to all other defect data sets. More data sets should be mined to verify the generalization of experimental results.

Construct validity. Three indicators, including $Recall$, AUC and bal, are used to reflect the performance of the classifier, which is also widely used in [5].

Internal validity. The threats to internal validity are mainly from experimental code. The mature python libraries are used and the code is checked to reduce the errors.

VII. Conclusions and Future Work

In JIT-SDP, the performance of the classifier often depends on high-quality data. In the past, the impact of overlapping classes on learning models was ignored. Therefore, we propose a method IKMCCA-TL that can better remove unimportant boundary instances and unimportant redundant instances, and

TABLE II
EXPERIMENTAL RESULTS OF THE ABOVE METHODS ON THE CLASSIFIERS LR, RF, KNN

project	method	Logistic regression			Random forests			K-nearest neighbor		
		Recall	AUC	Bal	Recall	AUC	Bal	Recall	AUC	Bal
BUG	Without removing	0.408	0.634	0.569	0.408	0.679	0.625	0.448	0.615	0.579
	NCL	0.458	0.652	0.561	0.553	0.700	0.666	0.523	0.650	0.627
	KMCCA	0.394	0.631	0.636	0.490	0.677	0.626	0.458	0.620	0.586
	IKMCCA	0.495	0.646	0.612	0.583	0.680	0.665	0.546	0.623	0.613
	IKMCCA-TL	**0.676**	**0.658**	**0.656**	**0.697**	**0.692**	**0.689**	**0.685**	**0.630**	**0.624**
COL	Without removing	0.334	0.6284	0.526	0.387	0.639	0.560	0.364	0.603	0.536
	NCL	0.399	0.649	0.569	0.450	0.666	0.602	0.435	0.634	0.583
	KMCCA	0.361	0.635	0.543	0.405	0.643	0.570	0.372	0.604	0.540
	IKMCCA	0.489	0.662	0.619	0.517	0.661	0.630	0.501	0.621	0.601
	IKMCCA-TL	**0.592**	**0.674**	**0.662**	**0.634**	**0.678**	**0.674**	**0.601**	**0.629**	**0.627**
JDT	Without removing	0.041	0.516	0.322	0.098	0.541	0.362	0.128	0.546	0.382
	NCL	0.059	0.524	0.335	0.149	0.564	0.398	0.177	0.566	0.417
	KMCCA	0.045	0.518	0.324	0.100	0.542	0.363	0.131	0.548	0.385
	IKMCCA	0.152	0.559	0.400	0.274	0.605	0.484	0.298	0.594	0.498
	IKMCCA-TL	**0.156**	**0.561**	**0.403**	**0.283**	**0.607**	**0.490**	**0.316**	**0.599**	**0.509**
MOZ	Without removing	0.029	0.514	0.314	0.054	0.526	0.331	0.080	0.536	0.349
	NCL	0.031	0.515	0.315	0.065	0.531	0.339	0.098	0.544	0.362
	KMCCA	0.031	0.515	0.315	0.055	0.526	0.332	0.082	0.537	0.350
	IKMCCA	0.059	0.527	0.334	0.130	0.560	0.358	0.163	0.570	0.408
	IKMCCA-TL	**0.061**	**0.528**	**0.336**	**0.134**	**0.562**	**0.387**	**0.167**	**0.572**	**0.411**
PLA	Without removing	0.079	0.535	0.349	0.162	0.571	0.407	0.145	0.553	0.395
	NCL	0.092	0.540	0.358	0.202	0.589	0.435	0.205	0.579	0.437
	KMCCA	0.081	0.535	0.350	0.152	0.567	0.401	0.146	0.553	0.395
	IKMCCA	0.223	0.590	0.449	0.336	0.633	0.528	0.325	0.603	0.515
	IKMCCA-TL	**0.225**	**0.591**	**0.451**	**0.355**	**0.639**	**0.541**	**0.338**	**0.605**	**0.523**
POS	Without removing	0.339	0.641	0.530	0.391	0.662	0.567	0.369	0.633	0.548
	NCL	0.363	0.650	0.547	0.435	0.678	0.597	0.400	0.644	0.568
	KMCCA	0.326	0.636	0.522	0.392	0.663	0.567	0.377	0.637	0.553
	IKMCCA	0.493	0.685	0.631	0.536	0.700	0.658	0.512	0.664	0.631
	IKMCCA-TL	**0.532**	**0.693**	**0.653**	**0.594**	**0.712**	**0.689**	**0.565**	**0.669**	**0.653**
Average	Without removing	0.205	0.578	0.435	0.263	0.603	0.475	0.256	0.581	0.465
	NCL	0.234	0.588	0.454	0.309	0.621	0.506	0.306	0.603	0.499
	KMCCA	0.206	0.579	0.436	0.265	0.603	0.477	0.261	0.583	0.468
	IKMCCA	0.318	0.611	0.508	0.396	0.640	0.554	0.391	0.612	0.544
	IKMCCA-TL	**0.373**	**0.618**	**0.527**	**0.450**	**0.648**	**0.579**	**0.445**	**0.618**	**0.558**

