Combing Data Filter and Data Sampling for Cross-Company Defect Prediction: An Empricial Study

Xiao Yu[1,2,3], Man Wu[2,3], Yan Zhang[2,3*], Mandi Fu[4]
[1]State Key Lab. of Software Engineering, Computer School, Wuhan University, Wuhan, China
[2]School of Computer Science and Information Engineering, HuBei University, Wuhan, China
[3]Educational Informationalization Engineering Research Center of HuBei Province, Wuhan, China
[4]Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China
*Corresponding author email: zhangyan@hubu.edu.cn

***Abstract*—Cross-company defect prediction (CCDP) is a practical way that trains a prediction model by exploiting one or multiple projects of a source company and then applies the model to target company. Unfortunately, larger irrelevant cross-company (CC) data usually makes it difficult to build a prediction model with high performance. On the other hand, the CC data has the highly imbalanced nature between the defective-prone and non-defective classes, which will degrade the performance of CCDP. To address such issues, this paper proposes an approach, in which data sampling is combined with data filter, to overcome these problems. Data sampling seeks a more balanced dataset through the addition or removal of instances, while data filter is a process of filtering out the irrelevant CC data so that the performance of CCDP models can be improved. We employ two data filtering methods called NN filter and DBSCAN filter combined with SMOTE (Synthetic Minority Oversampling Technique) and RUS (Random Under-Sampling). Eight different approaches would be produced when combing these four techniques: 1- NN filter performed prior to RUS; 2- NN filter performed after RUS; 3- NN filter performed prior to SMOTE; 4- NN filter performed after SMOTE; 5- DBSCAN filter performed prior to RUS; 6- DBSCAN filter performed after RUS; 7- DBSCAN filter performed prior to SMOTE; 8- DBSCAN filter performed after SMOTE. The empirical study was carried out on 15 publicly available project datasets. The experimental results demonstrate that NN filter performed prior to RUS (Approach 1) performs better than the other seven approaches.**

Keywords—software defect prediction;cross-company defect prediction;data sampling; data filter

I. INTRODUCTION

Software defect prediction is one of the most important software quality assurance techniques. It aims to detect the defect proneness of new software modules via learning from defect data. So far, many efficient software defect prediction approaches [1-6] have been proposed, but they are usually confined to within project defect prediction (WPDP). WPDP works well if sufficient data is available to train a defect prediction model. However, it is difficult for a new project to perform WPDP if there is limited historical data. Cross-company defect prediction (CCDP) is a practical approach to solve the problem. It trains a prediction model by exploiting one or multiple projects of a source company and then applies the model to target company [7]. In recent years, most existing CCDP approaches have been proposed. The challenges of building a CCDP model with high performance usually include:

(1) How to weaken the impact of irrelevant CC data to improve the performance of CCDP.

The ability to transfer knowledge from a source company to a target company depends on how they are related. The stronger the relationship, the more usable will be CC data. The performance of CCDP is generally poor because of larger irrelevant CC data. Previous work [8] founds that using raw CC data directly would increase false alarm rates due to irrelevant instance in CC data, so several data filtering works should be done before building the prediction model. For example, Turhan et al. [8] and Peters et al. [9] proposed the NN filter and the Peters filter to select the CC instances which are mostly similar to WC data as the training dataset.

(2) How to cope with the class imbalance problem to improve the performance of CCDP.

The CC data has the highly imbalanced nature, because the number of non-defective instances is usually larger than the number of non-defective ones. The class imbalance problem may cause difficulties for learning, as most classification algorithms only perform optimally when the number of instances of each class is roughly the same. When these algorithms are trained by a highly skewed dataset in which the minority class is heavily outnumbered by the majority class, these classifiers tend to favor the majority class and have less ability to classify the minority class. Therefore, several data sampling works should be done before building the CCDP model. For example, Lin et al. [10] introduced a novel CCDP approach named Double Transfer Boosting (DTB). DTB firstly uses NN filter to filter out irrelevant CC data, and then uses SMOTE algorithm [11] to re-sample the CC data before building the CCDP model.

