ICIC Express Letters
Volume **11**, Number **10**, October **2017**

 ISSN 1881-803X
pp. **1507–1513**

COSDF: A NOVEL METHOD FOR SOFTWARE DEFECT DETECTION BASED ON CO-TRAINING AND SMOTE WITH DENSITY BASED NOISE FILTERING STRATEGY

Feng Yue[1] and Gang Wang[2]

[1]School of Computer and Information
[2]School of Management
Hefei University of Technology
No. 193, Tunxi Road, Hefei 230009, P. R. China
yuefeng@hfut.edu.cn; wgedison@gmail.com

Received May 2017; accepted July 2017

Abstract. *In recent years many machine learning and data mining methods have been proposed for software defect detection. However, the datasets of software defect detection are often imbalanced and only a small portion of instances are labeled in reality. In this paper, considering these practical issues simultaneously, a novel software defect detection method, COSDF, was proposed based on co-training and SMOTE. At the same time, in order to avoid introducing noise from the synthetic instances in SMOTE or new labeled instances in co-training, density based noise filtering strategy is used in the research. Experimental results on six public real-world datasets show that among the compared methods, COSDF gets the best result and COSDF is a potential solution for software defect detection.*

Keywords: Software defect detection, COSDF, Co-training, SMOTE, Density, Noise filtering

1. **Introduction.** Software systems have provided many benefits for the individual and for the community. On the other hand, the failures of software systems can cause a huge economic damage. According to the chairman of Standish Group, Jim Jonson, software defects cost businesses $78 billion every year [1,2]. Therefore, software defect detection has raised more and more interests from both academic and industry fields in recent years. Software defect detection has an important role in maintaining the quality of software systems. It is generally believed that repairing failures after the software development is one hundred times as expensive as repairing them before the software deployment [1,3].

In the existing software engineering literature, many machine learning and data mining methods, such as linear or logistic regression, Decision Tree (DT), K-Nearest-Neighbor (KNN), Artificial Neural Network (ANN), and Support Vector Machine (SVM), have been widely applied to the software defect detection [3-5]. For example, Czibula et al. proposed a novel ANN for detecting software faults by providing a two-dimensional representation of the faulty and non-faulty entities from a software system [6]. In reality, however, the datasets of software defect detection are often imbalanced and only a small portion of instances are labeled, which makes above mentioned methods getting the unsatisfied detection performance [7-9]. For the first problem, i.e., the imbalanced data problem, some researchers have noticed that the imbalanced distribution between defective and non-defective instances could greatly degrade the performance of software defect detection [7,9]. For example, Wang and Yao applied random sampling strategies, i.e., under sample non-defective instances and over sample defective instances, and found that balancing the skewed distribution could benefit to the detection performance [7]. For the second problem, i.e., the unlabeled data problem, some studies have found that the detection models learned from a small labeled training set may not perform well, and abundant unlabeled

instances could improve the detection accuracy significantly [8,9]. For instance, Seliya and Khoshgoftaar employed an EM-based semi-supervised learning method to detect software defect and found the generalization performance of the software defect detection improved greatly [8].

Although many studies have considered imbalanced data and unlabeled data problems independently, few researches have proposed methods specifically designed to solve these problems simultaneously [9]. In order to achieve more effective method for software defect detection, these two important problems should be considered simultaneously. Therefore, a new method, COSDF, is proposed for software defect detection based on co-training and SMOTE (Synthetic Minority Over-sampling TEchnique) with density based noise filtering strategy in this research. In the prior research, Jiang et al. proposed ROCUS method, which incorporates disagreement based semi-supervised learning with under sampling strategy, to detect software defect [9]. However, as the under sampling strategy may discard some useful information in the instances, co-training methods incorporating with over sampling strategy are employed in this research. At the same time, in order to avoid introducing noise from the synthetic instances in SMOTE or new labeled instances in co-training, density based noise filtering strategy is used. For the testing and illustration purpose, six public software defect detection datasets were selected to verify the effectiveness of the proposed method. Empirical results reveal that COSDF is a potential solution for software defect detection. Moreover, among the compared methods, COSDF gets the best result. All these results illustrate that COSDF could be used to software defect detection.

