Using Class Imbalance Learning for Cross-Company Defect Prediction

Xiao Yu[1], Mingsong Zhou[2], Xu Chen[1*], Lijun Deng[3], Lu Wang[4]
[1]State Key Lab. of Software Engineering, Computer School, Wuhan University, Wuhan, China
[2]Department of Information Science and Technology, Beijing Normal University, Beijing, China
[3]College of Business, City University of Hong Kong, Kowloon Tong, China
[4]School of Computer Science and Information Engineering, HuBei University, Wuhan, China
*Corresponding author email: xuchen@whu.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, the performance of such CCDP models is susceptible to the high imbalanced nature between the defect-prone and non-defect classes of CC data. Class imbalance learning is applied to alleviate this issue. Because many class imbalance learning methods have been proposed, there is an imperative need to analyze and compare the performance of these methods for CCDP. Although prior empirical studies have proven AdaBoost.NC algorithm achieves the best performance for defect prediction. This observation leads us to conduct a careful empirical study the issues of if and how class imbalance learning methods can benefit cross-company defect prediction. We investigate different types of class imbalance learning methods, including under-sampling technique, over-sampling technique and over sampling followed by under-sampling technique on the cross-company defect prediction performance over 15 publicly available datasets. Experimental results show that under-sampling technique achieves the best overall performance in terms of the g-measure among those methods we studied.**

Keywords—software defect prediction;cross-company defect prediction; class imbalance learning

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. With the advent of big data era and the development of machine learning techniques [1], many machine learning algorithms are applied to solve the practical problems in life [2-4]. Similarly, many efficient software defect prediction approaches [5-11] using machine learning techniques have been proposed, but they are usually confined to within company defect prediction (WCDP). WCDP works well if sufficient data is available to train a defect prediction model. However, it is difficult for a new company to perform WCDP 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 [12].

Due to the increased prevalence of machine learning and transfer learning techniques, a number of CCDP models have been proposed during the past decade [13-21]. Nevertheless, a challenge that threatens the performance of such CCDP models is the high imbalanced nature between the defect-prone and non-defect classes of CC data, i.e., defect datasets with much more non-defect instances (majority) than defect-prone instances (minority). Existing studies [22-23] have shown that 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.

Class imbalance learning is applied to alleviate this issue. Because many class imbalance learning methods have been proposed, there is an imperative need to analyze and compare the performance of these methods for CCDP. Although prior empirical studies have proven AdaBoost.NC algorithm achieves the best performance for defect prediction [23]. This observation leads us to conduct a careful empirical study the issues of if and how class imbalance learning methods can benefit cross-company defect prediction.

Therefore, several data sampling works should be done before building the CCDP model. For example, Lin et al. [19] 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 [24] to re-sample the CC data before building the CCDP model.

As the first effort of an in-depth study of class imbalance learning methods in CCDP, this paper explores their potential by focusing on research questions

RQ1: How effective are class imbalance learning methods?

RQ2: Do different class imbalance learning methods have significantly distinct effectiveness on CCDP?

For the two questions, we conduct a large scale empirical study on six class imbalance learning methods and compare them with CCDP models without involving any class imbalance learning method. The six class imbalance learning methods are Random under-sampling (RUS) [25], Near Miss (NM) [26], Synthetic Minority Oversampling Technique (SMOTE) [24]

DOI reference number: 10.18293/SEKE2017-035

and ADASYN [27], SMOTE Tomek links (STOME) [28] and SMOTE ENN (SENN) [29], covering three major types of solutions to learning from imbalanced data, i.e., under sampling technique, over sampling technique and over sampling followed by under-sampling technique. In the experiments, we choose the Naïve Bayes (NB), random forest (RF) and logistic regression (LR) as the CCDP models and pd, pf and g-measure as the evaluation measures. Experimental results over 15 publicly available datasets show that under-sampling technique achieves the best overall performance in terms of the g-measure among those methods we studied.

The remainder of this paper is organized as follows. Section 2 presents the background knowledge about cross-company defect prediction. Section 3 describes the class imbalance learning methods studied in this work. Section 4 presents the data sets, performance measures and the experimental results. Section 5 describes the treats to validity. Finally, Section 6 addresses the conclusion and points out the future work.

