Adaptive Weighted Broad Learning System for software defect prediction

Kankan Lan
Computer Science&Engineering
South China University of Technology
Guangzhou, China
201820132815@mail.scut.edu.cn

Kaixiang Yang
Computer Science&Engineering
South China University of Technology
Guangzhou, China
201710105782@mail.scut.edu.cn

Zhiwen Yu
Computer Science&Engineering
South China University of Technology
Guangzhou, China
zhwyu@scut.edu.cn

Guoqiang Han
Computer Science&Engineering
South China University of Technology
Guangzhou, China
csgqhan@scut.edu.cn

Jane You
Department of Computing
The Hong Kong Polytechnic University
Hong Kong, China
csyjia@comp.polyu.edu.hk

C. L. Philip Chen
Computer Science&Engineering
South China University of Technology
Guangzhou, China
philipchen@scut.edu.cn

***Abstract*—In order to improve the efficiency of software testing and save the costs, a series of machine learning methods are used to build software defect prediction models with the existing project data. However, the class imbalance problem has posed a leading challenge in software defect prediction (SDP). To alleviate the impact of class imbalance in SDP, we firstly propose a weighted broad learning system (WBLS), which assigns weight to each module and adopts a general weighting scheme to emphasize the defective modules. Further, we design a density based weight generation scheme and propose an adaptive weighted BLS (AWBLS). This mechanism considers both the inter-class and intra-class distance simultaneously to explore the prior distribution information of the original data. Extensive comparative experiments on NASA data sets verify that AWBLS tends to be an effective and efficient model in the area of software defect prediction.**

***Index Terms*—imbalance learning, broad learning system, software defect prediction**

I. Introduction

Software defect prediction (SDP) is an important way to improve software testing efficiency and ensure software reliability, which has become a research hotspot in the field of software engineering [1], [2]. Software defects arise from the coding process of software developers. In the development life cycle of a software project, the later the internal defect is detected, the higher the repairing cost is needed. SDP can predict the number and type of defects in software projects based on the historical software development data, and has attracted attention from academia to industry. The dominant method in the area of SDP is machine learning. In the process of SDP by machine learning, data in the software historical repositories including bug reports, source code log files and interaction between developers is explored, and models are built to predict new unseen software modules [3]–[5].

Software defect prediction in the real world usually face the imbalance problem. That is, the number of defective modules is very small, while non-defective modules occupy the vast majority of the software defect data set [6]. The conventional machine learning algorithms are proposed under the assumption that data follow a balanced distribution, it is easy to sacrifice the classification effect of defective modules for the overall accuracy. As a result, prediction models developed for SDP using traditional classification methods may be biased and inaccurate. Various approaches have been investigated to address the imbalance issue, including data-oriented methods and model-oriented methods [7]. By decreasing the majority class samples or increasing the minority class samples to get relatively balanced training set, data-oriented methods aim at modifying the distribution of original data. As the most typical type of the model-oriented methods, cost-sensitive learning provides different misclassification costs for different samples. In the case of imbalance problem, samples from the minority class will be easily misclassified. The cost matrices are explored to adjust the cost of misclassification and balance the data from another aspect [8], [9].

As an effective and efficient machine learning technique, broad learning system (BLS) [10] has been developed in the past few years, which has attracted attention from academia to industry. BLS is designed by the inspiration of random vector functional-link neural network (RVFLNN) [11], [12]. The hidden layer of BLS is composed of feature nodes and enhancement nodes in a broad manner. Feature nodes are firstly obtained from the input data with random initialized weights, and enhancement nodes are mapped again by a linear combination of feature nodes with random weights to expand the features from the width level. The output weights of hidden nodes are calculated by pseudo-inverse. BLS achieves comparable performances to existing DNNs in terms

This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1703600, in part by National Natural Science Foundation of China grant under number 61751202, 61751205, 61722205, 61572540, U1611461, U1813203, U1801262, and in part by the Key Research and Development Program of Guang Dong Province under Grant 2018B010107002. (Corresponding author: Kaixiang Yang.)

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

of accuracy with extremely fast training speed on MNIST and NORB [13]. Besides, it provides a reasonable mathematical proof of universal approximation property.