investigate whether NCL, KMCCA, IKMCCA and IKMCCA-TL methods can improve the performance of the classifiers. We conduct a large-scale empirical study on data sets of six open source projects. Experimental results show that using these methods to eliminate overlapping instances can achieve significantly better performance in terms of bal, Recall, and AUC. And our proposed method IKMCCA-TL can better improve the performance of the classifiers by eliminating class overlapping instances.

In the future, more data sets from commercial projects will be mined to verify the generalization of experimental results.

ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China under Grant nos. 61702334 and 61772200, Shanghai Municipal Natural Science Foundation under Grant nos. 17ZR1406900 and 17ZR1429700.

REFERENCES

[1] X. Chen, Q. Gu, W. Liu, S. Liu, C. Ni, Survey of static software defect prediction, Journal of Software, 27 (1) (2016).

[2] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, IEEE Transactions on Software Engineering, 39 (6) (2013) :757–773.

[3] W. Tang, T. M, Khoshgoftaar, Noise identification with the k-means algorithm, in: Proceedings of the 16th International Conference on Tools with Artificial Intelligence, ICTAI, 2004, 2004, pp. 373–378.

[4] L. Chen, B. Fang, Z. Shang, Y. Tang, Tackling class overlap and imbalance problems in software defect prediction, Software Quality Journal, 26 (1) (2018) :97–125.

[5] L. Gong, S. Jiang, R. Wang, L. Jiang, Empirical evaluation of the impact of class overlap on software defect prediction, in: Proceedings of the 34th International Conference on Automated Software Engineering, ASE 2019, 2019, pp. 698–709.

[6] A. Mockus, D. M. Weiss, Predicting risk of software changes, Bell Labs Technical Journal, 5 (2) (2000) :169–180.

[7] X. Yang, H. Yu, G. Fan, K. Yang, K. Shi, An empirical study on progressive sampling for just-in-time software defect prediction, in: Proceedings of the 26th Asia-Pacific Software Engineering Conference, APSEC, 2019, 2019, pp. 12–18.

[8] X. Chen, Y. Zhao, Q. Wang, Z. Yuan, MULTI: multi-objective effort-aware just-in-time software defect prediction, Information & Software Technology, 93 (2018) :1–13.

[9] X. Yang, H. Yu, G. Fan, K. Yang, An empirical study on optimal solutions selection strategies for effort-aware just-in-time software defect prediction, in: Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering, SEKE, 2019, 2019, pp. 319–324.

[10] X. Yang, H. Yu, G. Fan, K. Yang, Dejit: A differential evolution algorithm for effort-aware just-in-time software defect prediction, International Journal of Software Engineering, 31 (3) (2021) :289–310.

[11] T. I, Two modifications of CNN, IEEE Transactions on Systems, Man, and Cybernetics, 7 (2) (1976) :769–772.