2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction

Lina Gong
School of Computer Science and Technology,
China University of Mining and Technology,
Xuzhou 221116, China
Mine Digitization Engineering Research Center of
Ministry of Education, Xuzhou 221116, China
Department of Information Science and Engineering,
Zaozhuang University, Zaozhuang 277160, China
Email: linagong@cumt.edu.cn

Shujuan Jiang, Rongcun Wang and Li Jiang
School of Computer Science and Technology,
China University of Mining and Technology,
Xuzhou 221116, China
Mine Digitization Engineering Research Center of
Ministry of Education, Xuzhou 221116, China
Email: shjjiang@cumt.edu.cn
Email: rcwang@cumt.edu.cn
Email: lijiang@cumt.edu.cn

***Abstract*—Software defect prediction (SDP) utilizes the learning models to detect the defective modules in project, and their performance depends on the quality of training data. The previous researches mainly focus on the quality problems of class imbalance and feature redundancy. However, training data often contains some instances that belong to different class but have similar values on features, and this leads to class overlap to affect the quality of training data. Our goal is to investigate the impact of class overlap on software defect prediction. At the same time, we propose an improved K-Means clustering cleaning approach (IKMCCA) to solve both the class overlap and class imbalance problems. Specifically, we check whether K-Means clustering cleaning approach (KMCCA) or neighborhood cleaning learning (NCL) or IKMCCA is feasible to improve defect detection performance for two cases (i) within-project defect prediction (WPDP) (ii) cross-project defect prediction (CPDP). To have an objective estimate of class overlap, we carry out our investigations on 28 open source projects, and compare the performance of state-of-the-art learning models for the above-mentioned cases by using IKMCCA or KMCCA or NCL VS. without cleaning data. The experimental results make clear that learning models obtain significantly better performance in terms of *balance*, *Recall* and *AUC* for both WPDP and CPDP when the overlapping instances are removed. Moreover, it is better to consider both class overlap and class imbalance.**

***Index Terms*—Class overlap, Software defect prediction, K-Means clustering, Machine learning**

I. Introduction

Defect prediction technology is one of the research topics among academic and industrial organizations [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. With the development of machine learning methods, more and more classification models are applied to software defect prediction (SDP) to detect as many defective modules as possible with minimal cost. The quality of training dataset from software projects seriously affects the prediction capabilities of learning models. Previous studies in the data quality of SDP mainly focus on class imbalance and feature redundancy, and have proposed lots of models to deal with these quality problems.

However, in practical data collection, due to the objective condition, some instances with different classes may have similar values on some features, which results in overlap in feature space. These instances are called overlapping instances, triggering the class overlap problem. Class overlap is one of the bottlenecks in data mining and machine learning, and SDP is no exception. Many instances from NASA [11], [12], AEEEM [13], ReLink [14], SOFTLAB[11] and MORPH [15] are overlapped (As shown in figure 1, many instances are located in overlapping space, where learning models often fail to classify), which hinders the prediction performance of learning models.

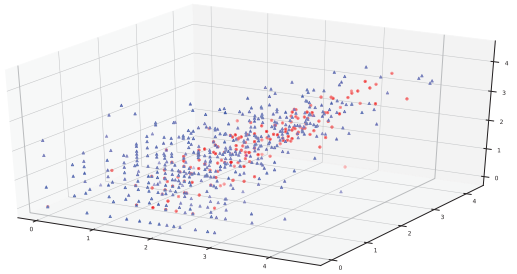

Fig. 1. The distribution of PC4 dataset in NASA. Figure 1 denotes the PC4 has high overlapping instances, these overlapping areas are diffcult to classifier.

In the past few years, many studies have encountered class overlap problem in practical fields, such as credit card fraud detection [16], text classification [17]. They found the overlapping instances involve the performance of learning models, which often becomes a thorny problem when combined with other factors, such as class imbalance. In SDP, researchers mainly focus on data preprocessing including class imbalance and noise cleaning, few studies have paid attention on class overlap problem. Chen et al. [18] attempted to apply the neighborhood cleaning learning (NCL) rule to remove overlapping instances for SDP. However, the NCL method only removed the conflicting non-defective instances to eliminate the class overlap, and the defective instances located in non-defective

978-1-7281-2508-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ASE.2019.00071

overlapping space were not considered.

In this study, we are committed to identify and process overlapping instances and propose an improved K-Means clustering cleaning approach (IKMCCA) to remove the overlapping instances. Our contribution is to investigate the impact of class overlap on software defect prediction, and excavate how class overlap influence the prediction performance for both WPDP and CPDP. So we have structured our work according to the following two research questions:

RQ1: How does class overlap influence the prediction performance of within-project defect prediction approaches?

RQ2: How does class overlap influence the prediction performance of cross-project defect prediction approaches?

We conduct experiments to investigate the impact of class overlap for WPDP and CPDP using 28 overlapping data from NASA, AEEEM, ReLink, SOFTLAB and MORPH. The rest of this study is organized as follows: the next section II provides a review of related work on SDP. Section III provides the methodology containing study questions, methods to build models, data and model evaluation. Section IV lists the results of study. The discussions of our work are described in Section V and the threats to validity of our work are analyzed in Section VI. We conclude the paper in Section VII.

II. Related Work

In this section, we mainly discuss the related researches on the SDP and class overlap.

A. Software Defect Prediction

Software defect prediction technology has been one of the most concerned research topics in software engineering since 1970s [6], [19], [20]. In recent years, with the rapid development of machine learning, various machine learning methods have been widely applied to improve the performance of SDP.

According to whether the training and testing data come from the same project, the SDP is divided into within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). WPDP is to directly apply classifiers such as Naive Bayes (NB) [7], [10], Random forests (RF) [21], K-nearest neighbor (KNN) [22], Support vector machine (SVM) [23], and Logistic regression (LR) [24] to detect the defective modules.

CPDP is to find as many defective modules as possible in one project based on the learning model trained by other projects. The most important problem of CPDP is the different distribution of training and testing data. Many researchers have presented a comparative research on CPDP methods [25], [26], [27]. Zhou et al. [27] found that the simple module-size model method had comparable or better predictive performance than most cross-project defect prediction methods. In our work, we focus our discussions on six empirical studies which could achieve better performance. Detailed descriptions are shown as follows.

Turhan et al. [28] chose similar instances as training set from different projects, and using these similar instances to train k-nearest neighbor (KNN) classifier. Nam et al. [6] used Transfer Component Analysis (TCA) method to map training and testing data into the common feature space. They conducted experiments on ReLink and AEEEM datasets, and proposed TCA+ method for automatic data selection standardization. Chen et al. [29] integrated two levels of data transfer and proposed double transfer boosting (DTB) approach. First, they reshaped the whole distribution of cross-company (CC) dataset by data gravitation to fit WP data. Then, they eliminated negative instances in CC dataset by transfer boosting with NB based classifier. Turhan et al. [30] found mixed-project prediction was reasonable in early phases of development though studying the impact of mixed-project dataset on binary defect prediction. Ryu et al. [31] discussed the class imbalance for the mixed-project defect prediction (MPDP), and put forward the value-cognitive boosting with support vector machine (VCB-SVM) approach. Their experimental results indicated that VCB-SVM could achieve better results when training data only contained fewer label instances of testing project. Xia et al. [32] proposed the HYDRA (HYbrid model Reconstruction Approach) including two phases of genetic and ensemble learning. The phase of genetic algorithm was to build the optimal weight combination of multiple classifiers by genetic algorithm. The phase of ensemble learning was to produce a classifier with good performance by AdaBoost method.