In response to the class imbalance problem in SDP, combined with the effective and efficient performance of broad learning system, we propose a weighted BLS (WBLS) that assigns a weight to each training sample. WBLS adopts a general weighting scheme, which augments the weight of samples from the minority class. To further explore the prior distribution information of original data, we design a density based weight generation scheme to guide the specific weight matrix generation and propose the adaptive weighted broad learning system (AWBLS). The corresponding weights are assigned automatically based on the inter-class and intra-class information simultaneously. The experimental results on real-world data sets against other imbalance classification approaches verify the effectiveness and superiority of our proposed method.

The contributions of this paper are summarized as follows:

1) We propose a weighted broad learning system (WBLS) with a general weighting scheme using information of imbalance class distribution to tackle imbalance problem in SDP.
2) We design a density based weight generation scheme to guide specific weight matrix generation and propose an adaptive weighted broad learning system (AWBLS), which considers both the inter-class and intra-class distances simultaneously in the density calculation.
3) We use non-parametric tests to compare multiple approaches over different data sets in SDP.

The remainder of this paper is organized as follows. Section II reviews related works on software defect prediction, broad learning system and imbalance learning. In Section III, we introduce the weighted broad learning system and an adaptive density based weight generation mechanism. Section IV presents the performance and analysis of experiments. Finally, we draw the conclusion in Section V.

II. Related work

This section describes three focal points of this paper. First, we introduce the newly proposed machine learning technique broad learning system (BLS) and its variants. Subsequently, we describe the research progress of imbalance learning. Finally, we review the related work on SDP associated with machine learning and imbalance learning.

Differing from deep learning, BLS is established in the form of a flat network. The input data are transformed first by random weights and biases to construct feature nodes, and then enhancement nodes are formed based on connection of feature nodes with random parameters and linear activation function. Finally, feature nodes and enhancement nodes are connected to the output layer, and output weights are calculated by a fast pseudo-inverse learning. Existing work has confirmed that such structure can accelerate the model training process [10]. BLS has gained widespread attention for its outstanding performance, and different variants have also been developed. For example, [14] shows some structural variations of BLS and details the corresponding mathematical proof. A fuzzy broad learning system (Fuzzy BLS) is designed with the fusion of fuzzy system and BLS [15]. A feature selection algorithm for orthogonal broad learning system based on mutual information is proposed in [16]. Reference [17] presents a novel regularized robust broad learning system to deal with uncertain data. With flexible structure, high efficiency and good generalization, the BLS gradually becomes a promising alternative approach.

Imbalance learning can be divided into: data-level methods, algorithm-level methods and hybrid approaches. Data-level methods modify the original distribution to get balanced data set and improve the classification performance. Among them, the simplest and non-heuristic techniques are random over-sampling (ROS) and random under-sampling (RUS). ROS duplicates the minority class samples randomly, which will cause the information redundancy and model over-fitting. RUS randomly deletes the samples of majority class, which will lead to the information loss and affect the classification performance. To enhance the rationality, universality and generalization of re-sampling, heuristic techniques have been proposed. The synthetic minority over-sampling technique (SMOTE) is the typical one. By randomly generating synthetic samples between the minority class instance and its neighbors [18], SMOTE alleviates the overfitting problem of ROS. Some variants of SMOTE have also been proposed to improve the generation of synthetic data from different aspects, including adaptive synthetic sampling (ADASYN) [19] and Safe-Level-SMOTE [20]. In ADASYN, the density information of minority class samples is used to assist the synthetic sample generation of SMOTE. Safe-Level-SMOTE applies different sample generation strategies based on different safe level ratios of the minority class samples. Another classic under-sampling technique is cluster-based [21], the main idea of which is to identify and preserve class space through the clustering methods to guide re-sampling process. For hybrid approaches, the typical way is combining multiple techniques at the data and algorithm level by adopting ensemble learning [22].

Algorithm-level approaches address imbalance problems by modifying or designing algorithms. Specifically, cost-sensitive learning methods consider different costs for misclassifying different samples. In the case of imbalance, the minority class samples can be easily misclassified and the misclassification costs of them worth further exploration. By constructing specific cost matrices, cost-sensitive learning methods minimize the overall misclassification cost instead of the overall error to address the imbalance issues from another perspective [23]. Empirical researches explore and compare the characteristics between re-sampling methods and cost-sensitive learning methods [24]. In [25], a cost-sensitive decision tree ensemble approach is investigated for effective imbalanced classification. Weighted ELM (WELM) [26] is designed to handle imbalanced data with the combination of ELM and cost-sensitive learning. Another classical cost-sensitive algorithm is AdaCost [27], which inherits the framework of adaboost and modifies

the weight update strategy to focus on minority samples.