The remainder of the paper is organized as follows. In Section 2, a new method, i.e., COSDF, is proposed for software defect detection. Next, Section 3 presents the experimental design, while Section 4 is responsible for analyzing the experimental results. Finally, Section 5 draws conclusions.

2. COSDF for Software Defect Detection.

2.1. Problem formulation.

In this research, software defect detection problem is formulated as a semi-supervised binary classification problem, in which software module is classified as defective or non-defective using a set of software metrics. Let $L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ denote the set of labeled instances and let $U = \{x_{m+1}, x_{m+2}, \ldots, x_N\}$ denote the set of unlabeled instances. x_i is a d-dimensional feature vector. $y_i \in \{-1, +1\}$ is the class label. "+1" is denoted as the minority class, e.g., defective in the software defect detection. "−1" is denoted as the majority class, e.g., non-defective in the software defect detection. Both L and U are independently drawn from the same unknown distribution D, whose marginal distributions satisfy $P_D(y_i = +1) \ll P_D(y_i = -1)$.

As SMOTE method synthesizes minority instances and co-training picks high reliable instances into the training datasets, the noise can be potentially introduced, which may damage the performance of software defect detection. Therefore, density based noise filtering strategy is employed in the research, and some concepts and terms to explain the density based noise filtering method can be defined as follows [10].

Definition 2.1. *(Neighborhood). It is determined by a distance function for two points x_p and x_q, denoted by $dist(x_p, x_q)$.*

Definition 2.2. *(Eps-neighborhood). The Eps-neighborhood of a point x_p, denoted by $N_{Eps}(x_p)$, is defined by $N_{Eps}(x_p) = \{x_q \in X | dist(x_p, x_q) < Eps\}$, $X = \{x_1, x_2, \ldots, x_m\}$.*

Definition 2.3. *(Core point). A core point refers to the point whose neighborhood of a given radius (Eps) has to contain at least a minimum number (MinPts) of the other points.*

Definition 2.4. *(Directly density reachable). A point x_p is directly density reachable from a point x_q if x_p is within the Eps-neighborhood of x_q, and x_q is a core point.*

Definition 2.5. *(Density reachable). A point x_p is density reachable from the point q with respect to Eps and MinPts if there is a chain of points $x_{p_1}, \ldots, x_{p_n}$, $x_{p_1} = x_q$ and $x_{p_n} = x_q$ such that $x_{p_{i+1}}$ is directly density reachable from x_{p_i} with respect to Eps and MinPts, for $1 \ll i \ll n$, $x_{p_i} \in X$.*

Definition 2.6. *(Density connected). A point x_p is density connected to point x_q with respect to Eps and MinPts if there is a point $x_o \in X$ such that both x_p and x_q are density reachable from x_o with respect to Eps and MinPts.*

Definition 2.7. *(Border point). A point x_p is a border point if it is not a core point but density reachable from another core point.*

Based on above concepts, we can use DBSCAN (Density-Based Spatial Clustering of Applications with Noise) method to construct density based clusters.

Definition 2.8. *(Density based cluster). A cluster C is a non-empty subset of X satisfying the following requirements:*

(1) $\forall x_p$, x_q: if $x_q \in C$ and x_p is density reachable from x_q with respect to Eps and MinPts, then $P \in C$.

(2) $\forall x_p$, $x_q \in C$: x_p is density connected to x_q with respect to Eps and MinPts.