B. Class overlap

Class overlap is that some instances in training data are close to or even overlap in the distribution space but have different class. These instances often lead to poor class boundary, and hinder to build a good learning model with good performance [33]. In the field of SDP, researchers considered class overlap problem as the data quality or noise detection.

Tang et al. [34] proposed a clustering-based noise detection method to remove noisy instances identified by the noise factor. They conducted experiments on NASA with C4.5 learner, and the results indicated that removing noisy instances could improve the accuracy of classifier. Kim et al. [35] evaluated the effect of the dataset containing both false positive and false negative noise, and proposed the closest list noise identification (CLNI) method to identify noise instances. Based on CLNI, Chen et al. [18] proposed neighborhood cleaning learning (NCL) to deal with both class overlap and class imbalance. The experimental results indicated that the new learning models could get best values in terms of *G-mean* and *AUC* compared with state-of-the-art methods.

However, they do not provide a comparison with CPDP models. In our work, we evaluate the impact of class overlap on both WPDP and CPDP.

III. Methodology

In this section, we firstly provide the rationale to our studying questions. Then, we report the experimental data. At last, we introduce the methods under study and model

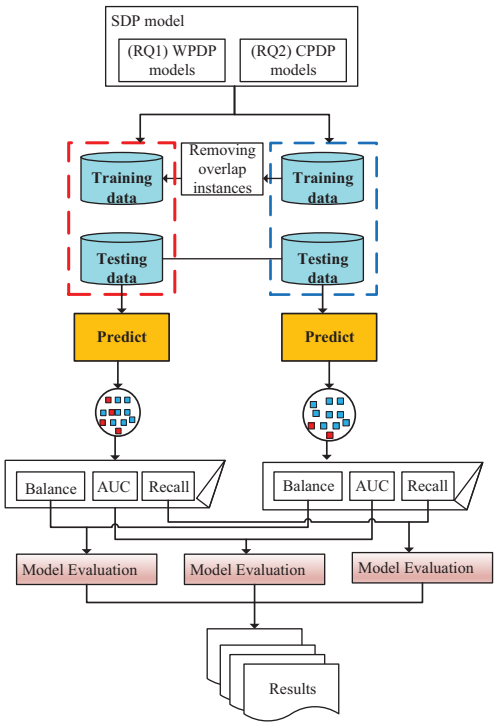

Fig. 2. The overview of our model construction and evaluation approach.

evaluation. Figure 2 provides an overview of the steps in our study.

A. Research questions

Our study is to evaluate the effect of overlapping instances on SDP. To this end, our work checks the following two questions.

RQ1: How does class overlap influence the prediction performance of existing within-project defect prediction models?

RQ2: How does class overlap influence the prediction performance of existing cross-project defect prediction models?

The aim of RQ1 and RQ2 are to compare the performance of the existing state-of-the-art learning models by removing overlapping instances against without removing under two prediction cases (WPDP and CPDP). For RQ1, we study NB [7], [10], RF [21], SVM [23], LR [24] and KNN [22] classifiers for WPDP models. For RQ2, we study NN-filter [28], TCA+ [6], VCB-SVM [31], DTB [29], MNB [30], and HYDRA [32] models for CPDP. Noted that all the above learning models were conducted on Python 3.6 and Scikit-learn (0.19.2). Since our works are to investigate the impact of class overlap on SDP, the parameters of NB, RF, SVM, LR and KNN classifiers are set as default parameters on Scikit-learn (0.19.2), and the parameters of CPDP learning models are set as the same as their studies.

If the learning models under removing overlapping instances are much better than these without removing overlapping instances, it would be a good option for practitioners to employ removing overlapping instances before building the models. In addition, if the performance ranking of learning model is changed, it would be indicated that class overlap problem has a greater impact on this learning model.

B. Datasets

In order to build and validate the effect of class overlap on software defect prediction, we use 28 projects from NASA, AEEEM, ReLink, SOFTLAB and MORPH groups. These datasets collected from these 28 projects includes static code metrics and process metrics, and the defect matching was relied on SZZ algorithm. The metrics along with label information of these datasets are at different level (function or class or file granularity).

Projects from NASA systems are NASA aerospace projects which were gathered as part of the metric data program (MDP). These projects are all only one version and their metric for SDP are function granularity. We only apply CM1, MW1, PC1, PC3 and PC4 projects as they were developed by C language and they have the 37 common metrics.

Projects from SOFTLAB systems are embedded controllers for household appliances from Turkish software company. These projects include AR1, AR3, AR4, AR5 and AR6 developed by C language. The metrics for these projects are also function granularity.

Projects from MORPH systems are collected by Jureczko and Spinellis [15] including 46 releases of 14 open-source projects and 11 student projects. The metrics of these projects are class granularity and developed by Java language. In our experiment, we choose one version for multi-version project, which were used in reference [36].

Projects from AEEEM and ReLink are developed by Java language. AEEEM are collected by D'Ambros et al. [13], and ReLink are collected by Wu et al. [14]. The metrics in AEEEM are class granularity, and the metrics of ReLink are file granularity.

Table I shows the testing projects used in our experiment. From this table, we can observe that (i) the test projects are from different fields and are developed by different languages (C or Java); (ii) the number of instances in the test projects vary greatly and the percentage of defective instances are very low leading to class imbalance. These observations conclude that these test projects could be used to provide a fair evaluation of the impact of class overlap on software defect prediction.

C. Methods to build the models

In the field of SDP, class overlap problem often is considered as the data quality or noise detection. In our experiment, we use KMCCA [34], NCL [18] and our proposed IKMCCA methods to remove the overlapping instances.

CLNI approach was presented by Kim [35] and used by Chen simultaneously [18]. Chen et al. considered both class imbalance and class overlap in the same time. So in our experiment, we used the NCL proposed by Chen. They used

TABLE I
THE EXPERIMENTAL DATASETS.

Group	Project	Langeage	granularity	Number of metrics	Number of total instances	% of defective instances
NASA	CM1	C	function	37	327	12.84
	MW1				253	10.67
	PC1				705	8.65
	PC3				1077	12.44
	PC4				1458	12.21
SOFTLAB	AR1			29	121	7.44
	AR3				63	12.70
	AR4				107	18.69
	AR5				36	22.22
	AR6				101	14.85
AEEEM	Equinox Framework (EQ)	Java	Class	61	324	39.81
	Eclipse JDT core (JDT)				997	20.66
	Apache Lucence (LC)				691	9.26
	Mylyn (ML)				1862	13.16
	Eclipse PDE UI (PDE)				1497	13.96
ReLink	Apache HTTP Server		File	26	194	50.52
	OpenIntents Safe				56	39.29
	ZXing				399	29.57
MORPH	ant1.3		Class	20	125	16
	camel1.0				339	3.83
	poi1.5				237	59.49
	tomcat				858	8.97
	velocity1.4				196	75
	Xalan2.4				723	15.21
	xerces1.2				440	16.14
	arc				234	11.54
	redktor				176	15.34
	skarbonka				45	20

the neighborhood cleaning learn (NCL) rule to remove the overlapping instances. It is noted that the non-defective instances only be removed to stress the class imbalance problem. This is to say, they searched the nearest neighbors of each defective instance and removed the nearest neighbors whose class were non-defective.

KMCCA approach was presented by Tang [34], and this was a clustering-based noise detection approach based on *K*-means. Firstly, the K-means algorithm was used to cluster the data into *K* clusters. Then, for each cluster, the noise factor values of each instance are computed. Finally, the top $p\%$ of instances were removed.