Existing software defect methods can be divided into static defect prediction methods and dynamic defect prediction methods [1]. Among them, the static prediction method is based on defect-related measurement data to analyze and predict the defect tendency, defect density and number of defects of program modules. The dynamic defect prediction method is used to predict the distribution of system defects over time, so as to discover the distribution rule of the software defect with its life cycle. A variety of machine learning methods have been widely used in the field of SDP, such as support vector machines [28], neural networks [29], [30], Naive Bayes [31] and ensemble learning. In software defect prediction problems, the number of defective modules is very small. Traditional machine learning algorithms will easily sacrifice the classification effect of defective modules for overall accuracy, resulting in biased and inaccuracy performance. To address the class imbalance problem of SDP, different strategies have been investigated by researchers. In [32], a dynamic version of AdaBoost.NC is proposed to address the imbalance problem in SDP, which adjusts parameters automatically during the training process. In [33], three cost-sensitive boosting approaches are proposed to boost neural networks for SDP. The effect of different classifier types combined with different imbalance learning methods are explored in the application of SDP [6].

III. Methodology

In this section, we describe the Weighted Broad Learning System (WBLS) and the Adaptive Weighted Broad Learning System (AWBLS) in detail. AWBLS includes the weighted broad learning system and the adaptive weight generation scheme. Fig.1 shows an overview of AWBLS and Algorithm 1 presents the corresponding pseudo code.

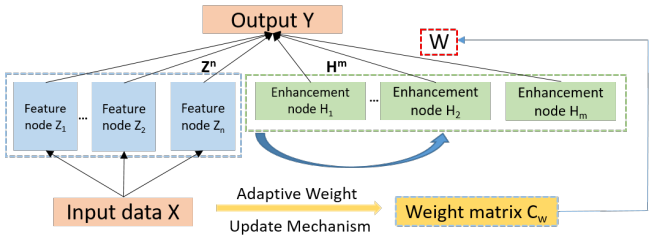

Fig. 1: An overview of adaptive weighted broad learning system.

A. Weighted Broad Learning System

Assume the input $T_r = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, where N is the total number of samples, and M is the dimensionality of features. The ground-truth labels $\boldsymbol{y}_i = [y_{i1}, y_{i2}, ..., y_{iK}]$ are one-hot vectors, where K is the number of categories.

With random generated parameters $\{\mathbf{W}_{e_1}, ..., \mathbf{W}_{e_n}\}$ and $\{\boldsymbol{\beta}_{e_1}, ..., \boldsymbol{\beta}_{e_n}\}$, the input data will be mapped into a new feature space by:

$$\mathbf{Z}_i = \phi_i(\mathbf{X}\mathbf{W}_{e_i} + \boldsymbol{\beta}_{e_i}), i = 1, 2, ..., n. \tag{1}$$

where ϕ_i denotes the transformation function. The mapping feature nodes $(\mathbf{Z}_1, \mathbf{Z}_2, ..., \mathbf{Z}_n)$ are concatenated to construct the feature layer $\mathbf{Z}^n$.

The enhancement layer is generated with $\mathbf{Z}^n$ by specific nonlinear transformation, which can be represented as:

$$\mathbf{H}_j = \zeta_j(\mathbf{Z^n}\mathbf{W}_{h_j} + \boldsymbol{\beta}_{h_j}). \tag{2}$$

where ζ_j denotes a nonlinear transformation. $\mathbf{W}_{h_j}$ and $\boldsymbol{\beta}_{h_j}$ are randomly generated and used to connect the feature layer and the enhancement layer.