However, it is still unclear what extent combining data filter and data sampling can contribute to CCDP, and how to make better use of them to improve CCDP. To address such issues, this paper proposes an approach, in which data sampling is combined with data filter, to cope with both class imbalance problem and the influence of irrelevant CC data. A question may arise when we combine the two techniques, that is, which technique, data filter or data sampling, should be performed first? To answer the question, we employ two data filtering methods called NN filter [8] and DBSCAN filter [12] combined with SMOTE [11] and RUS [13]. We investigate

DOI reference number: 10.18293/SEKE2017-134

eight different approaches: 1- NN filter performed prior to RUS; 2- NN filter performed after RUS; 3- NN filter performed prior to SMOTE; 4- NN filter performed after SMOTE; 5- DBSCAN filter performed prior to RUS; 6- DBSCAN filter performed after RUS; 7- DBSCAN filter performed prior to SMOTE; 8- DBSCAN filter performed after SMOTE. To our best knowledge, no study has been done for combining data filter with data sampling and investigating the eight approaches in the domain of CCDP.

The empirical study was carried out over 15 publicly available projects, all of which exhibit a high degree of class imbalance between the defective-prone and non-defective classes. Three different learners were used to build CCDP models. The experimental results demonstrate that NN filter performed prior to RUS (Approach 1) has significantly better performance than the other seven approaches.

The remainder of this paper is organized as follows. Section II presents the related work. Section III describes two data filtering methods, two data sampling methods, and eight combination approaches. Section IV demonstrates the experimental results. Section V discusses the potential threats to validity. Finally, Section VI addresses the conclusion and points out the future work.

II. Related Work

In this section, we first review the existing cross-company and cross-project defect prediction approaches. Then, we briefly review the data sampling methods.

A. Defect prediction

In order to solve the problem that the new companies have too limited historical data to perform WCDP well, the cross-project and cross-company defect prediction appear. Zimmermann et al. [55] studied CCDP models on 12 real-world applications datasets. Their results indicated that CCDP is still a serious challenge because of the different distribution between the training project data and the target project data. In order to narrow the distribution gap, there are three mainstream ways.

The first one is to apply the data filtering method to find the best suitable training data (e.g., [8, 9, 12, 14]). For example, Turhan et al. [8] proposed a nearest neighbor (NN) filter to select cross-company data. Peters et al. [9] introduced the Peters filter to select training data.

The second mainstream way is to design effective defect predictor based on transfer learning techniques (e.g., [7, 10, 15,16, 17, 18, 19]). For instance, Ma et al. [15] proposed Transfer Naïve Bayes (TNB) model. Chen et al. [10] proposed double transfer boosting (DTB) model. Another challenge in CCDP is that the set of metrics between the source company data and target company data is usually heterogeneous. Jing et al. [7] and Chen et al. [19] proposed the effective solutions for heterogeneous cross-company defect prediction.

The third mainstream way is to apply unsupervised classifier that does not require any training data to perform CCDP (e.g., [20-23]), therefore the distribution gap between the training project data and the target project data is no longer an issue. For instance, Zhang et.al [22] proposed to apply a connectivity-based unsupervised classifier that is based on spectral clustering to perform CPDP.

B. Data sampling

Besides the larger irrelevant instances in CC data, CCDP models also suffer from the class imbalance problem. A considerable amount of research has been done to investigate this problem at data and algorithm levels. Data-level methods include a variety of resampling techniques, manipulating training data to rectify the skewed class distributions, such as random oversampling and random under-sampling. They are simple and efficient, but their effectiveness depends greatly on the problem and training algorithms [24]. Algorithm-level methods address class imbalance by modifying their training mechanism directly with the goal of better accuracy on the minority class, including one-class learning [25], and cost-sensitive learning algorithms [26]. Algorithm-level methods require specific treatments for different kinds of learning algorithms, which hinders their use in many applications, because we do not know in advance which algorithm would be the best choice in most cases. In addition to the aforementioned data-level and algorithm-level solutions, ensemble learning [27] has become another major category of approaches to hand imbalanced data by combining multiple classifiers, such as SMOTEBoost [28], and AdaBoost.NC [29]. Ensemble learning algorithms have been shown to be able to combine strength from individual learners, and enhance the overall performance. They also offer additional opportunities to handle class imbalance at both the individual and ensemble levels. This paper investigates SMOTE and RUS, because of their simplicity, effectiveness, and popularity in the literature.