In order to the same as the NCL, we improved the KMCCA method called IKMCCA that considers the class imbalance at the same time. In step of the removing overlapping instances, the rule is based on the percentage of defective instances. If the percentage of defective instances in the i^{th} cluster is below the $p\%$, the defective instance in this cluster are removed. On the contrary, the non-defective instances in this cluster are removed. Pseudo-code for IKMCCA [1] is presented in algorithm 1.

Noted that we employ the log-transformation before using the datasets which was studied by previous researches [37]. The pseudo-code for the simulation experiments are provided in algorithm 2 for evaluating the impact of class overlap to answer RQ1 and RQ2.

[1]https://github.com/glnmzx888/class-overlap

TABLE II
PSEUDO-CODE FOR IKMCCA.

Algorithm 1 Pseudo-code for IKMCCA

Inputs: Traning data D
the parameter m
n= the number of instances in D
d= the number of defective instances in D
p= $\frac{d}{n}$
k=$\lfloor \frac{n}{m} \rfloor$
Using K-means algorithm to divide D into k clusters
for i=$1->k$ **do**:
  Compute the ratio r of defective instances to all instances in cluster i
  if $r > p$:
    delete the non-defective instances in cluster i
  else:
    delete the defective instances in cluster i
  end if
end for
Combine the remaining instances in each cluster

D. Model Evaluation

In order to investigate the impact of class overlap on learning models, we use three performance measures including *Balance* [30], *Recall* and Area Under the receiver operating characteristic Curve (*AUC*) [38].

Balance (*bal*) is the balance between *pd* and *pf*. The bigger the *bal* value is, the better the performance of learning model

TABLE III
PSEUDO-CODE FOR THE EXPERIMENTAL SETUP.

Algorithm 2 Pseudo-code for the experimental setup

Inputs: DATA1={CM1, MW1, PC1, PC3, PC4}
DATA2={AR1, AR3, AR4, AR5, AR6};
DATA3={EQ, JDT, LC, ML, PDE};
DATA4={Apache, Safe, ZXing};
DATA5={ant1.3, camel1.0, poi1.5, tomcat, velocity1.4, Xalan2.4, xerces1.2, arc, redktor, skarbonka};
learning model={WPDP:{NB, SVM, KNN, RF, LR}, CPDP:{NN-filter, TCA+, VCB-SVM, DTB, MNB, HYDRA}}
DATA is DATA1 or DATA2 or DATA3 or DATA4 or DATA5
methods=KMCCA, IKMCCA, NCL

```
for data in DATA do:
  for i=1−>20 do:
    if WPDP:
      {WPTrain=Select 85% of data
      Test=data-WPTrain}
    end if
    if CPDP:
      if Mixed-project:
        {WP=Select 15% of a project,
         Test=project-WP
        CPTrain=data-project+WP}
      else:
        {Test=a project in data
        CPTrain=data-project}
      end if
    end if
    employ log-trainsform on WPTrain, CPTrain, Test
    employ methods:KMCCA, IKMCCA, NCL on WPTrain
     and CPTrain to remove overlapping instances
    for model in learning model:
      Apply the learning model to train on WPTrain
      or CPTrain
      Apply the model to predict the class on Test and
      report the perftomance: Balance, Recall, AUC
      on Test for each learning model
    end for
  end for
end for
```

is. It is defined as

$$bal = 1 - \frac{\sqrt{(1-pd)^2 + pf^2}}{\sqrt{2}}. \quad (1)$$

Recall is the ratio of the number of correctly predicted defective instances to the total number of defective instances. The bigger the *Recall* value is, the better the performance of learning model is. It is defined as

$$Recall = \frac{TP}{TP + FN}. \quad (2)$$

AUC is the area under a Receiver Operating Characteristic (ROC) curve. The *x-axis* of ROC is *pf* and *y-axis* of ROC is *pd*. The bigger the *AUC* value is, the better the performance of learning model is.

In these measures, *pd* (probability of detection) is the ratio of correctly predicted defective instances to total defective instances. *pf* is the fraction of incorrectly predicted defective instances to total non-defective instances.

In order to make a statistical evaluation of the detailed prediction results, the non-parametric Friedman test with the Nemenyi test is used to compare multiple runs over the 28 projects. The confidence level is 95%. They are widely used in software defect prediction (SDP) [39], [40]. Firstly, the Friedman test is used to determine whether there are statistically significant differences among compared methods. If there are significant differences in statistical data, the differences will be test by post-hoc Nemenyi test.

In addition, to evaluate the degree of difference among the compared methods in terms of *bal*, *Recall* and *AUC* results, we apply Cliff's Delta to measure the effect size, which don't need the compared vectors meet normality assumption. This is calculated as followed [41]:

$$Cliff's \quad Delta = \frac{\#(x_1 > x_2) - \#(x_1 < x_2)}{n_1 n_2}. \quad (3)$$

Where x_1 and x_2 are scores within group 1 (performance values achieved by one compared method) and group 2 (performance values achieved by another compared method), and n_1 and n_2 are the sizes of the instances groups. The cardinality symbol # indicates the number.

The effect size of Cliff's Delta is divided into four levels: $|d| < 0.147$ (Negligible, N), $0.147 \leq |d| < 0.333$ (Small, S), $0.333 \leq |d| < 0.474$ (Medium, M) and $|d| \geq 0.474$ (Large, L).

IV. EXPERMENTAL RESULTS

In this section, we report the experimental results for the comparison of removing overlapping instances with without removing overlapping instances to answer RQ1 and RQ2.

A. RQ1: How does class overlap influence the prediction performance of existing within-project defect prediction models?

For answering the question, we replicate the experiments on the 28 projects by NB, RF, SVM, KNN and LR classifiers using IKMCCA VS. NCL VS. KMCCA VS. without removing data, respectively. In IKMCCA method, the percentage *p%* is set as the percentage of defective instances in the training data, and the parameter *m* is set as 20, that will be discussed in section V. Parameters of KMCCA and NCL methods are set the same as the references [34] and [18]. We randomly sample 85% instances as training data, and the rest instances are as testing data. Noted that all compared methods are performed on Python3.6 and Scikit-learn (0.19.2) in our study. At the same time, we repeat these experiments 20 times to avoid randomness, and report the violin plots of the results for each learning models in figures 3, 4 and 5. The violin plot combines the characteristics of box plot and density plot which could display the distribution and probability density of results. At

the same time, the white dot on each violin represents the medium value.

From these figures, we can observe that using removing the overlapping instances approaches including IKMCCA, NCL and KMCCA can get better medium values of *bal*, *Recall* and *AUC* than these methods without removing overlapping instances on NB, RF, SVM, KNN and LR classifiers. At the same time, these WPDP learning models by using IKMCCA can achieve best *bal*, *Recall* and *AUC* results than these by using NCL and KMCCA for most projects. That is to say, (i) compared with KMCCA, it is better to consider class imbalance when solving class overlap; (ii) compared with NCL, it makes more sense to eliminate both defective instances and non-defective instances.

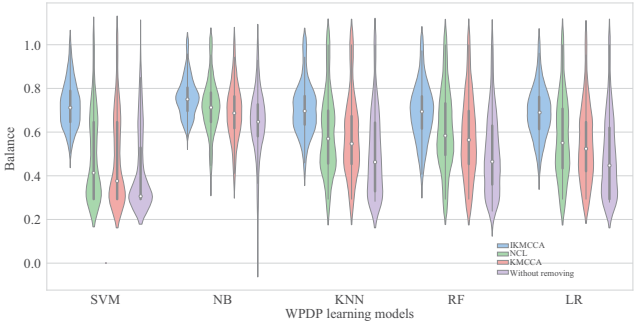

Fig. 3. The violin plot on *bal* for Within-project defect prediction (WPDP) learning models. The white dot on each violin plot represents the medium of each model.