The feature nodes and enhancement nodes are connected to the output layer, *i.e.*, there is one hidden layer in the weighted broad learning system (WBLS). The hidden layer can be denoted by:

$$\mathbf{A} = [\mathbf{Z}_1, \mathbf{Z}_2, ...\mathbf{Z}_n, \mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_m] \tag{3}$$

$$= [\mathbf{Z^n}\ \mathbf{H^m}]. \tag{4}$$

Algorithm 1 Adaptive Weighted Broad Learning System

Require:
Input: the training set T_r, the label set $\mathbf{Y}$;
transformation function ϕ and ζ, parameter C
Ensure:
1: Generate the weights $\{\mathbf{W}_{e_1}, ..., \mathbf{W}_{e_n}\}$ and the biases $\{\boldsymbol{\beta}_{e_1}, ..., \boldsymbol{\beta}_{e_n}\}$ randomly;
2: **For** i in $1, ..., n$:
3: Calculate $\mathbf{Z}_i = \phi_i(\mathbf{X}\mathbf{W}_{e_i} + \boldsymbol{\beta}_{e_i})$;
4: **end For**;
5: Concatenate the feature mapping nodes: $\mathbf{Z^n} = [\mathbf{Z}_1, \mathbf{Z}_2, ...\mathbf{Z}_n]$;
6: Generate m pairs of random weights $\{\mathbf{W}_{h_1}, ..., \mathbf{W}_{h_m}\}$ and biases $\{\boldsymbol{\beta}_{h_1}, ..., \boldsymbol{\beta}_{h_m}\}$;
7: **For** j in $1, ..., m$:
8: Calculate $\mathbf{H}_j = \zeta_i(\mathbf{X}\mathbf{W}_{h_j} + \boldsymbol{\beta}_{h_j})$;
9: **end For**;
10: Concatenate the enhancement nodes: $\mathbf{H^m} = [\mathbf{H}_1, \mathbf{H}_2, ...\mathbf{H}_m]$;
11: Concatenate the feature nodes and enhancement nodes: $\mathbf{A} \leftarrow [\mathbf{Z}\ \mathbf{H}]$;
12: Call the adaptive weight generation process in Algorithm 2 to generate the weight matrix $\mathbf{C_w}$;
13: Calculate the output weight $\mathbf{W}$ via equation (5)
Output: $\mathbf{W}$.

Assume that the weight of output layer is denoted as W, the optimization problem below can be considered to find the solution for $\mathbf{W}$:

$$\arg\min_{\mathbf{W}} : \mathbf{C_w} \left\|\mathbf{A}\mathbf{W} - \mathbf{Y}\right\|_2^2 + \lambda \left\|\mathbf{W}\right\|_2^2, \tag{5}$$

where $\mathbf{Y}$ is the label set, and λ is the trade-off parameter. The weight matrix $\mathbf{C_w}$ is a $N \times N$ diagonal matrix associated with each input sample, which is used to address imbalance problem. Samples from the majority class are usually assigned smaller weights than samples from the minority class.

The pseudo-inverse method can be used to solve the above problem. Consequently, we have:

$$\mathbf{W} = (\lambda \mathbf{I} + \mathbf{A}^{\mathbf{T}} \mathbf{C}_{\mathbf{w}} \mathbf{A})^{-1} \mathbf{A}^{\mathbf{T}} \mathbf{C}_{\mathbf{w}} \mathbf{Y}. \tag{6}$$

When the output weight $\mathbf{W}$ is obtained, the training process is finished without fine-tuning, which is an important factor of the high efficiency of WBLS. Algorithm 1 includes a detailed description of WBLS. With the predicted labels denoted as $\mathbf{Y}^*$, the result can be obtained by:

$$\mathbf{Y}^* = \mathbf{A}\mathbf{W}. \tag{7}$$

The weight matrix $\mathbf{C_w}$ considers misclassifying examples and provides different weights to different samples in classification. C_{ii} represents the specific weight of data x_i in $\mathbf{C_w}$. For the imbalance classification task, we can adopt a general weighting scheme for WBLS:

$$C_{ii} = 1/N_{ik}, \tag{8}$$

where N_{ik} represents the number of samples in class k which the i-th sample belongs to.

B. Adaptive Density Based Weight Generation Process

Noise and outliers often exist in the software defect data, which will obviously affect the performance of the prediction model. To adaptively adjust the weight matrix for different software defect data sets and reduce the impact of noise or outliers, we propose an adaptive density based weighted broad learning system (AWBLS), which improves WBLS by adopting an adaptive weight update mechanism based on the distribution information of original data. Samples around decision boundaries are always difficult to distinguish, which can provide more information for model construction. Besides, weights of noisy or outlier samples in the software defect data should be small. Based on these criteria, we design a density based weight calculation scheme to adaptively acquire the prior distribution information and generate more appropriate weight matrix. Algorithm 2 presents the adaptive weight generation process (AWGP), which pays more attention on samples around decision boundary and reduces negative impacts of noise or outlier samples. AWBLS improves the density assessment by evaluating both the inter-class and intra-class density simultaneously.