II. Related Work

In this section, we briefly review the existing cross-company and cross-company defect prediction approaches.

Briand et al. [12] used logistic regression and MARS models to learn a defect predictor, which is also the earliest work on CCDP. Zimmermann et al. [13] studied CCDP models on 12 real-world applications datasets. Their results indicate that CCDP is still a serious challenge. Turhan et al. [14] investigated the applicability of CC data for building localized defect predictors using 10 projects collected from two different companies including NASA and SOFTLAB. And they have proposed a nearest neighbor (NN) filter to select CC data. He et al. [15] investigates defect predictions in the cross-project context focusing on the selection of training data. Furthermore, they proposed an approach to automatically select suitable training data for projects without historical data so that the results of their experiments are comparable with WCDP, which indicated that some approach of CCDP can comparable to WCDP. They noted that learning predictors using the data from other projects can be a potential way to defect prediction without any historical data. In order to find data for quality prediction, Peters et al. [16] introduced the Peters filter to select training data via the structure of other projects. They compared the filter with two other approaches for quality prediction to assess the performance of the Peters filter, and found that 1) WCDP are weak for small data sets; 2) the Peters filter + CCDP builds better and more useful predictors. Zhang et al. [17] proposed sample-based methods for software defect prediction. For a large software system, they could select and test a small percentage of modules, and then built a defect prediction model to predict defect-proneness of the rest of the modules. They described three methods for selecting a sample and proposed a novel active semi-supervised learning method ACoForest to facilitate the active sampling. The results showed that the proposed methods are effective and have potential to be applied to industrial practice. Ma et al. [18] proposed a novel algorithm called Transfer Naive Bayes (TNB) to transfer cross-company data information into the weights of the training data and then build the predictor based on re-weighted CC data. The results indicated that TNB is more accurate in terms of AUC, within less runtime than the state of the art methods and can effectively achieve the CCDP task. The heterogeneous CCDP (HCCDP) task is that the source and target company data is heterogeneous. Jing et al. [11] provided an effective solution for HCCDP. They proposed a unified metric representation (UMR) for the data of source and target companies and introduced canonical correlation analysis (CCA), an effective transfer learning method, into CCDP to make the data distributions of source and target companies similar. Results showed that their approach significantly outperforms state-of-the-art CCDP methods for HCCDP with partially different metrics and for HCCDP with totally different metrics, their approach is also effective.

Turhan et al. [20] introduced a mixed model of within and cross data for CCDP to investigate the merits of using mixed project data for binary defect prediction. Results showed that when there is limited project history, mixed model for CCDP can achieve good performance which can be comparable to WCDP. It provided a new idea to CCDP that the use of a small amount of labeled WC data would be very valuable to improve the performance of CCDP. Lin et al. [19] introduced a novel approach named Double Transfer Boosting (DTB) to narrow the gap of different distributions between CC data and WC data and to improve the performance of CCDP by reducing negative samples in CC data.

III. Methodology

In this section, we only provide a brief description of the 6 class imbalance learning methods studied in this work due to the space limit. These methods cover three types including under sampling technique, over sampling technique and over sampling followed by under-sampling technique.

A. Under sampling techniques

Under-sampling is a technique to reduce the number of samples in the majority class, where the size of the majority class sample is reduced from the original datasets to balance the class distribution. In this study, we employ two representative under-sampling methods, i.e, Random under-sampling (RUS) and Near Miss (NM).

Random under-sampling (RUS) is a simple method to select a subset of majority class samples randomly and then combine them with minority class sample as a training set. The procedure of random under-sampling is as follows:

1. Calculate the ratio of the minority class to the majority class, and get the sampling frequency.
2. Sample the majority class by the sampling frequency.
3. Select all samples in the minority.
4. Combine selected samples and attributes for training.

Near Miss selects negative examples that are close to some of the positive examples. In the method, we select negative examples whose average distances to three closest positive examples are the smallest. This method guarantee every positive example is surrounded by some negative examples. Finally, in selection of most distant negative examples, we

choose the negative examples whose average distances to the three closest positive examples are the farthest. We expect the Near Miss methods should perform better than the random and distant methods, and the random method should work better than the distant method. We also expect that Near Miss method should achieve high precision and low recall while the distant method should achieve high recall and low precision.

