Two-Stage Cost-Sensitive Learning for Software Defect Prediction

Mingxia Liu, Linsong Miao, and Daoqiang Zhang

***Abstract*—Software defect prediction (SDP), which classifies software modules into defect-prone and not-defect-prone categories, provides an effective way to maintain high quality software systems. Most existing SDP models attempt to attain lower classification error rates other than lower misclassification costs. However, in many real-world applications, misclassifying defect-prone modules as not-defect-prone ones usually leads to higher costs than misclassifying not-defect-prone modules as defect-prone ones. In this paper, we first propose a new two-stage cost-sensitive learning (TSCS) method for SDP, by utilizing cost information not only in the classification stage but also in the feature selection stage. Then, specifically for the feature selection stage, we develop three novel cost-sensitive feature selection algorithms, namely, Cost-Sensitive Variance Score (CSVS), Cost-Sensitive Laplacian Score (CSLS), and Cost-Sensitive Constraint Score (CSCS), by incorporating cost information into traditional feature selection algorithms. The proposed methods are evaluated on seven real data sets from NASA projects. Experimental results suggest that our TSCS method achieves better performance in software defect prediction compared to existing single-stage cost-sensitive classifiers. Also, our experiments show that the proposed cost-sensitive feature selection methods outperform traditional cost-blind feature selection methods, validating the efficacy of using cost information in the feature selection stage.**

***Index Terms*—Cost-sensitive learning, feature selection, software defect prediction.**

Acronyms and Abbreviations

SDP	Software Defect Prediction
TSCS	Two-Stage Cost-Sensitive learning
VS/LS/CS	Variance/Laplacian/Constraint Score
CSVS	Cost-Sensitive Variance Score
CSLS	Cost-Sensitive Laplacian Score
CSCS	Cost-Sensitive Constraint Score
NN	Neural Network
CSNN	Cost-Sensitive Neural Network

Notations

N	Number of samples
N_k	Number of samples for the kth class
d	Number of features
c	Number of classes
x_i	The ith sample
f_{ri}	The rth feature of sample x_i
μ_r	The mean of the rth feature
$\mathbf{M}$	Pair-wise must-link constraint set
$\mathbf{C}$	Pair-wise cannot-link constraint set
$\phi(i)$	The importance value of the ith class
$cost(i,j)$	Cost of classifying a sample from class i as class j

Manuscript received June 02, 2013; revised October 31, 2013; accepted November 12, 2013. Date of publication April 21, 2014; date of current version May 29, 2014. This work was supported in part by the Jiangsu Natural Science Foundation for Distinguished Young Scholar (Grant BK20130034), the Specialized Research Fund for the Doctoral Program of Higher Education (Grant 20123218110009), the NUAA Fundamental Research Funds (Grants NE2013105, NZ2013306), the Jiangsu Qinglan Project, the Funding of Jiangsu Innovation Program for Graduate Education (Grant CXZZ13_0173), and the National Natural Science Foundation of China (Grant 61379015). Associate Editor: J.-C. Lu.

M. Liu is with the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China. She is also with the School of Information Science and Technology, Taishan University, Taian 271021, China (e-mail: mingxialiu@nuaa.edu.cn).

L. Miao and D. Zhang are with the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China (e-mail: dqzhang@nuaa.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2014.2316951

I. Introduction

SOFTWARE DEFECT PREDICTION (SDP) plays an important role in reducing the costs of software development and maintaining the high quality of software systems [1]–[3]. It allows software project managers to allocate limited time and manpower resources to defect-prone modules through early defect detection. Existing SDP work can be categorized into three types [1]: 1) estimating the number of defects existing in a software system, 2) mining defect associations, and 3) classifying software modules into defect-prone and not-defect-prone categories.

The first type of work attempts to estimate the number of defects existing in a software system based on code metrics, inspection data, and process quality data. Various methods have been applied, such as statistical approaches [2]–[5], detection profile methods [6], and capture-recapture models [7]–[10]. The second type of work is usually based on data mining technologies (e.g., association mining algorithms [11], [12]) to capture software defect associations, and to discover

rule sets that may cause a larger number of defects. The third type of work classifies the software modules into defect-prone and not-defect-prone categories by using machine learning techniques, such as ensemble learning [13], tree-based methods [14], neural networks [15]–[17], and analogy-based methods [18]. These methods predict the defect-proneness of new software modules by building a binary classifier with historical data represented by software metrics [19]–[21].