Given the input data x_i, the class which x_i belongs to is defined as T_p and the rest class is defined as T_n. Specifically, for x_i, we firstly calculate the distance $D_i^p = \{d_{i1}^p, d_{i2}^p, ..., d_{ik}^p\}$ as the k nearest neighbors of data x_i in T_p, and the distance $D_i^n = \{d_{i1}^n, d_{i2}^n, ..., d_{ih}^n\}$ as the h nearest neighbors f data x_i in T_n. The densities of x_i in T_p and T_n are defined as follows:

$$\delta_i^p = \frac{1}{\mathbb{E}\left[D_i^p\right]} = \frac{1}{\frac{1}{k}\sum_{j=1}^{k} d_{ij}^p}, \tag{9}$$

$$\delta_i^n = \frac{1}{\mathbb{E}\left[D_i^n\right]} = \frac{1}{\frac{1}{h}\sum_{j=1}^{h} d_{ij}^n}. \tag{10}$$

The mixed density of x_i is calculated by:

$$R_i = \omega \delta_i^p + (1-\omega)\, \delta_i^n, \tag{11}$$

where ω is a trade-off parameter. The mixed density of each sample can be normalized as follows:

$$\gamma_i = \frac{R_i}{\sum_{x_i \in T_r} R_i} \tag{12}$$

Considering the imbalance rate IR of the data set, the weight of data x_i will be calculated by:

$$C_{ii} = \gamma_i * (1/N_{ik}) \tag{13}$$

The weight matrix $\mathbf{C_w}$ is obtained by calculating each sample with formula (9) - (13). Compared to formulation (8), $\mathbf{C_w}$ allows paying more attention on difficult samples while reducing impacts of noisy and outlier samples in the classification.

Algorithm 2 Adaptive weight generation process

Require:
Input: the input data set $T_r = \{x_1, x_2, ..., x_N\}$
Ensure:
for i in $1, ...N$:
  Divide T_r into two components as $T = T_p \cup T_n$,
  Compute the local density of x_i by Eq.(9);
  Calculate the global density of x_i via Eq.(10)
  Get the density weight γ_i by Eqs.(11)-(12);
  Compute the regulatory factor $\eta_i = 1/N_{ik}$;
  Update the weight: $C_{ii} = \eta_i \gamma_i$;
end For;
Output: the weight matrix $\mathbf{C_w}$

IV. Experiments

In this section, we compare performances of AWBLS and other approaches on NASA software defect prediction data set selected from the PROMISE database [34]. The NASA data sets are public data sets released by the National Aeronautics and Space Administration (NASA), which are used to build software defect prediction models. The measurement attributes include Halstead [35], McCabe [36], and lines of code. The data sets cover three programming languages. TABLE I presents the detailed description of the data sets.

A. Evaluation measure and Parameter setting

Considering the class imbalanced distribution of software defect prediction data sets and the different requirements of software systems, multiple evaluation criteria are used to measure the performance of the prediction model. The evaluation measures in the experiments include G-mean, AUC and balance. TABLE II defines a confusion matrix for performance evaluation. The probability of detection PD and the probability of false alarm PF are used to measure the performance on the defect class are defined as follows:

$$PD = TP/(TP + FN) \tag{14}$$

TABLE I: Summary description for software defect data sets, where e denotes the number of samples, m denotes the attribute number, d_m denotes the number of defective modules and d_r denotes the defective ratio

Dataset	language	e	m	d_m	$d\_r(\%)$
mc2	C++	161	39	44	32.29
kc2	C++	522	21	107	20.49
jm1	C	10885	21	2106	19.35
kc1	C++	2109	21	36	15.45
pc4	C	1458	37	326	12,20
pc3	C	1563	37	134	10.23
cm1	C	498	21	178	9.83
kc3	java	458	39	27	9.38
mw1	C	403	37	49	7.69
pc1	C	1109	21	77	6.94

$$PF = FP/(FP + TN) \tag{15}$$

G-mean is one widely-used criterion for imbalanced data, which achieves a better balance by maximizing the accuracy of each category. It is an evaluation criterion that calculate the geometric means of those accuracies within each class. The G-mean value is calculated by:

$$\text{G-mean } = \sqrt{PD \times (1 - PF)} \tag{16}$$

The area under receiver operating characteristic (AUC) measures performance of the classifier as a whole, which can be obtained by summing the area under the ROC curve [37]. AUC is not sensitive to the distribution between different classes evaluation metric and is a general evaluation metric of SDP.