Definition 2.9. *(Noise). Let $C_1, \ldots, C_k$ be the clusters of non-empty subset of X. Then the noise is the set of points in X not belonging to any C_i, where $i = 1, \ldots, k$, noise $= \{p \in X | \forall i : p \notin C_i\}$.*

2.2. **A novel software defect detection method: COSDF.** For the imbalanced data problem, a number of methods were proposed from the perspective of data or algorithm [11]. Compared with algorithmic level methods, data level methods are algorithm independent and often used in practice. At the data level, sampling is a popular strategy to handle the imbalanced data problem since it straightforwardly re-balances the data set at the data processing stage. The simplest sampling methods are random over sampling and random under sampling [11]. As random under sampling method takes away some majority instances and could discard some useful information, over sampling strategy is employed in this research. Moreover, random over sampling method augments the minority instances and could make the decision regions more specific and cause over-fitting. SMOTE, one of the popular advanced over sampling method, was proposed [12]. However, as SMOTE randomly adds the synthetic minority instances into the training dataset, noises can probably be introduced at the same time, which will injure the detection performance. In order to reduce this kind of influence, SMOTE with the density based noise filtering strategy is proposed. Just like SMOTE, it also searches the K nearest neighbors for each minority instance at the very beginning. Unlike SMOTE, it uses DBSCAN method to judge whether the neighbor belongs to the noise. If the neighbor is the noise, this neighbor can be discarded directly. Then, it also employs DBSCAN method to judge whether the minority instance and neighbor belong to the same density based cluster. If the minority instance and neighbor do not belong to the same cluster, in order to avoid introducing new noise, it uses:

$$x_{new} = \dot{x} + rand(0, 1) \times Eps \tag{1}$$

to generate the new instance. $\dot{x}$ denotes the minority instance or the neighbor, which can be randomly selected in the method. If the minority instance and neighbor belong to the same cluster, it uses:

$$x_{new} = x + rand(0, 1) \times (\tilde{x} - x) \tag{2}$$

to generate the new instance. x denotes the minority instance and $\tilde{x}$ denotes the neighbor. Then the new instance x_{new} is judged whether it belongs to the noise. If it is the noise, this news instance could also be discarded directly. Compared with the traditional SMOTE method, SMOTE with the density based noise filtering strategy can generate more accurate minority instances by using density based noise filtering strategy.

For the unlabeled data problem, many approaches have also been proposed in the literature, such as semi-supervised learning, and active learning. Among these approaches, disagreement based semi-supervised learning approach explores the unlabeled data automatically, where no human intervention is assumed. Therefore, co-training, one of the popular disagreement based semi-supervised learning approaches, is used to software defect detection in this research. Co-training method starts with a few labeled instances to initialize a weak classifier and then use only unlabeled data to improve the learner's performance. It is based on the idea that feature sets are redundant sometimes and could be split into two sets which can be used to train two classifiers. Subsequently, we can train two classifiers which could be used to go through unlabeled instances, label them, and add the most confident instances to the labeled dataset. Just like SMOTE, co-training method could also potentially introduce noise into the training data, especially after the more minority instances are generated. Learning on the noisy dataset could humble the performance of classifiers. Therefore, density based noise filtering strategy is also employed into the standard co-training method. For every training iteration, the potential instance, which will be added into the training dataset, must be judged whether it belongs to the noise. If the potential instance belongs to the noise, it will be discarded directly.

Based on above analysis, COSDF is proposed for software defect detection to solve imbalanced data and unlabeled data problems simultaneously. In COSDF, each classifier is generated using the original labeled training dataset with improved SMOTE firstly. Then, it iterates the following procedures K times. Firstly, each classifier labels P positive and N negative unlabeled instances for its peer learner with density based noise filtering strategy. Then, the classifiers will be refined using newly labeled instances provided by its peer. The whole process will repeat until classifiers are unchanged or pre-set number of learning rounds K has been executed. The pseudo code of COSDF is shown in Figure 1.

3. **Experimental Design.** We evaluated the effectiveness of COSDF on six software defect detection benchmark datasets: KC1, KC2, KC3, PC1, PC3, and PC5. Each dataset is comprised of the number of defects and some static code metrics, including LOC (Lines of Codes) counts, McCabe complexity measures, Halstead attributes, etc., software components which contain one or more defects are labeled as defective, while the others are labeled as non-defective. The statistic summary of the six defect datasets is shown in [4].