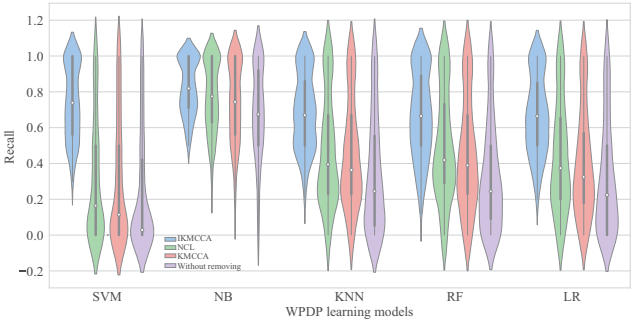

Fig. 4. The violin plot on *Recall* for Within-project defect prediction (WPDP) learning models. The white dot on each violin plot represents the medium value of each model.

Furthermore, in order to statistically investigate the experimental results, we rank the WPDP learning models by non-parametric Fredman test with post-hoc Nemenyi test. Figures 6, 7, 8 show the rank results in terms of *bal*, *Recall* and *AUC*. From these figures we can find that (i) for each WPDP classifier, IKMCCA is always in the first rank, NCL and KMCCA are in same rank, and without removing approach is in the last rank. (ii) Regardless of the different class overlap methods, the NB classifier always ranks the first, and RF,

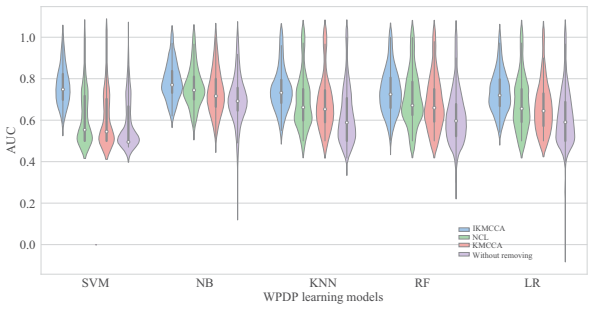

Fig. 5. The violin plot on *AUC* for Within-project defect prediction (WPDP) learning models. The white dot on each violin plot represents the medium of each model.

KNN and LR rank the same. (iii) For SVM classifier, using IKMCCA method can achieve the second in these classifiers, that is to say IKMCCA could greatly improve the predicting performance for SVM.

In addition, in order to make a clearer comparison of these models, we compare the effectiveness between without removing method and each of class overlap methods in all 28 projects based on Cliff's Delta, and the results are shown in table IV.

For IKMCCA method, we can find the values of effect size are large for compared WPDP learning models in terms of *bal*, *Recall* and *AUC*. For NCL methods, the values of effect size in terms of *bal* and *Recall* are most small, and the values of effect size in terms of *AUC* are most Medium. For KMCCA, the values of effect size in terms of *bal*, *Recall* and *AUC* are all small. That is to say, for improving the performance of learning models, considering both class overlap and class imbalance is more effective than only considering class overlap.

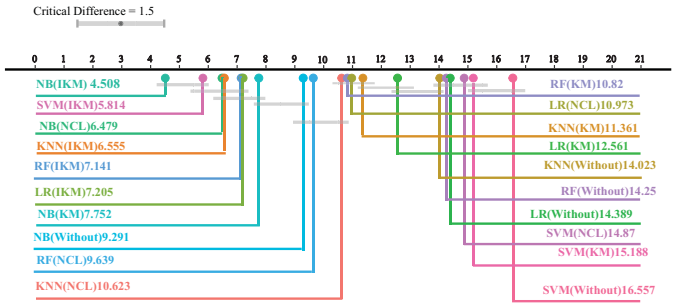

Fig. 6. The ranks on *bal* for within-project defect prediction (WPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

B. RQ2: How does class overlap influence the prediction performance of existing cross-project defect prediction models?

For answering this question, we replicate the experiments on the 28 projects by NN-filter, TCA+, VCB-SVM, DTB, MNB, and HYDRA CPDP learning models using IKMCCA VS.

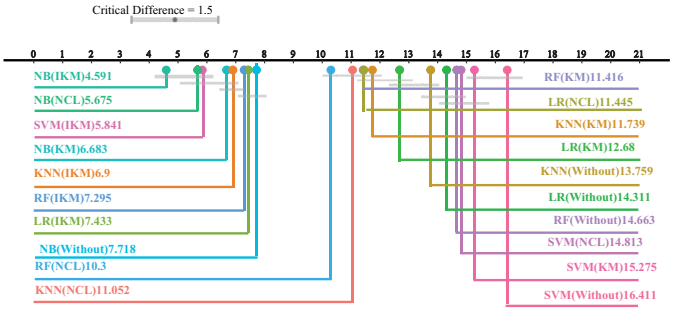

Fig. 7. The ranks on *Recall* for within-project defect prediction (WPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

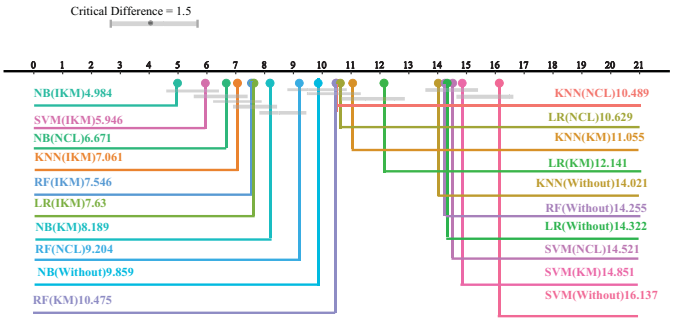

Fig. 8. The ranks on *AUC* for within-project defect prediction (WPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

TABLE IV
THE EFFECTIVENESS RESULTS OF WITHOUT REMOVING AGAINST EACH COMPARED APPROACH FOR WPDP LEARNING MODELS IN TERMS OF *bal*, *Recall*, AND *AUC*.

Learning model	removing VS Without	*bal*	*Recall*	*AUC*
SVM	IKMCCA Vs. Without	0.797(L)	0.769(L)	0.761(L)
	NCL Vs. Without	0.232(S)	0.192(S)	0.206(S)
	KMCCA Vs. Without	0.205(S)	0.161(S)	0.172(S)
NB	IKMCCA Vs. Without	0.523(L)	0.367(M)	0.479(L)
	NCL Vs. Without	0.3956(M)	0.198(S)	0.338(M)
	KMCCA Vs. Without	0.263(S)	0.168(S)	0.168(S)
KNN	IKMCCA Vs. Without	0.645(L)	0.614(L)	0.718(L)
	NCL Vs. Without	0.298(S)	0.258(S)	0.334(M)
	KMCCA Vs. Without	0.269(S)	0.233(S)	0.299(S)
RF	IKMCCA Vs. Without	0.617(L)	0.636(L)	0.745(L)
	NCL Vs. Without	0.357(M)	0.336(M)	0.398(M)
	KMCCA Vs. Without	0.257(S)	0.24(S)	0.292(S)
LR	IKMCCA Vs. Without	0.651(L)	0.621(L)	0.741(L)
	NCL Vs. Without	0.357(M)	0.236(S)	0.334(M)
	KMCCA Vs. Without	0.205(S)	0.179(S)	0.234(S)

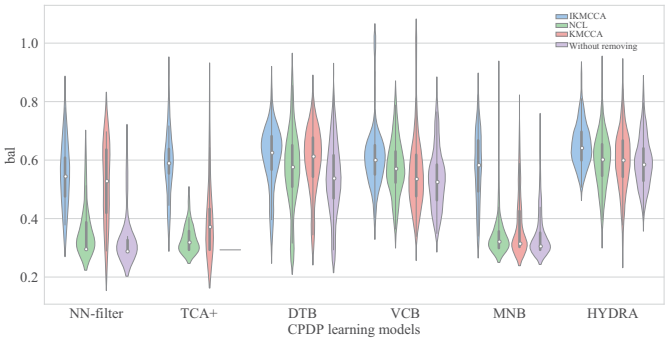

Fig. 9. The violin plot on *bal* for cross-project defect prediction (CPDP) learning models. The white dots on each violin plot represents the medium value of each model.