Considering that the point (PF = 0, PD = 1) is the most ideal point in the ROC curve, which represents all the modules can be correctly identified. A comprehensive indicator balance is frequently used in software engineering. This indicator calculates the Euclidean distance from the (PF, PD) point to (0,1) as follows:

$$\text{balance } = 1 - \frac{\sqrt{(0 - PF)^2 + (1 - PD)^2}}{\sqrt{2}} \tag{17}$$

For the algorithms associated with BLS, the number of feature nodes and enhancement nodes are both set in the range $(50, 1000)$ by grid search. For WELM, we follow [26]. The optimal result is explored by the grid search of C $\{2^{-18}, 2^{-16}, \cdots, 2^{48}, 2^{50}\}$ and L $\{10, 20, \cdots, 990, 1000\}$. The results of DNC are referenced from [32]. We employ 10-fold cross-validation (CV) to evaluate the performances of different models. The average value and the corresponding standard deviation of evaluate metrics are used as the final results by repeating the above procedure 10 times.

TABLE II: Confusion matrix of SDP

Actual Class	**Predicted Class**	
	Defective(positive)	Non-defective(negative)
Defective(positive)	TP	FN
Non-defective(negative)	FP	TN

B. Experimental Analysis

To show the effectiveness of our proposed method, comparisons are conducted by AWBLS and other approaches. The counterparts include RUS-BLS, SOMTE-BLS, DNC [32], WELM1 [26] and WELM2 [26]. Among them, RUS-BLS represents the combination of random under-sampling strategy and BLS, while SOMTE-BLS represents sampling method SMOTE [18] combined with BLS. Table III - Table V present the comparison results between AWBLS and other methods by AUC, G-mean and balance, respectively.

According to the AUC results depicted in Table III, we can observe that AWBLS obtains the best performance on 8 out of 10 data sets. AWBLS is improved over other re-sampling based imbalance learning algorithms and two WELM algorithms, and it is also better than the DNC method based on ensemble learning. The G-mean results in Table IV demonstrate that AWBLS achieves best or comparable results on these data sets. For balance, the commonly used evaluation metric by software engineers, the results in Table V indicate that AWBLS performs best on most data sets.

To further analyse the experimental results, Friedman test [38] is conducted to evaluate the difference of multiple approaches. The average rankings of AUC, Balance and G-mean are denoted as $Ranking_A$, $Ranking_B$ and $Ranking_G$, respectively. As the results shown in Tables VI, the average rankings of AWBLS are the highest among all the three measurements, which indicates the superior performance of AWBLS. There are two reasons: (1) The weight matrix incorporated into BLS alleviates the impact of class imbalance problem in SDP. (2) The weight update mechanism combined with the prior distribution information further improves the prediction model, while reducing the negative impact of noise and outliers.

TABLE VI: Average rankings of the algorithms (Friedman test).

Algorithm	$Ranking_A$	$Ranking_G$	$Ranking_B$
AWBLS	1.40	1.90	1.40
DNC	2.40	2.99	3.09
WELM1	3.59	3.59	3.59
WELM2	3.59	2.40	2.90
SMOTE-BLS	5.49	5.79	5.79
RUS-BLS	4.49	4.20	4.20

V. Conclusion and future work

In this paper, we propose an adaptive weighted broad learning system for software defect prediction. First, to alleviate the impact of imbalance in SDP, we design a weighted BLS which assigns a weight to each module. Next, a density based weight generation mechanism considering both the inter-class and intra-class information is adopted to guide the weight matrix generation adaptively, which aims at reducing negative effects of noise or outlier modules. We conduct experiments on NASA software defect prediction data sets and draw the following conclusions: i) the weighted BLS can deal with the imbalance problem of SDP ; ii) the adaptive weight