B. Over sampling techniques

Over-sampling is a technique in which the minority class is over-sampled by creating "synthetic" examples rather than by under-sampling with replacement. In this study, we employ two representative under-sampling methods, i.e, Synthetic Minority Oversampling Technique (SMOTE) and ADASYN.

The procedure of SMOTE is as follows:

1. For each instance in the minority class, calculate the Euclidean distance between it and other samples in the minority 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 samples from k nearest neighbor randomly.

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 samples for each instance in the minority class and add new samples into it.

ADASYN is based on the idea of adaptively generating minority data samples according to their distributions: more synthetic data is generated for minority class samples that are harder to learn compared to those minority samples that are easier to learn. The procedure of ADASYN is as follows:

1. Calculate the degree of class imbalance.
2. If degree is less than *d* then (*d* is a preset threshold for the maximum tolerated degree of class imbalance ratio).
3. Calculate the number of synthetic data examples that need to be generated for the minority class.
4. For each example, find *K* nearest neighbors based on the Euclidean distance in *n* dimensional space.
5. Normalize according to a density distribution.
6. Calculate the number of synthetic data examples that need to be generated for each minority example.
7. For each minority class data example, generate synthetic data examples according to the following steps:

   Do the loop:

   i. Randomly choose one minority data example from the *k* nearest neighbors for data.

   ii. Generate the synthetic data example.

C. Over sampling followed by under sampling

Over sampling followed by under-sampling is a technique in which the minority class is over-sampled by creating "synthetic" examples followed by under-sampling with replacement. In this study, we employ two representative under-sampling methods, i.e, SMOTE Tomek links (STOME) and SMOTE ENN (SENN).

Although over-sampling minority class examples can balance class distributions, some other problems usually present in data sets with skewed class distributions are not solved. Frequently, class clusters are not well defined since some majority class examples might be invading the minority class space. The opposite can also be true, since interpolating minority class examples can expand the minority class clusters, introducing artificial minority class examples too deeply in the majority class space. Inducing a classifier under such a situation can lead to overfitting. In order to create better-defined class clusters, we propose to apply Tomek links to the over-sampled training set as a data cleaning method. Thus, instead of removing only the majority class examples that form Tomek links, examples from both classes are removed. The application of this method can be illustrated as follows. First, the original data set is over-sampled with Smote, and then Tomek links are identified and removed, producing a balanced data set with well-defined class clusters. The STOME links method was first used to improve the classification of examples for the problem of annotation of proteins in Bioinformatics.

The motivation behind SENN is similar to STOME. SENN tends to remove more examples than the Tomek links does, so it is expected that it will provide a more in depth data cleaning. Differently from NCL which is an under-sampling method, ENN is used to remove examples from both classes. Thus, any example that is misclassified by its three nearest neighbors is removed from the training set.

IV. Experiments

In this section, we first introduce the experiment dataset and the performance measures. Then, in order to investigate the performance of class imbalance learning methods, we perform some empirical experiments to find answers to the research questions 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

Project	*Examples*	*%Defective*	*Description*
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*

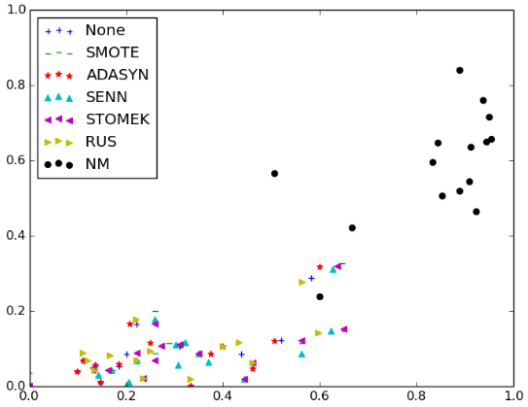
In every experiment, one dataset is selected as WC data and the rest are regarded as CC data to conduct the experiment. The CC data is considered as basic training data which will be adjusted in every experiment. All processing steps (data filter and data sampling) are done on CC data. Then processed 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), Random Forest (RF), and Logistic Regression (LR).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.