While a great deal of work has been done for data filter and data sampling separately, limited research has been done and reported on both together, especially in the context of CCDP. Among the few studies, Lin et al. [17] proposed the combination of NN filter and SMOTE to preprocess the CC data before building the CCDP model. However, it is still unclear what extent combining data filter and data sampling can contribute to CCDP, and how to make better use of them to improve CCDP. This is exactly what we will solve in this paper.

III. Methodology

A. Data filter

Previous work [8] founds that using raw CC data directly would increase false alarm rates due to irrelevant instances in CC data, so several data filtering works should be done before building the prediction model. The main goal of data filter is to select the most valuable training data for the CCDP model by filtering out irrelevant instances in CC data. In this study, we employ two representative data filtering methods, NN filter and DBSCAN filter.

The NN filter was proposed by Turhan et al. [8]. Based on the widely used KNN algorithm, NN filter can find out the most similar $k \times n$ instances from CC data while n is the number of WC instances and k is the parameter of the KNN algorithm. The procedure of NN filter is as follows:

1. Find k neighbors from the CC data for each WC instance based on Euclidean distance, and

2. Collect the selected neighbors without duplication into a new CC data.

The DBSCAN filter was proposed by Kawata et al. [12]. It assumes that CC instances which are in the same cluster as WC instances are the most valuable instances in CC data. The procedure of DBSCAN filter is as follows:

1. Combine the CC data and WC data,

2. Find sub-clusters by using DBSCAN algorithm,

3. Select sub-clusters which consist at least one WC instance, and

4. Collect the CC data in the selected sub-clusters into a new CC data.

B. Data sampling

Besides larger irrelevant instances in CC data, CCDP models also suffer from the class imbalance problem. A variety of data sampling techniques have been studied in the literature. The main goal of data sampling is to achieve a certain balance between the defective-prone class and non-defective class. In this study, we employ two representative data sampling methods, SMOTE (Synthetic Minority Oversampling Technique) and RUS (Random Under-Sampling).

SMOTE [11] was proposed by Chawla in 2002. It is an over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples. The procedure of SMOTE is as follows:

1. For each instance in the defective-prone class, calculate the Euclidean distance between it and other instances in the defective-prone class to find its *k* nearest neighbor.

2. According to the amount of over-sampling, determine the sampling rate and select a certain number of instances from *k* nearest neighbor randomly. The sampling rate (between defective-prone and non-defective instances) was set to 50:50 throughout the experiments.

3. Take the difference of the feature vector between it and its nearest neighbor.

4. Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration.

5. Generate new instances for each instance in the defective-prone class and add new samples into it.

RUS (Random Under-Sampling) [13] is a simple method to select a subset of non-defective instances randomly and then combine them with defective-prone instances as a training set.

The procedure of random under-sampling is as follows:

1. Calculate the ratio of the defective-prone class to the non-defective class, and get the sampling frequency.

2. Sample the non-defective class by the sampling frequency.

3. Select all defective-prone instances.

4. Combine selected instances and attributes for training.

C. Eight combination approaches

The primary goal of this study is to evaluate the data preprocessing technique in which data filter is combined with data sampling. Eight different scenarios (also called approaches) would be produced depending on whether data filter is performed before or after data sampling. The eight different approaches are described as follows:

- Approach 1: NN filter performed prior to RUS;
- Approach 2: NN filter performed after RUS;
- Approach 3: NN filter performed prior to SMOTE;
- Approach 4: NN filter performed after SMOTE;
- Approach 5: DBSCAN filter performed prior to RUS;
- Approach 6: DBSCAN filter performed after RUS;
- Approach 7: DBSCAN filter performed prior to SMOTE;
- Approach 8: DBSCAN filter performed after SMOTE.