Unfortunately, software defect prediction still remains a difficult problem to be solved, and is faced with two challenges [22]–[24]: high dimensionality, and class imbalance. As modern software systems grow in both size and complexity, the number of features (i.e., software metrics) extracted from software modules becomes much larger than ever before, and these features may be redundant or irrelevant [23]. It is a great challenge for classification algorithms to deal with such superabundant features. As an important pre-processing procedure, feature selection is beneficial to facilitate data understanding, to reduce the storage requirements, and to overcome the curse of dimensionality for improved prediction performance [25], [26]. As shown in previous studies [23], [24], [27], [28], feature selection is effective to deal with the high dimensionality problem in SDP.

The second challenge for SDP is the class-imbalanced data, where the majority of defects in a software system are only found in a small portion of its modules [29]. In such cases, standard machine learning based SDP models may be inaccurate for (or never predict) the minority class (i.e., the defect-prone module), because they do not explicitly consider different error costs or class distributions. There are two approaches drawn from machine learning for addressing that problem [22], i.e., stratification and cost-sensitive learning. Stratification has been investigated in several studies of SDP [30]–[32], by creating a balanced data set through adding more samples to the minority class (over-sampling) or reducing the sample number of the majority class (under-sampling). Recently, cost-sensitive learning has attracted increasing attention in the SDP domain [17], [33], [34], which explicitly considers those different error costs, and aims to minimize the total expected costs rather than the classification error rates. In general, there are two types of errors in software defect prediction [34]. Type I is defined as misclassifying a not-defect-prone module as defect-prone, while Type II misclassification is when a defect-prone module is predicted as not-defect-prone. The cost incurred by Type II misclassification is much higher than that of Type I misclassification [34].

However, in most cost-sensitive learning based SDP studies, cost information is used in the classification stage instead of in the feature selection stage. But considering the valuable cost information in the feature selection stage may further boost the performances of SDP models because features associated with the minority class (i.e., defect-prone modules) are more likely to be selected. The goal of this paper is to develop a two-stage cost-sensitive (TSCS) learning method for SDP by using cost information in both the classification and the feature selection stages. Work has been done on cost-sensitive feature selection; we also develop three novel cost-sensitive feature selection algorithms by emphasizing samples with higher misclassification costs, and de-emphasizing those with lower misclassification costs in the feature selection stage. The experimental results on the public NASA Metrics Data Program repository [19] validate the efficacy of our proposed methods.

The remainder of this paper is organized as follows. We first review related work of software defect prediction, cost-sensitive learning, and feature selection for SDP in Section II. Then, the proposed two-stage cost-sensitive learning method for SDP and three cost-sensitive feature selection algorithms are described in Section III. Section IV presents the experiments on real SDP benchmark data sets. Finally, we conclude this paper in Section V.

II. Related Work

A. Software Defect Prediction

SDP can be formulated as a binary classification problem, where software modules are classified as defect-prone or not-defect-prone, using a set of software metrics. There are many kinds of software metrics collected from previously developed systems by standard tools [35]. The first suite of software metrics, known as CK metrics, was developed by Chidamber and Kemerer [20]. Lorenz and Kidd [21] proposed additional object-oriented metrics dealing with message and inheritance passing in a class. Many other software metrics such as code metrics [35]–[37], process metrics [38]–[40], and previous defaults [41], [42] were subsequently developed.

In the literature, many machine learning techniques, including parametric and nonparametric methods, have been applied for constructing SDP models [22], [43]. Parametric methods utilize the relationship between software complexity metrics and the occurrence of faults in program modules to construct SDP models, including case-based reasoning [37], [44], regression trees [45], and multiple linear regression [46]. Because the relationships between software metrics and defect-proneness of software modules are often complicated, traditional parametric models cannot predict defect occurrence or rates accurately [16], [22], [47]. To overcome the shortcoming of parametric methods, various nonparametric methods are used to build SDP models. Some examples include ensemble learning [13], [48], tree-based methods [14], [45], [49], neural networks [15]–[17], analogy-based methods [18], genetic programming [50], [51], clustering [52], [53], kernel-based methods [54]–[56], cost-sensitive learning [17], [34], and transfer learning [57].