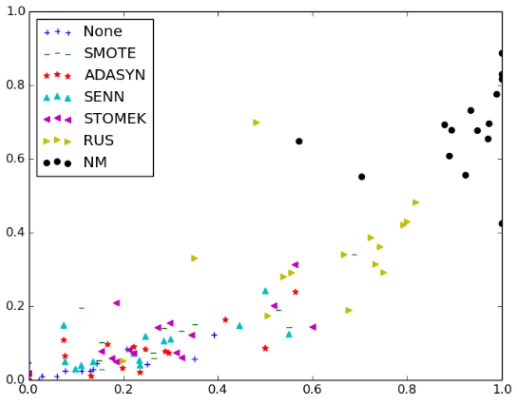
D. *Experimental results*

Fig. 1 presents the scatter plots of (PD, PF) points from the seven training methods on the ten SDP data sets. We can gain the following results from Fig. 1.

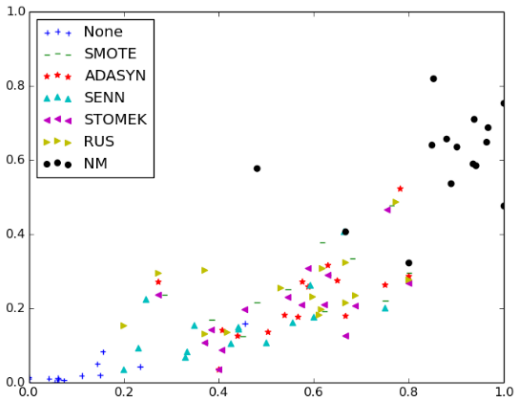
Although CCDP models without involving any class imbalance learning algorithms appears to be better at PF than other class imbalance learning models, it has lowest PD value, which proves class imbalance learning should be applied to CCDP models. In terms of the defect detection rate (PD), NM outperforms other class imbalance learning models, which shows its effectiveness in finding defects. However, NM has relatively high PF value. In terms of the false alarm rate (PF), although SENN is the best, it performs not well in PD, which makes it hardly useful in practice.



(a)NB



(b)RF



(c)LR

Fig. 1. Performances with Scatter plots of (PD, PF) points of the six class imbalacne learning methods and the CCDP model without class imbalance learning on the fifteen projects

Table 3-5 also show the g-measure values for each project on all the methods. From the tables, we can find that CCDP models without involving any class imbalance learning algorithms perform the worst in g-measure, which proves that the class imbalance problem is a big challenge for CCDP. In terms of NB model, NM achieves the best average g-measure value among the 15 projects. In terms of RF, RUS achieves the best average g-measure value among the 15 projects. In terms of LG, ADASYN achieves the best average g-measure value among the 15 projects. RUS and NM are under-sampling techniques, which reduce the number of CC instances in the non-defect class. We guess the reason under-sampling technique performs better is that it retains the defect-prone instances.

TABLE III. G-MEASURE PERFORMANCES WITH NAVIE BAYES

Project	*None*	*SMOTE*	*ADASYN*	*SENN*	*STOME*	*RUS*	*NM*
ant	0.32	0.50	0.38	0.50	0.50	**0.55**	0.43
arc	0.30	0.35	0.30	**0.53**	0.40	0.35	0.62
camel	0.62	0.61	0.62	0.46	0.61	0.61	**0.67**
elearn	0.00	0.00	0.00	0.33	0.00	0.00	**0.67**
jedit	0.46	0.47	0.55	0.47	0.46	**0.58**	0.51
log4j	0.25	0.37	0.25	0.34	0.37	0.37	**0.62**
lucene	0.17	0.23	0.17	0.24	0.23	0.21	**0.52**
poi	0.23	0.28	0.23	0.28	0.27	0.23	**0.49**
prop	0.23	0.43	0.23	0.45	0.41	0.28	**0.54**
redaktor	0.19	**0.40**	0.19	0.35	0.35	0.19	0.27
synapse	0.59	0.68	0.53	**0.69**	0.68	0.39	0.38
system	0.50	0.61	0.50	0.61	0.61	0.49	**0.62**
tomcat	0.65	**0.73**	0.64	0.72	0.73	0.70	0.60
xalan	0.64	**0.65**	0.63	**0.65**	0.65	0.63	0.50
xerces	0.34	0.39	0.33	0.39	0.39	0.34	**0.46**
AVG	0.37	0.45	0.37	0.47	0.44	0.39	**0.53**