IV. EXPERIMENTS

In this section, we evaluate the eight combination approaches to perform CCDP empirically. We first introduce the experiment dataset, the performance measures and the experimental procedure. Then, in order to investigate the performance of the eight combination approaches, we perform some empirical experiments to find answers to the research question mentioned above.

A. Data set

In this experiment, we employ 15 available and commonly used datasets which can be obtained from PROMISE. The 15 datasets have the same 20 attributes, so we can apply all attribute information directly. Table 1 tabulates the details about the datasets.

TABLE I. DETAILS OF EXPERIMENT DATASET

Project	***Examples***	***%Defective***	***Description***
ant	125	16	Open-source
arc	234	11.5	Academic
camel	339	3.8	Open-source
elearn	64	7.8	Academic
jedit	272	33.1	Open-source
log4j	135	25.2	Open-source
lucene	195	46.7	Open-source
poi	237	59.5	Open-source
prop	660	10	Proprietary
redaktor	176	15.3	Academic
synapse	157	10.2	Open-source
systemdata	65	13.8	Open-source
tomcat	858	9	Open-source
xalan	723	15.2	Open-source
xerces	162	47.5	Open-source

B. Performance measures

In the experiment, we employ three commonly used performance measures including *pd*, *pf* and *g-measure*. They are defined in Table 2 and summarized as follows.

TABLE II. PERFORMANCE MEASURES

		Actual	
		yes	***no***
Predicted	***yes***	***TP***	***FP***
	no	***FN***	***TN***
pd	$\frac{TP}{TP+FN}$		
pf	$\frac{FP}{FP+TN}$		
G-measure	$\frac{2*pd*(1-pf)}{pd+(1-pf)}$		

- Probability of detection or *pd* is the measure of defective modules that are correctly predicted within the defective class. The higher the *pd*, the fewer the false negative results.

- Probability of false alarm or *pf* is the measure of non-defective modules that are incorrectly predicted within the non-defective class. Unlike *pd*, the lower the *pf* value, the better the results.

- *G-measure* is a trade-off measure that balances the performance between *pd* and *pf*. A good prediction model should have high *pd* and low *pf*, and thus leading to a high *g-measure*.

C. Experimental Procedure

In every experiment, one dataset is selected as WC data and the rest are regarded as CC data to conduct the experiment. The CC data is considered as basic training data which will be adjusted in every experiment. All processing steps (data filter and data sampling) are done on CC data. Then processed CC data are used to build the CCDP model. Finally, the resulting model is evaluated on the WC data. The procedure will be repeated 30 times in every experiment to avoid sample bias. Then, the mean values of performance are calculated.

In this experiment, we choose three representative classifiers as the basic prediction model, Naive Bayes (NB) [33], Random Forest (RF) [34], and Logistic Regression (LR) [35].The reason we choose these classifiers is that these classifiers fall into three different families of learning methods. NB is a probabilistic classifier; RF is a decision-tree classifier; and LR is a linear model for classification.

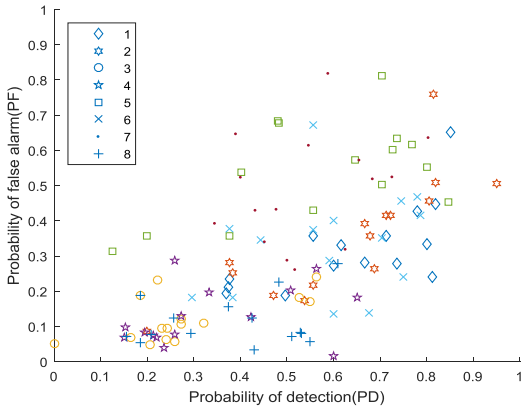
D. Experimental results

Fig. 1 presents the scatter plots of (PD, PF) points from the eight approaches on the fifteen projects. Note that a CCDP approach has more points distributed at bottom right if it has higher PD value and lower PF value. We can gain the following results from Fig. 1.