B. Cost-Sensitive Learning for Software Defect Prediction

The class imbalance problem corresponds to the problem encountered by a learning system where the sample size of one class is significantly larger than those of other classes [58]. This problem is prevalent in many real-world applications (e.g. software defect detection [34], face recognition [59], [60], and credit card fraud detection [61]), and remains a significant bottleneck in the performance of standard learning methods as they tend to be overwhelmed by the majority classes, and thus ignore the minority categories [62], [63].

For addressing the class imbalance problem, a kind of learning algorithm called cost-sensitive learning has been studied in the machine learning and data mining community [59], [64]–[71]. In cost-sensitive learning, cost information

is used to evaluate the misclassification costs from different types of errors. Other than focusing on lower classification error rates, cost-sensitive learning methods aim to minimize the total expected costs by utilizing the cost information [49], [60], [63]–[65], [72]. The misclassification cost can be further categorized into two classes [66], i.e., class-dependent cost [60], [64], [66], and example-dependent cost [67]. The former assumes that misclassifying any example of class i as class j will lead to the same loss, while the latter assume that different examples have different costs even if they share the same type of errors. Comparatively, class-dependent cost is often used because it is easier to obtain than example-dependent cost.

In the SDP domain, many cost-sensitive classification methods have been applied and shown effective to deal with the class imbalance problem [4], [17], [33], [34], [73]. Khoshgotaar *et al.* first introduced cost-sensitive learning into software defect prediction [4], and used a boosting method to build software quality models [34]. Ling *et al.* [33] developed a system to mine the historic defect report data, and to predict the escalation risk of current defect reports for maximum Return On Investment (ROI), where the maximum ROI problem is converted to a cost-sensitive learning problem. Zheng [17] developed three cost-sensitive boosting algorithms to boost neural networks for SDP. The first algorithm using a threshold-moving strategy tries to move the classification threshold towards the not-defect-prone modules. The other two weight-updating based algorithms incorporate the misclassification costs into the weight-update rule of the boosting procedure.

C. Feature Selection for Software Defect Prediction

Feature selection has been widely used in many pattern recognition and machine learning applications for decades [25], [74]. The aim of feature selection is to find the minimally sized feature subset that is necessary and sufficient for a specific task [74]. Typically, feature selection can be categorized into three classes [25]: 1) wrapper-type methods, 2) embedded-type methods, and 3) filter-type methods. Wrapper-type methods use a learning machine of interest as a black box to evaluate each candidate feature subset according to their predictive power, and are usually computationally expensive [75], [76]. Embedded-type methods perform feature selection in the process of training, and are specific to given learning algorithms. Unlike wrapper-type and embedded-type methods, filter-type methods select features according to some criteria (e.g., mutual information [77]), and involve no learning algorithm. Hence, filter-type methods are usually adopted in practice due to their simplicity and computational efficiency [78], [79]. Also, previous studies have shown that filter-type feature selection techniques are simple, quick, yet effective to reduce the feature dimension for SDP [24]. Thus, in this paper, we focus on the filter-type feature selection.

Here, we briefly introduce three popular filter-type feature selection methods (including Variance [80], Laplacian Score [81], and Constraint Score [78]), which are relevant to our proposed methods. Variance score (VS) is a simple unsupervised evaluation criterion of features. It selects features that have the maximum variance among all samples, with the basic idea that the variance among a feature space reflects the representative power of this feature. As another popular unsupervised feature selection method, Laplacian Score (LS) not only prefers features with larger variances which have more representative power, but also prefers features with stronger locality preserving ability [81]. Constraint Score (CS) is a semi-supervised feature selection method, which performs feature selection according to the constraint preserving ability of features [78]. It uses must-link and cannot-link pair-wise constraints as supervision information, where features that can best preserve the must-link constraints as well as the cannot-link constraints are assumed to be important.

In software engineering, several feature selection methods have been used [23], [27], [82]. Menzies *et al.* [82] employ exhaustive InfoGain subsetting to reduce the dimension, and speed up the learning of the SDP model. Rodriguez *et al.* [27] apply feature selection to software engineering data sets, and conclude that the reduced data sets with fewer features maintain or improve the prediction capability over the original data sets. Gao *et al.* [23] propose a hybrid feature selection method composed of a feature ranking and a feature subset selection stage. However, to the best of our knowledge, existing feature selection methods designed for SDP are cost-blind, i.e., the issue of different costs for different errors is not considered.