TABLE IV. G-MEASURE PERFORMANCES WITH RANDOM FOREST

Project	*None*	*SMOTE*	*ADASYN*	*SENN*	*STOME*	*RUS*	*NM*
ant	0.51	0.67	0.64	**0.67**	0.70	0.66	0.29
arc	0.19	0.25	0.44	0.19	0.31	**0.62**	0.54
camel	0.14	0.26	0.14	0.14	0.26	**0.61**	0.60
elearn	0.00	0.00	0.00	0.00	0.00	0.33	**0.73**
jedit	0.25	0.47	0.44	0.44	0.49	**0.70**	0.36
log4j	0.11	0.41	0.37	0.37	0.48	**0.73**	0.51
lucene	0.04	0.26	0.23	0.17	0.29	**0.62**	0.41
poi	0.05	0.40	0.32	0.37	0.46	**0.66**	0.47
prop	0.23	0.34	0.28	0.23	0.34	**0.68**	0.45
redaktor	0.00	0.19	0.13	0.13	0.30	0.37	0.20
synapse	0.39	0.67	0.64	0.60	0.61	**0.72**	0.31
system	0.35	0.35	0.35	0.35	0.35	**0.66**	0.54
tomcat	0.33	0.49	0.55	0.43	0.44	**0.66**	0.48
xalan	0.54	0.63	**0.64**	0.58	0.62	0.63	0.46
xerces	0.22	0.42	0.38	0.38	0.41	**0.46**	0.43
AVG	0.20	0.39	0.37	0.34	0.40	**0.56**	0.45

TABLE V. G-MEASURE PERFORMANCES WITH LOGISTIC REGRESSION

Project	*None*	*SMOTE*	*ADASYN*	*SENN*	*STOME*	*RUS*	*NM*
ant	0.26	0.74	0.75	**0.77**	0.76	0.76	0.39
arc	0.13	0.52	0.55	0.48	0.52	0.51	**0.62**
camel	0.00	0.52	0.64	0.36	0.53	**0.69**	0.68
elearn	0.00	0.56	0.56	0.33	0.56	0.32	**0.73**
jedit	0.25	0.70	0.67	**0.69**	**0.69**	**0.69**	0.47
log4j	0.11	0.62	**0.65**	0.58	0.63	**0.65**	0.57
lucene	0.10	**0.59**	0.58	0.48	0.56	0.56	0.51
poi	0.08	**0.67**	0.63	0.57	0.66	**0.67**	0.49
prop	0.11	0.63	**0.64**	0.49	0.63	0.61	0.50
redaktor	0.00	**0.66**	0.65	0.65	**0.66**	0.48	0.29
synapse	0.11	**0.76**	0.74	0.64	0.73	0.72	0.44
system	0.19	**0.75**	0.73	0.66	**0.75**	0.72	0.61
tomcat	0.37	0.59	**0.68**	0.58	0.58	0.67	0.57
xalan	0.59	**0.62**	0.59	**0.62**	**0.62**	0.61	0.51
xerces	0.26	0.41	0.39	0.37	0.40	0.39	**0.45**
AVG	0.17	0.62	**0.63**	0.55	0.62	0.60	0.52

RQ1 Summary. According to the experiment results in Table 3-5, we conclude that these class imbalance learning algorithms can yield better prediction results than CCDP models without involving any class imbalance learning algorithms.

RQ2 Summary. According to the experiment results in Table 3-5, the above observations show that the effectiveness of these class imbalance learning methods exhibits significant differences on the performance of CCDP. Among the three types of imbalance learning methods, under-sampling technique is the winner according to g-measure.

V. THREATS TO VALIDITY

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

A. 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 our method on new datasets.

B. Internal validity

In our study, we repeat 30 times to avoid sample bias, and calculate average results to verify the performance of all test 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.