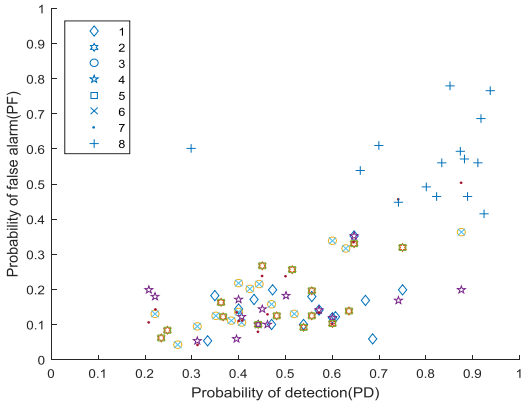
(1) In terms of the defect detection rate (PD), Approach 1 outperforms other approaches for NB model, which shows its effectiveness in finding defects. However, Approach 1 has a little high PF value. In terms of the false alarm rate (PF), although Approach 3 is the best, it performs not well in PD, which makes it hardly useful in practice.

(2)Although Approach 8 appears to be better at PD than other approaches for RF model, it performs the worst in PF. Approach 1 presents generally higher PD and lower PF than other approaches for RF model.

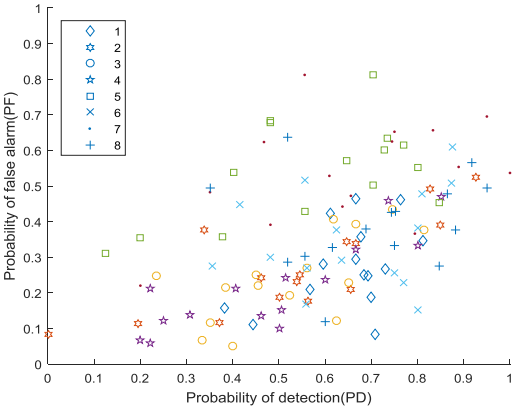
(3) Approach 8 shows better PD than other approaches for LR model, but its advantage is rather limited. Approach 1 presents generally higher PD and lower PF than other approaches for RF model.



(a)NB



(b)RF



(c)LR

Fig. 1. Performances with Scatter plots of (PD, PF) points of the eight approaches on the fifteen projects

Tables 3-5 show the G-measure values for each project on all approaches with three CCDP models.

TABLE III. G-MEASURE PERFORMANCES WITH NAÏVE BAYES

Project	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
ant	**0.64**	0.56	0.53	0.56	0.50	0.57	0.60	0.54
arc	0.66	0.62	0.56	0.56	0.63	**0.65**	0.52	0.63
camel	0.67	0.68	0.54	0.60	0.72	**0.72**	0.54	0.71
elearn	0.71	0.72	0.55	0.72	0.62	**0.75**	0.31	0.72
jedit	0.57	0.52	0.46	0.55	0.59	**0.65**	0.55	0.59
log4j	0.62	0.59	0.60	0.60	0.65	**0.80**	0.58	0.63
lucene	0.64	0.51	0.50	0.54	0.54	**0.67**	0.57	**0.67**
poi	0.49	0.37	0.42	0.47	0.55	**0.62**	0.45	0.57
prop	0.49	**0.61**	0.55	0.60	0.58	**0.66**	0.50	0.59
redaktor	0.61	**0.66**	**0.66**	0.63	0.35	0.45	0.29	0.34
synapse	**0.77**	0.71	0.74	0.63	0.37	0.54	0.65	0.43
system	0.66	0.68	0.57	0.35	0.67	0.65	0.51	**0.71**
tomcat	0.69	**0.73**	0.65	0.69	0.58	0.62	0.45	0.60
xalan	**0.67**	0.66	0.63	0.66	0.47	0.52	0.52	0.51
xerces	**0.51**	0.39	0.35	0.34	0.34	0.48	0.37	0.36
AVG	**0.63**	0.60	0.55	0.57	0.54	**0.62**	0.50	0.57