NCL VS. KMCCA VS. without removing data, respectively. These CPDP learning models are introduced in section II.A. In IKMCCA method, the percentage p is set as the percentage of defective instances in the training data, and the parameter m is set as 20, that will be discussed in section V. Parameters of KMCCA and NCL methods are set the same as the references [34] and [18]. Note that the implementation of all compared methods is based on Python3.6 and Scikit-learn (0.19.2) in our experiments. In order to avoid randomness, we repeat the above experiments 20 times and report the violin plots of the results for each learning models in figures 9, 10, 11.

From these figures, we can observe that the CPDP learning models using IKMCCA, NCL and KMCCA can obtain the better median values than these models without removing overlapping instances in terms of *bal*, *Recall* and *AUC*. At the same time, the high density values getting by IKMCCA, NCL and KMCCA are better than that without removing overlapping instances.

Furthermore, in order to statistically investigate the experimental results, we apply the non-parametric Fredman test with post-hoc Nemenyi test to rank the CPDP learning models. Figures 12, 13, 14 show the rank results in terms of *bal*, *Recall* and *AUC*.

From these figures, we can find that (i) for all CPDP learning models, removing overlapping instances including IKMCCA, NCL and KMCCA can get significantly better than these without removing method, and IKMCCA ranks better than NCL and KMCCA; (ii) for class overlapping methods, HYDRA, DTB and VCB always rank better than NN, TCA+ and MNB; (iii) the performance ranking of TCA+ learning model by removing overlapping instances becomes better than without removing, it would be indicated that class overlap has a greater impact on TCA+.

In addition, to give a clearer comparison of the methods, we compare the effect size over all 28 projects between without removing method and each of class overlap methods according to Cliff's Delta, and the results are shown in table V.

For IKMCCA method, we can find the values of effect size are large or medium for compared CPDP learning models in terms of *bal*, *Recall* and *AUC*. For NCL methods, the values of effect size in terms of *bal*, *Recall* and *AUC* are most small except for TCA+. For KMCCA, the values of effect size in terms of *bal*, *Recall* and *AUC* for NN-filter and TCA+ models are all large, and the values of effect size for other models are

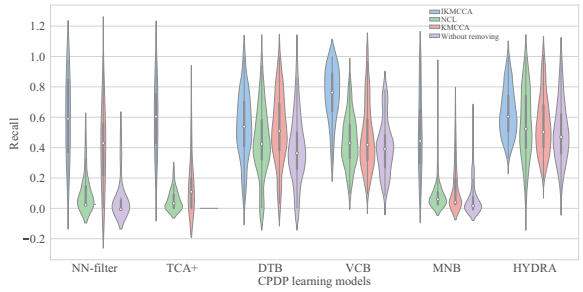

Fig. 10. The violin plot on *Recall* for cross-project defect prediction (CPDP) learning models. The white dots on each violin plot represents the medium value of each model.

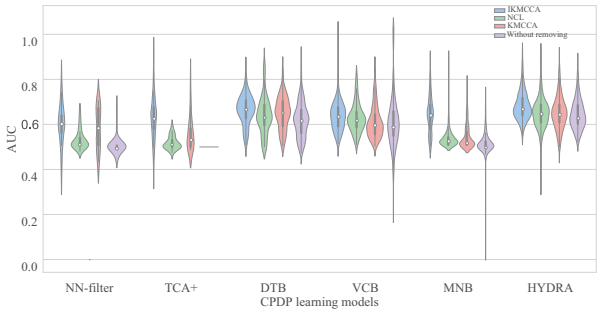

Fig. 11. The violin plot on *AUC* for cross-project defect prediction (CPDP) learning models. The white dots on each violin plot represents the medium value of each model.

small.

C. Answers to research questions

- RQ1: How does class overlap influence the prediction performance of existing within-project defect prediction models?
  This work is to investigate whether it is feasible to removing overlapping instances to improve the defect predicting performance of the WPDP learning models. From the experimental results, we observe a statistically significant improvement in favor of removing overlapping instances including IKMCCA, NCL and KMCCA. It is useful to consider removing the overlapping instances before building the WPDP defect prediction models, in particular, using SVM as the classifier. IKMCCA and NCL can achieve better results than KMCCA for WPDP classifiers. Therefore, it is better to consider class imbalance when dealing with class overlap. Also, IKMCCA can achieve better than NCL, it can not only consider the overlapping non-defective instances when handling both class overlap and class imbalance for WPDP.
- RQ2: How does class overlap influence the prediction performance of existing cross-project defect prediction models?

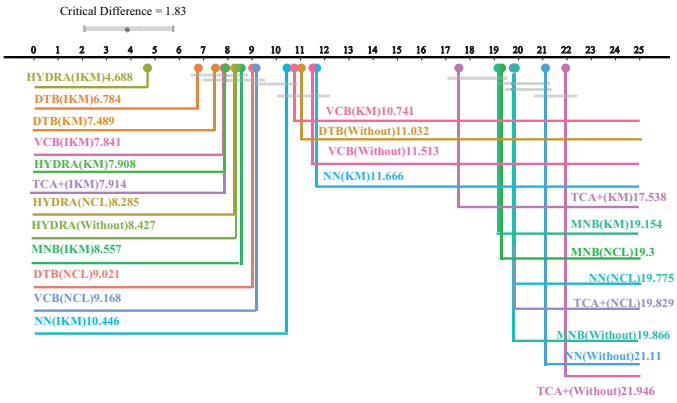

Fig. 12. The ranks on *bal* for within-project defect prediction (CPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

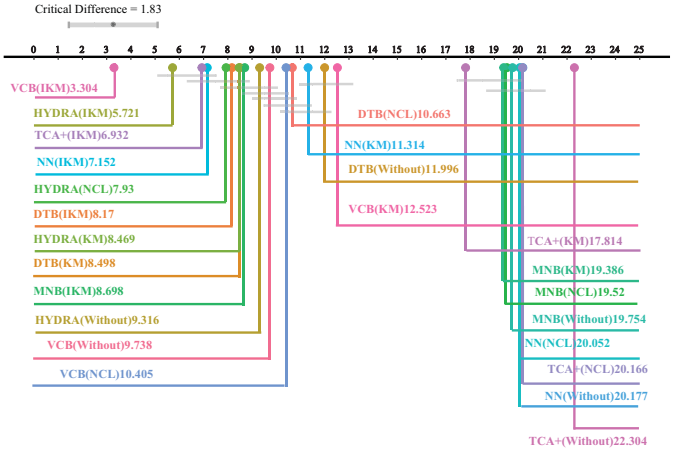

Fig. 13. The ranks on recall for within-project defect prediction (CPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