It is now well-known that average accuracy is not an appropriate evaluation criterion when there is class imbalance. Thus, we use AUC (Area Under the ROC Curve) [13] as performance evaluation measure in this research. It has been proven scientifically that AUC is a reliable measurement for performance measuring in the imbalanced data problem [13,14]. AUC is produced based on ROC curve. ROC curve is a two-dimensional graph to select possibly optimal models based on the True Positive Rate (TPR) and False Positive Rate (FPR). In the experiment, SVM was chosen as base learner for the COSDF, imbalanced classification methods, i.e., Under Sampling method (US), Over Sampling method (OS), SMOTE, Bagging, Boosting, and co-training related methods, i.e., the standard Co-Training (CT), Co-Training with Under Sampling strategy (CT-US), Co-Training with Over Sampling strategy (CT-OS), Co-Training with SMOTE (CT-SMOTE), Co-Training with Bagging (CT-Bagging), Co-Training with Boosting (CT-Boosting). The standard co-training assumes there are two redundant views, which are difficultly obtained for the software defect detection problem. Moreover, following [15], we also randomly partitioned

Input: Labeled instance set L;
Unlabeled instance set U;
Eps-neighborhood Eps;
Mminimum number of points MinPts;
Learning round K;
The number of positive instance p;
The number of negative instance n;
Base classifiers f_1, f_2.

Process:

1. Use DBSCAN (L, Eps, MinPts) to get $C_1^p, \ldots, C_k^p$ and $C_1^n, \ldots, C_k^n$;
2. Use improved SMOTE to generate balanced data set L_1, L_2 using Equations (1) and (2);
3. Loop for K iterations:
4. Use L_1 to train a classifier f_1 that considers only the x_1 portion of x;
5. Use L_2 to train a classifier f_2 that considers only the x_2 portion of x;
6. Do
7. Allow f_1 to label most confident positive instance from U;
8. If ($x_p \notin Noise$), add x_p into L_2;
9. While (f_1 labels p positive instance)
10. Do
11. Allow f_1 to label most confident negative instance from U;
12. If ($x_n \notin Noise$), add x_n into L_2;
13. While (f_1 labels n negative instance)
14. Do
15. Allow f_2 to label most confident positive instance from U;
16. If ($x_p \notin Noise$), add x_p into L_1;
17. While (f_2 labels p positive instance)
18. Do
19. Allow f_2 to label most confident negative instance from U;
20. If ($x_n \notin Noise$), add x_n into L_1;
21. While (f_2 labels n negative instance)

Output: $F(x) = f_1(x_1) * f_2(x_2)$

FIGURE 1. The pseudo code of COSDF

the original features into two subsets with similar size, and then each subset is regarded as a view to train co-training based on methods.

To minimize the influence of the variability of the training set, ten times 10-fold cross validation is performed on the dataset. During the process of experiment, each dataset is partitioned into ten subsets with similar sizes. Then, the union of nine subsets is used as the training set while the remaining subset is used as the test set, which is repeated for ten times such that every subset has been used as the test set once. For the union dataset of nine subsets, it is partitioned into a labeled training dataset L, and an unlabeled training dataset U under different label rates including 20%, 40%, 60%, and 80%.

4. **Results and Discussion.** Table 1 summarizes the experiment results of different methods when label rate is 20%. The highest AUCs under different datasets are boldfaced. Generally speaking, the results obtained from Table 1 show that the performance of proposed COSDF method is the better than the performance of the other methods.