TABLE IV. G-MEASURE PERFORMANCES WITH RANDOM FOREST

Project	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
ant	**0.73**	0.65	0.66	0.72	0.57	0.60	0.60	0.59
arc	0.51	**0.65**	0.41	0.40	0.58	0.67	0.51	0.31
camel	0.64	0.51	0.37	0.26	**0.66**	0.52	0.54	0.37
elearn	0.66	0.33	0.00	**0.75**	0.31	0.71	0.31	0.50
jedit	**0.69**	0.66	0.47	0.57	0.48	0.53	0.55	0.27
log4j	**0.73**	0.60	0.34	0.38	0.49	0.76	0.58	0.45
lucene	0.51	**0.65**	0.28	0.36	0.51	0.44	0.57	0.34
poi	0.62	**0.65**	0.38	0.35	0.39	0.67	0.45	0.57
prop	**0.67**	0.64	0.38	0.26	0.52	0.65	0.50	0.40
redaktor	**0.50**	0.37	0.30	0.38	0.30	0.41	0.29	0.30
synapse	**0.78**	0.71	0.65	0.64	0.21	0.75	0.55	0.52
system	0.60	**0.64**	0.34	0.47	0.56	0.59	0.51	0.50
tomcat	**0.66**	0.65	0.42	0.42	0.43	0.63	0.55	**0.66**
xalan	**0.66**	0.62	0.64	0.62	0.51	0.63	0.52	0.59
xerces	**0.50**	0.49	0.42	0.32	0.38	0.47	0.37	0.53
AVG	**0.63**	0.59	0.40	0.46	0.46	0.60	0.49	0.46

TABLE V. G-MEASURE PERFORMANCES WITH LOGISTIC REGRESSION

Project	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
ant	**0.74**	0.71	0.71	0.73	0.57	0.70	0.46	0.66
arc	**0.61**	0.52	0.49	0.36	0.58	0.57	0.54	0.60
camel	0.72	0.57	0.52	0.45	0.66	0.77	0.63	**0.78**
elearn	0.73	0.50	0.56	0.33	0.31	**0.82**	0.32	0.71
jedit	0.66	**0.72**	0.63	0.67	0.48	0.48	0.58	0.65
log4j	**0.77**	0.62	0.61	0.64	0.49	0.67	0.70	**0.73**
lucene	0.59	0.63	0.56	0.54	0.51	0.63	0.59	**0.64**
poi	0.53	0.65	0.63	0.60	0.39	0.63	0.50	**0.65**
prop	**0.66**	0.63	0.57	0.61	0.52	**0.66**	0.49	0.65
redaktor	**0.69**	0.63	0.71	0.65	0.30	0.52	0.28	0.43
synapse	0.72	0.67	0.73	0.39	0.21	**0.75**	0.47	0.71
system	0.59	0.66	0.64	0.67	0.56	0.54	0.59	**0.62**
tomcat	0.59	0.32	0.50	0.63	0.43	0.62	0.42	**0.65**
xalan	0.63	0.63	**0.64**	0.62	0.51	0.63	0.53	0.59
xerces	**0.60**	0.44	0.36	0.34	0.38	0.47	0.42	0.41
AVG	**0.66**	0.56	0.59	0.55	0.46	0.63	0.50	0.63

We can gain the following results from Tables 3-5.

(1) On more than half projects, Approach 6 performs better than the other seven approaches for NB model. However, Approach 1 achieves the best average G-measure value for NB model (see Table 4).

(2) On nine projects, Approach 1 achieves higher G-measure than the other seven approaches for RF model. In addition, Approach 1 achieves the best average G-measure value for RF model (see Table 5).

(3) On six projects, Approach 1 achieves higher G-measure than the other seven approaches for LR model, while Approach 8 displayed similar or slightly worse performance than Approach 1. In addition, Approach 1 achieves the best average G-measure value for LR model (see Table 5).

Therefore, we can conclude that NN filter performed prior to RUS (Approach 1) has significantly better performance than the other seven approaches.

V. THREATS TO VALIDITY

In this section, we discuss several validity threats that may have an impact on the results of our studies.