TABLE III: A comparison of AWBLS and other methods by AUC

Dataset	SMOTE-BLS	RUS-BLS	WELM1	WELM2	DNC	AWBLS
mc2	0.729±0.025	0.715±0.023	0.619±0.031	0.646±0.031	0.649±0.142	**0.743**±0.025
kc2	0.525±0.025	0.666±0.009	0.798±0.012	0.792±0.015	0.828±0.074	**0.836**±0.005
jm1	0.626±0.007	0.602±0.014	0.689±0.019	0.699±0.016	**0.766**±0.016	0.744±0.013
kc1	0.551±0.027	0.601±0.013	0.786±0.004	0.793±0.033	0.818±0.034	**0.829**±0.002
pc4	0.720±0.021	0.829±0.009	0.868±0.009	0.867±0.006	0.917±0.031	**0.927**±0.001
pc3	0.575±0.026	0.682±0.029	0.803±0.006	0.795±0.008	0.816±0.056	**0.827**±0.001
cm1	0.677±0.064	0.738±0.025	0.730±0.023	0.732±0.021	0.787±0.097	**0.813**±0.006
kc3	0.566±0.042	0.657±0.040	0.817±0.011	**0.819**±0.012	0.797±0.102	0.773±0.005
mw1	0.587±0.033	0.668±0.044	0.750±0.019	0.749±0.025	0.714±0.139	**0.764**±0.018
pc1	0.597±0.026	0.799±0.014	0.789±0.008	0.788±0.012	0.866±0.081	**0.872**±0.004

TABLE IV: A comparison of AWBLS and other methods by G-mean

Dataset	SMOTE-BLS	RUS-BLS	WELM1	WELM2	DNC	AWBLS
mc2	0.613±0.048	0.644±0.037	0.575±0.042	0.617±0.029	0.571±0.152	**0.654**±0.026
kc2	0.574±0.033	0.659±0.009	0.721±0.005	0.728±0.007	**0.777**±0.071	0.762±0.003
jm1	0.601±0.008	0.619±0.025	0.633±0.019	0.637±0.012	0.672±0.017	**0.679**±0.012
kc1	0.611±0.024	0.626±0.012	0.707±0.005	0.726±0.026	0.734±0.040	**0.747**±0.002
pc4	0.677±0.025	0.767±0.020	0.815±0.010	0.819±0.009	**0.859**±0.047	0.815±0.006
pc3	0.590±0.017	0.651±0.017	0.730±0.011	0.734±0.007	0.745±0.061	**0.762**±0.003
cm1	0.591±0.071	0.693±0.026	0.665±0.034	0.671±0.025	0.659±0.133	**0.740**±0.018
kc3	0.524±0.061	0.619±0.053	**0.733**±0.014	0.729±0.011	0.662±0.160	0.707±0.009
mw1	0.449±0.073	0.592±0.061	0.631±0.044	**0.668**±0.032	0.647±0.272	0.657±0.032
pc1	0.596±0.019	0.740±0.019	0.693±0.016	0.747±0.008	0.698±0.145	**0.750**±0.006

TABLE V: A comparison of AWBLS and other methods by balance

Dataset	SMOTE-BLS	RUS-BLS	WELM1	WELM2	DNC	AWBLS
mc2	0.606±0.047	0.633±0.035	0.580±0.033	0.611±0.025	0.569±0.134	**0.644**±0.022
kc2	0.562±0.027	0.654±0.008	0.721±0.019	0.713±0.027	**0.777**±0.071	0.755±0.005
jm1	0.591±0.008	0.618±0.013	0.628±0.009	0.629±0.011	0.672±0.017	**0.679**±0.010
kc1	0.593±0.023	0.620±0.013	0.704±0.006	0.710±0.022	0.733±0.042	**0.741**±0.002
pc4	0.664±0.026	0.762±0.020	0.810±0.011	0.814±0.008	**0.850**±0.043	0.819±0.006
pc3	0.585±0.018	0.648±0.017	0.723±0.011	0.729±0.008	0.739±0.064	**0.758**±0.005
cm1	0.591±0.061	0.681±0.025	0.658±0.029	0.661±0.022	0.653±0.117	**0.726**±0.017
kc3	0.557±0.039	0.615±0.039	0.703±0.017	0.706±0.012	0.655±0.143	**0.709**±0.007
mw1	0.514±0.043	0.588±0.048	0.642±0.026	0.664±0.025	0.623±0.177	**0.669**±0.025
pc1	0.592±0.018	0.730±0.019	0.684±0.015	0.738±0.009	0.682±0.133	**0.743**±0.008

generation mechanism with the prior distribution information of original data can further improve the prediction model; iii) Experimental results on real-world SDP data sets reveal the superiority of AWBLS over other methods. As a future work, we will explore the properties of AWBLS and investigate the scalability on cross-project defect prediction and semi-supervised software defect prediction.