This work is to investigate whether it is feasible to removing overlapping instances to improve the defect predicting performance of the CPDP learning models. In our experiments, we observe statistically significance results in terms of *bal*, *Recall* and *AUC*. However, we observe the values of effect size of IKMCCA for all models are large, on the contrary, the values of effect size of NCL and KMCCA for most CPDP models are small. IKMCCA and NCL can achieve better results than KMCCA for CPDP learning models. Therefore, it is better to consider class imbalance when dealing with class overlap and using IKMCCA to remove overlapping instances before building the CPDP learning models. At the same time, IKMCCA can achieve better results than NCL, it can not only consider the overlapping non-defective instances when handling both class overlap and

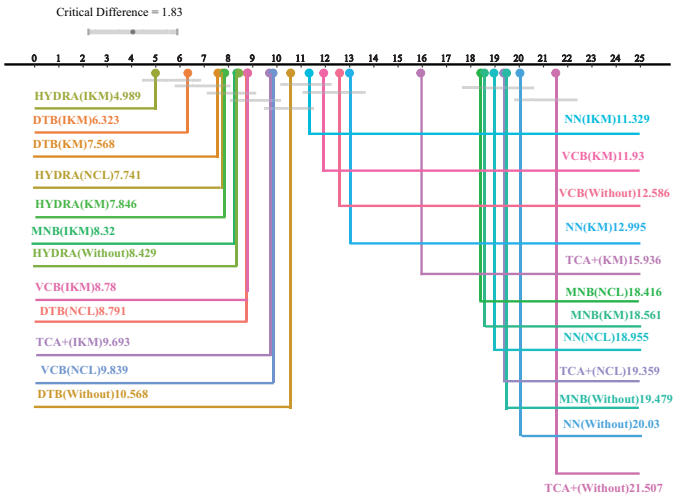

Fig. 14. The ranks on *AUC* for within-project defect prediction (CPDP) methods with post-hoc Nemenyi test. Methods connected by gray lines are not significantly different.

TABLE V
THE EFFECTIVENESS RESULTS OF WITHOUT REMOVING AGAINST EACH COMPARED APPROACH FOR CPDP LEARNING MODELS IN TERMS OF *bal*, *Recall*, AND *AUC*.

Learning model	removing VS Without	*bal*	*Recall*	*AUC*
NN-filter	IKMCCA Vs. Without	0.88(L)	0.932(L)	0.747(L)
	NCL Vs. Without	0.357(M)	0.146(N)	0.164(S)
	KMCCA Vs. Without	0.753(L)	0.795(L)	0.503(L)
TCA+	IKMCCA Vs. Without	1(L)	1(L)	0.857(L)
	NCL Vs. Without	0.5(L)	0.535(L)	0.489(L)
	KMCCA Vs. Without	0.642(L)	0.642(L)	0.535(L)
DTB	IKMCCA Vs. Without	0.389(M)	0.387(M)	0.38(M)
	NCL Vs. Without	0.174(S)	0.234(S)	0.234(S)
	KMCCA Vs. Without	0.327(M)	0.333(M)	0.261(S)
VCB	IKMCCA Vs. Without	0.327(S)	0.633(L)	0.296(S)
	NCL Vs. Without	0.156(S)	0.011(N)	0.198(S)
	KMCCA Vs. Without	0.009(N)	0.309(S)	0.0248(N)
MNB	IKMCCA Vs. Without	0.897(L)	0.903(L)	0.867(L)
	NCL Vs. Without	0.016(N)	0.093(N)	0.158(S)
	KMCCA Vs. Without	0.046(N)	0.114(S)	0.138(S)
HYDRA	IKMCCA Vs. Without	0.342(M)	0.333(M)	0.334(M)
	NCL Vs. Without	0.083(N)	0.148(S)	0.023(N)
	KMCCA Vs. Without	0.054(N)	0.061(N)	0.0262(N)

class imbalance for CPDP, and this is the same conclusion as WPDP.

V. DISCUSSION

In previous section, we observe that removing the overlapping instances including IKMCCA, NCL and KMCCA can achieve statistically better than that without removing for WPDP and CPDP learning models. As IKMCCA could achieve better results than KMCCA and NCL. Therefore, we will discuss the performance of IKMCCA in this section.

A. Performance with different p of IKMCCA

In this experiment, we investigate the impact of the parameter p to IKMCCA, we change the value of it in the range of $\{\frac{1}{2}, \frac{1}{3}, \cdots, \frac{1}{\alpha}, \frac{1}{\alpha+1}, \frac{1}{\alpha+2}\}$, where α is the ratio of all instances to defective instances in training data. To avoid the randomness, we repeat 20 times and report the average values of *bal*, *Recall* and *AUC* on PC4 project: $\{CM1, MW1, PC1, PC3\} => PC4$. Figures 15, 16 and 17 show the results for compared WPDP and CPDP learning models on different p. Considering various factors, we can observe that the parameter p based on the fixed ratio of the number of defective instances to all instances can obtain better performance for most learning models.

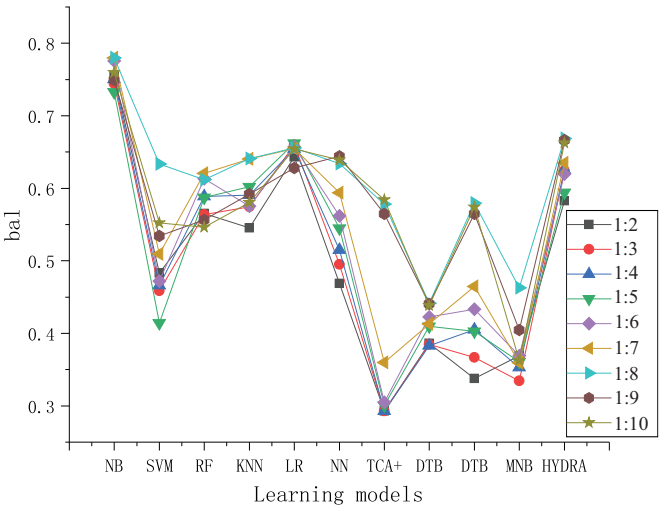

Fig. 15. *bal* values on different p for WPDP and CPDP learning models.

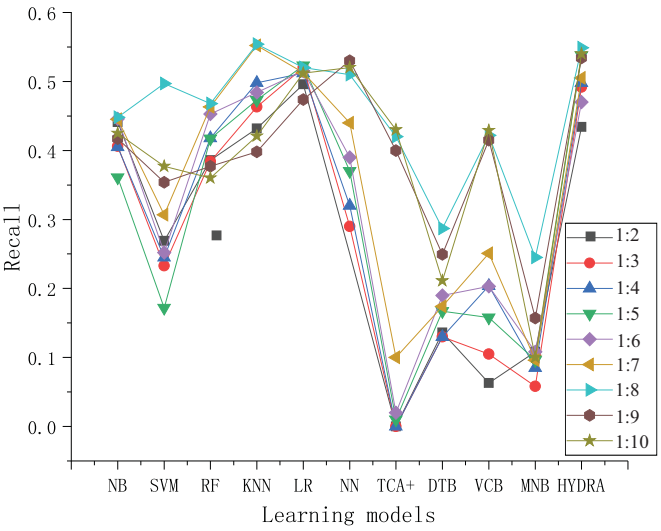

Fig. 16. *Recall* values on different p for WPDP and CPDP learning models.