As shown in Table 1, for the imbalanced data classification method, compared with SVM, SMOTE gets the best results at KC1, KC2, PC1, and PC3. Bagging gets the best results at KC3 and PC5. For the Co-training method, compared with SVM, it gets the

TABLE 1. Experimental resutls at label rate = 20%

Method	KC1	KC2	KC3	PC1	PC3	PC5
SVM	0.6921	0.8004	0.6406	0.7041	0.6952	0.8836
US	0.7929	0.8372	0.5849	0.7412	0.7849	0.8483
OS	0.7882	0.8356	0.6306	0.7468	0.7880	0.8666
SMOTE	0.7911	0.8378	0.6099	0.7475	0.7988	0.8710
Bagging	0.7188	0.8178	0.6784	0.7452	0.7504	0.8838
Boosting	0.7358	0.7626	0.6379	0.7209	0.7383	0.8518
CT	0.7975	0.8518	0.7012	0.7560	0.7904	0.8874
CT-US	0.8028	0.8550	0.7057	0.7844	0.8165	0.8555
CT-OS	0.8038	0.8510	0.6692	0.7921	0.8106	0.8695
CT-SMOTE	0.8005	0.8490	0.6819	0.7775	0.8111	0.8662
CT-Bagging	0.8043	0.8530	0.7160	0.7762	0.8041	0.8866
CT-Boosting	0.7957	0.8523	0.7306	0.8093	0.8130	0.8896
ROCUS	0.7986	0.8539	0.7143	0.7898	0.8017	0.8882
COSDF	**0.8051**	**0.8564**	**0.7329**	**0.8145**	**0.8157**	**0.8943**

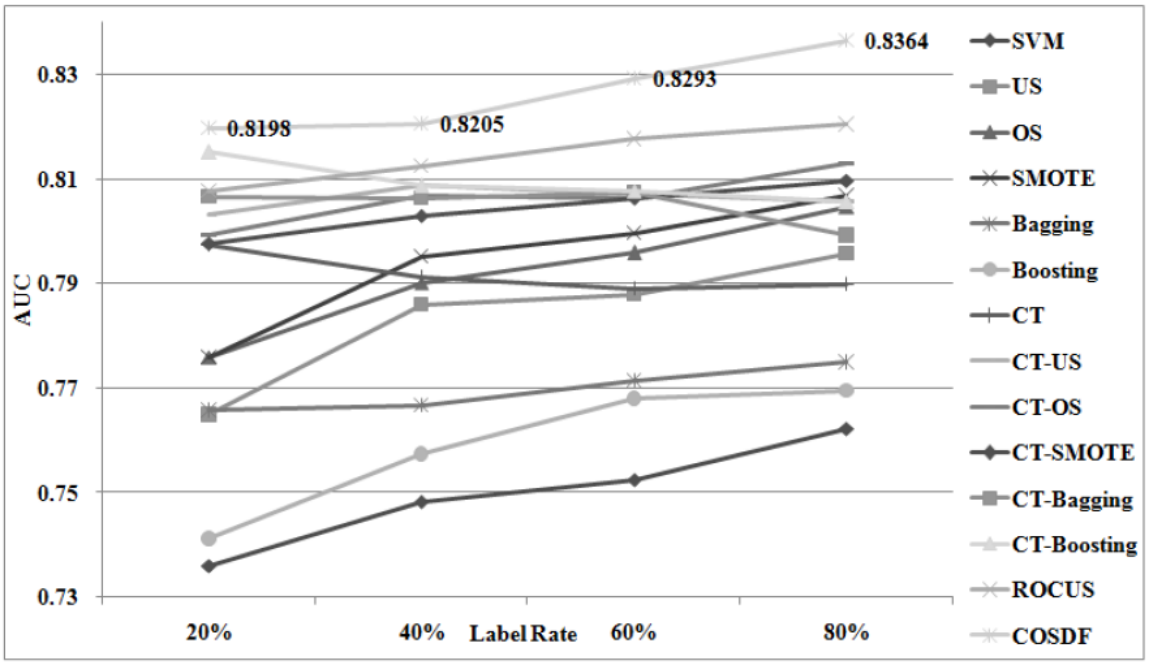

FIGURE 2. Performance comparisons under different label rates

better results at the six datasets. Moreover, COSDF gets the highest AUC, i.e., 80.51%, 85.64%, 73.29%, 81.45%, 81.57%, and 89.43%, at the six datasets.