REFERENCES

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

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

[3] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[4] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, 2011.

[5] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, and R. Mishra, "Predicting software defects in varying development lifecycles using bayesian nets," *Information and Software Technology*, vol. 49, no. 1, pp. 32–43, 2007.

[6] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2019.

[7] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[8] X. Jiang, S. Pan, G. Long, J. Chang, J. Jiang, and C. Zhang, "Cost-sensitive hybrid neural networks for heterogeneous and imbalanced data," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

[9] C. Zhang, K. C. Tan, and R. Ren, "Training cost-sensitive deep belief networks on imbalance data problems," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4362–4367.

[10] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 1, pp. 10–24, 2018.

[11] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.

[12] B. Igelnik and P. Yoh-Han, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.

[13] Y. LeCun, H. Fu Jie, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, pp. II–104 Vol.2.

[14] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 4, pp. 1191–1204, 2019.

[15] S. Feng and C. L. P. Chen, "Fuzzy broad learning system: A novel neuro-fuzzy model for regression and classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 414–424, 2020.

[16] Z. Liu, B. Chen, B. Xie, H. Qiang, and Z. Zhu, "Feature selection for

orthogonal broad learning system based on mutual information," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.
[17] J.-W. Jin and C. L. Philip Chen, "Regularized robust broad learning system for uncertain data modeling," *Neurocomputing*, vol. 322, pp. 58–69, 2018.
[18] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 321–357, 2002.
[19] H. Haibo, B. Yang, E. A. Garcia, and L. Shutao, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1322–1328.
[20] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Advances in Knowledge Discovery and Data Mining* (T. Theeramunkong, B. Kijsirikul, N. Cercone, and T.-B. Ho, eds.), pp. 475–482, Springer Berlin Heidelberg.
[21] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, and J.-S. Jhang, "Clustering-based undersampling in class-imbalanced data," *Information Sciences*, vol. 409-410, pp. 17–26, 2017.
[22] K. Yang, Z. Yu, X. Wen, W. Cao, C. L. P. Chen, H. Wong, and J. You, "Hybrid classifier ensemble for imbalanced data," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2019.
[23] L. Zhang and D. Zhang, "Evolutionary cost-sensitive extreme learning machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 12, pp. 3045–3060, 2017.
[24] X. Liu and Z. Zhou, "The influence of class imbalance on cost-sensitive learning: An empirical study," in *Sixth International Conference on Data Mining (ICDM'06)*, pp. 970–974.
[25] B. Krawczyk, M. Woźniak, and G. Schaefer, "Cost-sensitive decision tree ensembles for effective imbalanced classification," *Applied Soft Computing*, vol. 14, pp. 554–562, 2014.
[26] W. Zong, G.-B. Huang, and Y. Chen, "Weighted extreme learning machine for imbalance learning," *Neurocomputing*, vol. 101, p. 229C242, 2013.
[27] W. Fan, S. Stolfo, J. Zhang, and P. Chan, "Adacost: Misclassification cost-sensitive boosting," *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, 1999.
[28] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *Engineering Applications of Neural Networks* (D. Palmer-Brown, C. Draganova, E. Pimenidis, and H. Mouratidis, eds.), pp. 223–234, Springer Berlin Heidelberg.
[29] M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *Journal of Systems and Software*, vol. 76, no. 2, pp. 147–156, 2005.
[30] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.
[31] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.
[32] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.
[33] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.
[34] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005.
[35] P. Freeman, A. I. Wasserman, and R. E. Fairley, "Essential elements of software engineering education," in *Proceedings of the 2nd International Conference on Software Engineering*, ICSE 76, (Washington, DC, USA), p. 116C122, IEEE Computer Society Press, 1976.
[36] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.
[37] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," *Machine Learning*, vol. 31, pp. 1–38, 2004.
[38] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of The American Statistical Association - J AMER STATIST ASSN*, vol. 32, pp. 675–701, 1937.