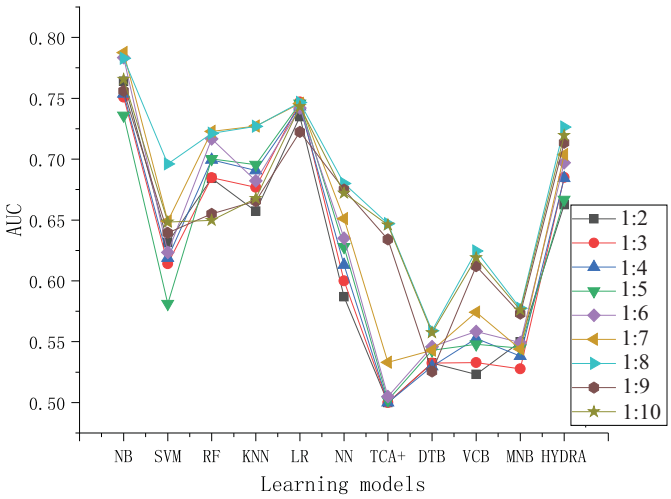

Fig. 17. *AUC* values on different *p* for WPDP and CPDP learning models.

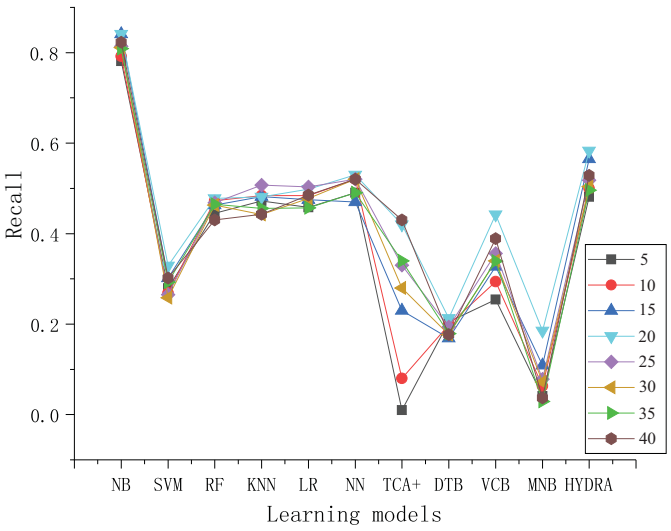

Fig. 19. *Recall* values on different *m* for WPDP and CPDP learning models.

B. Effect of the parameter m for IKMCCA

In this experiment, we investigate the impact of the parameter m to IKMCCA, we change the value of it in the range of $\{5, 10, \cdots, 35, 40\}$. To avoid the randomness, we repeat 20 times and report the average values of *bal*, *Recall* and *AUC* on PC4 project:$\{CM1, MW1, PC1, PC3\} => PC4$. Noted that performance p is fixed on the ratio of the number of defective instances to all instances in this experiments. Figures 18, 19 and 20 show the results for compared WPDP and CPDP learning models on different m. Considering various factors, we can find that the parameter m fixed 20 could obtain better results for most learning models.

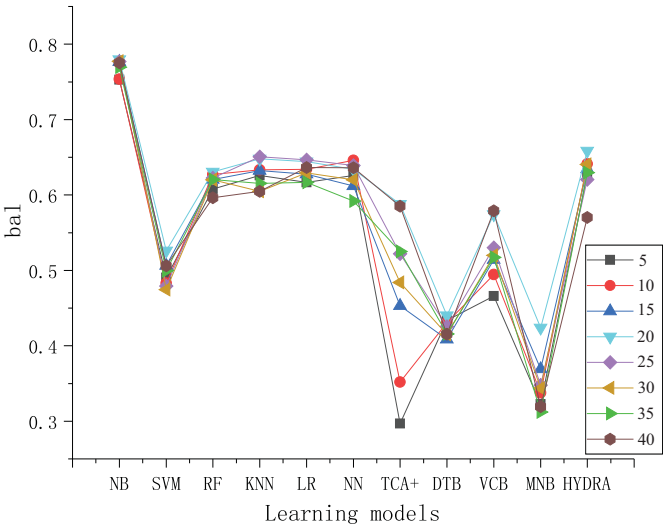

Fig. 18. *bal* values on different *m* for WPDP and CPDP learning models.

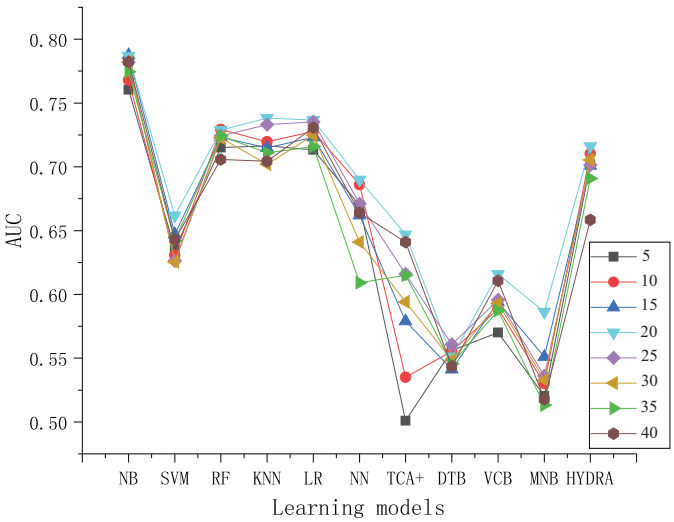

Fig. 20. *AUC* values on different *m* for WPDP and CPDP learning models.

C. Why IKMCCA performs Best

The experimental results in above sections provide substantial empirical support that removing overlapping instances can improve the prediction performance. For all compared learning models, IKMCCA had a superior prediction performance under WPDP and CPDP cases.

The performance of IKMCCA can be attributed to take into account class imbalance factor in removing overlapping instances. In IKMCCA, not only do we remove overlapping instances in non-defective class, we also remove overlapping instances in defective class. For example, if one cluster has more than a certain proportion of non-defective instances, the defective instances would be removed. To demonstrate this idea, we compare with NCL approach only considering

overlapping instances in non-defective instances (introduced in section IV). Figures 3-14 show IKMCCA gain superior performance than NCL for learning models.

In addition, when IKMCCA removes overlapping instances, the certain proportion p based on the class imbalanced ratio is taken into account. This is different from KMCCA only removing top $\%p$ overlapping instances. Figures 3-14 indicate considering the class imbalanced ratio could gain more adaptation training datasets.

VI. Threats to validity

A. Threats to construct validity

Our study in SDP is a binary classification indicating whether one module is defective. The datasets used in our experiments are collected from open-source or proprietary projects, and the defect matching is relied on SZZ algorithm [40], the discovered defective modules may be incomplete, which is a potential threat to construct validity. However, these projects are often used in SDP studies previously, we also assume that the defect matching criteria is the best extent possible.

B. Threats to Internal validity

The WPDP and CPDP learning models that we select are not all WPDP and CPDP studies. To minimize this threat, we select the learning models based on the existing systematic literature reviews that provided a comparative study to SDP models. The selected WPDP and CPDP learning models are the benchmark for SDP, therefore, this would minimize this threat in our work.

C. Threats to External validity

The 28 projects used in this study are NASA, ReLink, SOFTLAB, AEEEM and MORPH which are from either open-source projects (ReLink, MORPH and AEEEM) or proprietary projects (NASA and SOFTLAB), and are different in metric set, scales and types. However, they may threaten the generalizable results of closed software projects. In future, we will investigate more commercial software projects to reduce this threat.

VII. Conclusion

Software defect prediction utilizes the learning models to detect the defective modules in software project. Also, the performance of these learning models depend on the quality of training data. Previous studies in the data quality of SDP mainly focus on class imbalance and feature redundancy. However, in practical data collection, some instances with different label may have similar values on some metrics, which results in overlap in feature space. Inspired by this, we propose an improved K-Means clustering cleaning approach (IKMCCA) to remove the overlapping instances, and investigate whether IKMCCA, KMCCA and NCL are feasible to improve defect detection performance for WPDP and CPDP.