External validity. Threats to external validity occur when the results of our experiments cannot be generalized. As a preliminary result, we performed our experiments on the 15 datasets to answer the research questions. Although these datasets have been widely used in many software defect prediction studies, we still cannot claim that our conclusions can be generalized to other software projects. Nevertheless, this work provides a detail experimental description, including parameter settings (default parameter settings specified by *sklearn*), thus other researchers can easily replicate this empirical study on new datasets.

Internal validity. Threats to internal validity refer to the bias of the choice of CCDP classifiers, data filtering methods and data sampling methods. In this work, we only use three classifiers, Naive Bayes (NB), Random Forest (RF), and Logistic Regression (LR) due to its popularity in defect prediction. In addition, we choose two representative data filtering methods, i.e., NN filter and DBSCAN filter, two representative data sampling methods, i.e., RUS and SOMTE.

Construct validity. Threats to construct validity focus on the bias of the measures used to evaluate the performance of CCDP. In our experiments, we mainly use pd, pf, G-measure to measure the effectiveness of the eight approaches. Nonetheless, other evaluation measures such as AUC measure can also be considered.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose data filter combined with data sampling to overcome the influence of larger irrelevant CC data and class imbalance problems that often affect CCDP models. Eight combination approaches are investigated: 1- NN filter performed prior to RUS; 2- NN filter performed after RUS; 3- NN filter performed prior to SMOTE; 4- NN filter performed after SMOTE; 5- DBSCAN filter performed prior to RUS; 6- DBSCAN filter performed after RUS; 7- DBSCAN filter performed prior to SMOTE; 8- DBSCAN filter performed after SMOTE.

We conduct experiments on the 15 datasets to evaluate the performance of the proposed eight approach. The experimental results indicate that NN filter performed prior to RUS (Approach 1) has significantly better performance than the other seven approaches.

In the future, we would like to employ more datasets to validate the generalization of the derived conclusions [36-37].

ACKNOWLEDGMENT

This work is partly supported by Educational Informationalization Engineering Research Center of HuBei Province.

REFERENCES

[1] Elish K O, Elish M O. Predicting defect-prone software modules using support vector machines[J]. Journal of Systems and Software, 2008, 81(5): 649-660.

[2] Zheng J. Cost-sensitive boosting neural networks for software defect prediction[J]. Expert Systems with Applications, 2010, 37(6): 4537-4543.

[3] Sun Z, Song Q, Zhu X. Using coding-based ensemble learning to improve software defect prediction[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2012, 42(6): 1806-1817.

[4] Wang S, Yao X. Using class imbalance learning for software defect prediction[J]. IEEE Transactions on Reliability, 2013, 62(2): 434-443.

[5] Liu M, Miao L, Zhang D. Two-stage cost-sensitive learning for software defect prediction[J]. IEEE Transactions on Reliability, 2014, 63(2): 676-686.

[6] Jing X Y, Ying S, Zhang Z W, et al. Dictionary learning based software defect prediction[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 414-423.

[7] Jing X, Wu F, Dong X, et al. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 496-507.

[8] Turhan B, Menzies T, Bener A B, et al. On the relative value of cross-company and within-company data for defect prediction[J]. Empirical Software Engineering, 2009, 14(5): 540-578.

[9] Peters F, Menzies T, Marcus A. Better cross company defect prediction[C]//Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE, 2013: 409-418.

[10] Chen L, Fang B, Shang Z, et al. Negative samples reduction in cross-company software defects prediction[J]. Information and Software Technology, 2015, 62: 67-77.

[11] Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: synthetic minority over-sampling technique[J]. Journal of artificial intelligence research, 2002, 16: 321-357.

[12] Kawata K, Amasaki S, Yokogawa T. Improving relevancy filter methods for cross-project defect prediction[M]//Applied Computing & Information Technology. Springer International Publishing, 2016: 1-12.

[13] Tahir M A, Kittler J, Yan F. Inverse random under sampling for class imbalance problem and its application to multi-label classification[J]. Pattern Recognition, 2012, 45(10): 3738-3750.

[14] Herbold S. Training data selection for cross-project defect prediction[C]//Proceedings of the 9th International Conference on Predictive Models in Software Engineering. ACM, 2013: 6.