C. Construct validity

In our experiments, we mainly use pd, pf, g-measure to measure the effectiveness of the proposed method. Nevertheless, other evaluation measures such as AUC measure can also be considered.

VI. Conclusion and future work

The cross-company defect prediction is an interest problem in the field of software engineer. The class imbalance problem of defect datasets usually makes it difficult to build a CCDP model with high performance. In this paper, we address the issue how class imbalance learning methods can contribute to CCDP. We conduct a larger scale empirical study to investigate the impact of 6 class imbalance learning methods on the CCDP performance. We conduct experiments on the 15 datasets to evaluate the performance of the methods. The experimental results indicate that under-sampling technique is the winner according to g-measure among the three types of imbalance learning methods.

In the future, we would like to validate the generalization of our conclusion on more company data. In addition, we plan to apply our method to more real-life systems [30-31] to predict the defective module.

Acknowledgment

This work is partly supported by the grants of National Natural Science Foundation of China (No.61572374, No.U163620068, No.U1135005) and the Academic Team Building Plan from Wuhan University and National Science Foundation (NSF) (No. DGE-1522883).

References

[1] Xu Z, Liu Y, Mei L, et al. Semantic based representing and organizing surveillance big data using video structural description technology[J]. Journal of Systems and Software, 2015, 102: 217-225.

[2] Xu Z, Zhang S, Choo K K, et al. Hierarchy-cutting model based association semantic for analyzing domain topic on the web[J]. IEEE Transactions on Industrial Informatics, 2017.

[3] Xu Z, Mei L, Lu Z, et al. Multi-modal Description of Public Security Events using Surveillance and Social Data[J]. IEEE Transactions on Big Data, 2017.

[4] Xu Z, Liu Y, Mei L, et al. The mobile media based emergency management of web events influence in cyber-physical space[J]. Wireless Personal Communications, 2016: 1-14.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] Briand L C, Melo W L, Wust J. Assessing the applicability of fault-proneness models across object-oriented software projects[J]. IEEE transactions on Software Engineering, 2002, 28(7): 706-720.

[13] Zimmermann T, Nagappan N, Gall H, et al. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process[C]//Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009: 91-100.

[14] 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.

[15] He Z, Shu F, Yang Y, et al. An investigation on the feasibility of cross-project defect prediction[J]. Automated Software Engineering, 2012, 19(2): 167-199.

[16] 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.

[17] Zhang F, Mockus A, Keivanloo I, et al. Towards building a universal defect prediction model[C]//Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014: 182-191.

[18] 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.

[19] 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.

[20] Turhan B, Mısırlı A T, Bener A. Empirical evaluation of the effects of mixed project data on learning defect predictors[J]. Information and Software Technology, 2013, 55(6): 1101-1118.

[21] Xiao Yu,Jin Liu, Mandi Fu, et al. A Multi-Source TrAdaBoost Approach for Cross-Company Defect Prediction[C]// The 28th International Conference on Software Engineering & Knowledge Engineering. San Francisco Bay, California, USA,2016: 237-242.

[22] Ozturk M M, Zengin A. HSDD: a hybrid sampling strategy for class imbalance in defect prediction data sets[C]//Future Generation Communication Technologies (FGCT), 2016 Fifth International Conference on. IEEE, 2016: 60-69.

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

[24] 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.

[25] 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.

[26] Mani I, Zhang I. kNN approach to unbalanced data distributions: a case study involving information extraction[C]//Proceedings of workshop on learning from imbalanced datasets. 2003.

[27] He H, Bai Y, Garcia E A, et al. ADASYN: Adaptive synthetic sampling approach for imbalanced learning[C]//Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. IEEE, 2008: 1322-1328.

[28] Batista G E, Bazzan A L C, Monard M C. Balancing Training Data for Automated Annotation of Keywords: a Case Study[C]//WOB. 2003: 10-18.

[29] Batista G E, Prati R C, Monard M C. A study of the behavior of several methods for balancing machine learning training data[J]. ACM Sigkdd Explorations Newsletter, 2004, 6(1): 20-29.

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

[31] Liu Z, Niu X, Lin X, et al. A Task-Centric Cooperative Sensing Scheme for Mobile Crowdsourcing Systems[J]. Sensors, 2016, 16(5): 746.