We conduct the experiments to compare the performance of state-of-the-art learning models on 28 projects, and the results make clear that using these removing overlapping instances could obtain significantly better performance in terms of *bal*, *Recall* and *AUC*. That is to say, the overlapping instances affect the defect detecting performance of WPDP and CPDP learning models. The effect size of IKMCCA is large, and the effect size of NCL and KMCCA are small or medium for WPDP and CPDP models, so considering class imbalance when dealing with class overlap may achieve better results, meanwhile it can not only consider the overlapping non-defective instances.

In the further works, we will employ removing overlapping methods to more software defect prediction models and some commercial projects to evaluate the performance.

Acknowledgment

The authors would like to thank the anonymous reviewers and editors for suggesting improvements and for their very helpful comments. This work is supported by the National Natural Science Foundation of China under grant No. 61673384, the Natural Science Foundation of Jiangsu Province under grant No. K20181353, Postgraduate Research & Practice Innovation Program of Jiangsu Province under grant No. KYCX19_2168 and Postgraduate Research & Practice Innovation Program of China University of Mining and Technology under grant No.KYCX19_2168.

References

[1] H. Zhimin, P. Fayola, M. T, and Y. Ye, "Learning from open-source projects: An empirical study on defect prediction," in *Proc. IEEE International Conference on Empirical Software Engineering and Measuremen(ESEM'2013)*, Baltimore, MD, USA, Oct. 2013.

[2] H. Zhimin, S. Fengdi, Y. Ye, L. Mingshu, and W. Qing, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[3] L. Ming, Z. Hongyu, W. Rongxin, and Z. Zhihua, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.

[4] M. Ying, L. Guangchun, Z. Xue, and C. Aiguo, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[5] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proc. International Conference on Automated Software Engineering (ASE)*, Lawrence KS, Nov. 2011.

[6] J. Nam, Sinno, J. Pan, and S. Kim, "Transfer defect learning," ser. Proc. International Conference on Software Engineering (ICSE 2013), San Francisco, November 06–10, 2011.

[7] B. Turhan, T. Menzies, B. Ayse, and D. Justin, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[8] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter language reuse," in *Proc. International Conference on Software Engineering (ICSE 2008)*, Leipzig, May 2008.

[9] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "Multi: Multi-objective effort-aware just-in-time software defect prediction," *Information and Software Technology*, vol. 93, no. 5, pp. 1–13, 2018.

[10] T. Menzies, A. Dekhtyar, and J. Distefano, "Problems with precision: a response to comments on."

[11] G. Blanchard and R. Loubere, "High-order conservative remapping with a posteriori MOOD stabilization on polygonal meshes," 2015, https://hal.archives-ouvertes.fr/hal-01207156, the HAL Open Archive, hal-01207156. Accessed January 13, 2016.

[12] M. Shepperd, Q. Song, and Z. Sun, "Data quality: some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[13] D. Marco, L. Michele, and R. Romain, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Commun Comput Phys*, vol. 17, no. 4–5, pp. 531–577, 2012.

[14] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: recovering links between bugs and changes," in *Proc. International Conference on the Foundations of Software Engineering and European Software Engineering (SIGSOFT/FSE 2011)*, Szeged Hungary, Sep. 2011.

[15] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets," in *Proc. IEEE International Conference on Automated Software Engineering (ASE'2015)*, Lincoln, NE, USA, Nov. 2015.

[16] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: the use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering.*, vol. 40, no. 6, pp. 603–616, 2014.

[17] Z. Zheng, X. Wu, and R. Srihari, "Feature selection for text categorization on imbalanced data," *ACM SIGKDD Explorations Newsletter.*, vol. 6, no. 1, pp. 80–89, 2004.

[18] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal.*, vol. 26, no. 01, pp. 97–125, 2018.

[19] X. Jing, S. Ying, S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proc. IEEE International Conference on Software Engineering(ICSE'2014)*, Hyderabad, India, May 2014, pp. 414–423.

[20] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering.*, vol. 17, no. 04, pp. 375–407, 2010.

[21] D. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *International Conference on New Trends in Computing Sciences (ICTCS)*, Amman, Jordan, October 2017, pp. 252–257.

[22] L. Hribar and D. Duka, "Software component quality prediction using knn andfuzzy logic," *Information and Software Technology.*, vol. 58, no. 2, pp. 388–402, 2010.

[23] L. Miao, M. Liu, and D. ZHang, "Cost-sensitive feature selection with application in software defect prediction," in *Proc. IEEE International Conference on Pattern Recognition(ICPR'2012)*, ICPR, Nov. 2012, pp. 967–970.

[24] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Information and Software Technology.*, vol. 27, no. 0, pp. 504–518, 2015.

[25] A. Panichella, R. Oliveto, and L. AD, "Cross-project defect prediction models: L'union fait la force," in *Proc. IEEE International Conference on Software Maintenance, Reengineering, and Reverse Engineering(CSMR-WCRE'2014)*, Antwerp, Belgium, Feb. 2014, pp. 164–173.

[26] S. Herbold and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering.*, vol. 44, no. 09, pp. 811–833, 2018.

[27] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, and Y. Zhao, "How far we have progressed in the journey? an examination of cross-project defect prediction," *ACM Transactions on Software Engineering and Methodology.*, vol. 27, no. 01, pp. 1–51, 2018.

[28] B. Turhan, A. Bener, and J. Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering.*, vol. 14, no. 05, pp. 540–578, 2009.

[29] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology.*, vol. 62, no. 01, pp. 67–77, 2015.

[30] B. Turhan, A. Misirli, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information and Software Technology.*, vol. 55, no. 06, pp. 1101–1118, 2013.

[31] D. Ryu and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Software Engineering.*, vol. 21, no. 01, pp. 43–71, 2016.

[32] X. Xia, D. Lo, S. Pan, and N. Nagappan, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering.*, vol. 42, no. 10, pp. 977–998, 2016.

[33] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering.*, vol. 38, no. 2, pp. 425–438, 2012.

[34] T. Tang, W amd Khoshhgoftaar, "Noise identification with the k-means algorithm," in *Proc. IEEE International Conference on Tools with Artificial Intelligence(ICTAI '2004)*, Boca Raton, FL, USA, USA, Nov. 2004, pp. 373–378.

[35] S. Kim, T. Zimmemann, and A. Zeller, "Dealing with noise in defect prediction," in *Proc. IEEE International Conference on Software Engineering(ICSE'2011)*, Honolulu, HI, USA, May 2011, pp. 481–490.

[36] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proc. IEEE International Conference on Mining Software Repositories(MSR'2013)*, San Francisco, CA, USA, May 2013, pp. 409–418.

[37] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *Proc. ACM International Conference on the 15th Annual Conference on Mining software repositories(MSR'2005)*, Louis, Missouri, May 2005, pp. 1–5.

[38] J. A, Hanley, B. J, and McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve," *Radiology.*, vol. 143, no. 01, pp. 29–36, 1982.

[39] D. Marco, L. Michele, and R. Romain, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Commun Comput Phys*, vol. 17, no. 4–5, pp. 531–577, 2012.

[40] H. Steffen, T. Alexander, and G. Jens, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, 2018.

[41] G. Macbeth, E. Razumiejczyk, and R. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychological*, vol. 10, no. 2, pp. 545–555, 2011.