[15] Ma Y, Luo G, Zeng X, et al. Transfer learning for cross-company software defect prediction[J]. Information and Software Technology, 2012, 54(3): 248-256.

[16] Nam J, Pan S J, Kim S. Transfer defect learning[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 382-391.

[17] Chen L, Fang B, Shang Z, et al. Negative samples reduction in cross-company software defects prediction[J]. Information and Software Technology, 2015, 62: 67-77.

[18] Ryu D, Choi O, Baik J. Value-cognitive boosting with a support vector machine for cross-project defect prediction[J]. Empirical Software Engineering, 2016, 21(1): 43-71.

[19] Cheng M, Wu G, Jiang M, et al. Heterogeneous Defect Prediction via Exploiting Correlation Subspace[J].

[20] Bishnu P S, Bhattacherjee V. Software fault prediction using quad tree-based k-means clustering algorithm[J]. IEEE Transactions on knowledge and data engineering, 2012, 24(6): 1146-1150.

[21] Catal C, Sevim U, Diri B. Metrics-driven software quality prediction without prior fault data[M]//Electronic Engineering and Computing Technology. Springer Netherlands, 2010: 189-199.

[22] Zhang F, Zheng Q, Zou Y, et al. Cross-project defect prediction using a connectivity-based unsupervised classifier[C]//Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 309-320.

[23] Zhong S, Khoshgoftaar T M, Seliya N. Unsupervised Learning for Expert-Based Software Quality Estimation[C]//HASE. 2004: 149-155.

[24] Estabrooks A, Jo T, Japkowicz N. A multiple resampling method for learning from imbalanced data sets[J]. Computational intelligence, 2004, 20(1): 18-36.

[25] Japkowicz N, Myers C, Gluck M. A novelty detection approach to classification[C]//IJCAI. 1995, 1: 518-523.

[26] Zhou Z H, Liu X Y. Training cost-sensitive neural networks with methods addressing the class imbalance problem[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(1): 63-77.

[27] Rokach L. Ensemble-based classifiers[J]. Artificial Intelligence Review, 2010, 33(1-2): 1-39.

[28] Chawla N V, Lazarevic A, Hall L O, et al. SMOTEBoost: Improving prediction of the minority class in boosting[C]//European Conference on Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 2003: 107-119.

[29] Wang S, Yao X. Negative Correlation Learning for Class Imbalance Problems School of Computer Science, University of Birmingham[J]. 2012.

[30] Lewis D D. Naive (Bayes) at forty: The independence assumption in information retrieval[C]//European conference on machine learning. Springer Berlin Heidelberg, 1998: 4-15.

[31] Hall T, Beecham S, Bowes D, et al. A systematic literature review on fault prediction performance in software engineering[J]. IEEE Transactions on Software Engineering, 2012, 38(6): 1276-1304.

[32] Boetticher G, Menzies T, Ostrand T. PROMISE Repository of empirical software engineering data[J]. West Virginia University, Department of Computer Science, 2007.<http://promisedata.org/repository>.

[33] Lewis D D. Naive (Bayes) at forty: The independence assumption in information retrieval[C]//European conference on machine learning. Springer Berlin Heidelberg, 1998: 4-15.

[34] Breiman L. Random forests[J]. Machine learning, 2001, 45(1): 5-32.

[35] Hosmer Jr D W, Lemeshow S, Sturdivant R X. Introduction to the Logistic Regression Model[J]. Applied Logistic Regression, 2000.

[36] Liu Z, Wei C, Ma Y, et al. UCOR: an unequally clustering-based hierarchical opportunistic routing protocol for WSNs, International Conference on Wireless Algorithms, Systems, and Applications. Springer Berlin Heidelberg, 2013: 175-185.

[37] Liu Z, Wei C, Ma Y, et al. UCOR: an unequally clustering-based hierarchical opportunistic routing protocol for WSNs, International Conference on Wireless Algorithms, Systems, and Applications. Springer Berlin Heidelberg, 2013: 175-185.