Next, the average AUCs across six dataset are compared at the different label rates. The experimental result is shown in Figure 2. COSDF gets the highest average AUC: 81.98%, 82.05%, 82.93%, and 83.64%. It is interesting that with the increase of label rate, the average AUCs of compared methods are also increasing, except for CT, CT-US, CT-OS, CT-Bagging, and CT-Boosting. On the one hand, these results indicate that the labeled data are very important for software defect detection, although the co-training method can utilize the unlabeled data. On the other hand, the performances of co-training based methods almost are not improved, as the new added instances probably include noise and deteriorate the performance of base classifiers. However, these results verify that the density based noise filtering strategy is effective on the contrary.

5. **Conclusions.** In this study, a new method, COSDF, is proposed for software defect detection based on co-training and SMOTE with density based noise filtering strategy. Experimental results based on the six public software defect detection datasets show that COSDF gets the highest average AUC among the compared methods. Several future research directions also emerge. Firstly, large datasets for experiments and applications should be collected to further verify the conclusions of this study. Secondly, as this research only verifies the proposed method experimentally, more deep theoretical analyses for COSDF are needed in the future research.

Acknowledgments. This work is partially supported by the National Natural Science Foundation of China (71471054, 91646111), Anhui Provincial Natural Science Foundation (1508085QF129, 1608085MG150).

REFERENCES

[1] C. Catal, Software fault prediction: A literature review and current trends, *Expert Systems with Applications*, vol.38, pp.4626-4636, 2011.

[2] P. K. Singh, R. K. Panda and O. P. Sangwan, A critical analysis on software fault prediction techniques, *World Applied Sciences Journal*, vol.33, pp.371-379, 2015.

[3] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, *Applied Soft Computing*, vol.27, pp.504-518, 2015.

[4] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans. Software Engineering*, vol.34, pp.485-496, 2008.

[5] V. Garousi, M. Felderer and T. Hacaloglu, Software test maturity assessment and test process improvement: A multivocal literature review, *Information and Software Technology*, vol.85, pp.16-42, 2017.

[6] I.-G. Czibula, G. Czibula, Z. Marian and V.-S. Ionescu, A novel approach using fuzzy self-organizing maps for detecting software faults, *Studies in Informatics and Control*, vol.25, pp.207-216, 2016.

[7] S. Wang and X. Yao, Using class imbalance learning for software defect prediction, *IEEE Trans. Reliability*, vol.62, pp.434-443, 2013.

[8] N. Seliya and T. M. Khoshgoftaar, Software quality estimation with limited fault data: A semi-supervised learning perspective, *Software Quality Journal*, vol.15, pp.327-344, 2007.

[9] Y. Jiang, M. Li and Z.-H. Zhou, Software defect detection with ROCUS, *Journal of Computer Science and Technology*, vol.26, pp.328-342, 2011.

[10] D. Birant and A. Kut, ST-DBSCAN: An algorithm for clustering spatial-temporal data, *Data & Knowledge Engineering*, vol.60, pp.208-221, 2007.

[11] H. He and E. A. Garcia, Learning from imbalanced data, *IEEE Trans. Knowledge and Data Engineering*, vol.21, pp.1263-1284, 2009.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, *Journal of Artificial Intelligence Research*, vol.16, pp.321-357, 2002.

[13] T. Fawcett, ROC graphs: Notes and practical considerations for researchers, *Machine Learning*, vol.31, pp.1-38, 2004.

[14] A. P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, *Pattern Recognition*, vol.30, pp.1145-1159, 1997.

[15] Z.-H. Zhou and M. Li, Tri-training: Exploiting unlabeled data using three classifiers, *IEEE Trans. Knowledge and Data Engineering*, vol.17, pp.1529-1541, 2005.