III. Proposed Methods

A. Cost Information

In most real-world applications, different misclassifications are usually associated with different costs [49], [64], [65]. Denote class labels as $\{1, \ldots, c\}$. Without loss of generality, we assume that misclassifying a sample from the ith class ($i \in \{1, \ldots, c-1\}$) as the cth class will cause higher costs than misclassifying a sample of the cth class as other classes. Here, we call the class from the 1st class to the $(c-1)$th class the in-group class, while the cth class is called the out-group class. Then, we can categorize misclassification costs into three types: 1) the cost of false acceptance C_{OI}, i.e., the cost of misclassifying a sample from the out-group class as being from the in-group class; 2) the cost of false rejection C_{IO}, i.e., the cost of misclassifying a sample from the in-group class as being from the out-group class; and 3) the cost of false identification C_{II}, i.e., the cost of misclassifying a sample from one in-group class as being from another in-group class.

In this work, we assume $C_{\mathrm{OI}} = C_{\mathrm{OI}}/C_{\mathrm{II}}$, $C_{\mathrm{IO}} = C_{\mathrm{IO}}/C_{\mathrm{II}}$, and $C_{\mathrm{II}} = 1$, as this will not change the final results [60], [66]; and we denote the defect-prone module as being from the out-group class, and the not-defect-prone module as being from the in-group class. Then, we construct a cost matrix as shown in Table I, where the element $cost(i, j)$ $(i, j \in \{1, \ldots, c\})$ indicates the cost value of classifying a sample from the ith class as the jth class. The diagonal elements in the cost matrix are zero because a correct classification will cause no cost.

Similar to the work in [59], we utilize the function $\phi(i)$to describe the importance of the ith class ($i \in \{1, \ldots, c\}$), which is defined as

$$\phi(i) = \begin{cases} (c-2)C_{\mathrm{II}} + C_{\mathrm{IO}}, & if\ i = 1, \ldots, c-1 \\ (c-1)C_{\mathrm{OI}}, & otherwise \end{cases} \quad (1)$$

TABLE I
THE COST MATRIX

	I_1	...	I_{c-1}	O
I_1	0	...	C_{II}	C_{IO}
...	...	...	...	...
I_{c-1}	C_{II}	...	0	C_{IO}
O	C_{OI}	...	C_{OI}	0

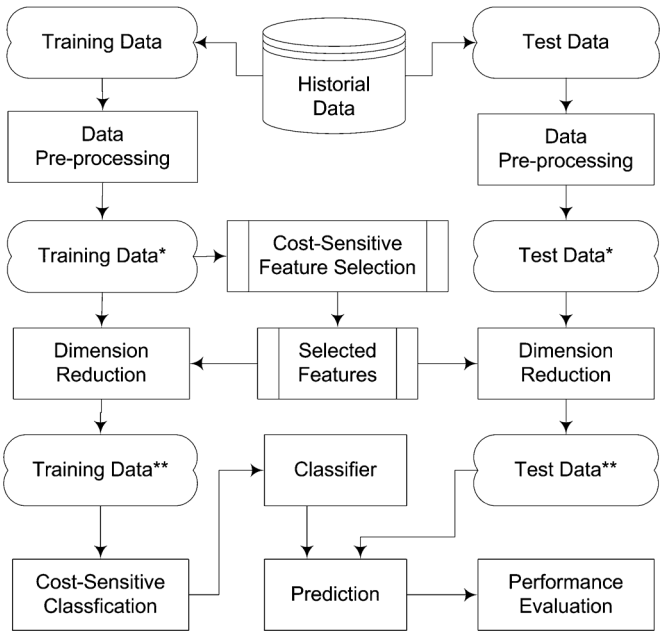

Fig. 1. Flowchart of the proposed TSCS method.

where c is the number of classes including the in-group and out-group classes. A larger value of $\phi(i)$ means that the ith class is more important.

B. Two-Stage Cost-Sensitive Learning

To fully utilize the valuable cost information, we propose a two-stage cost-sensitive learning (TSCS) method for software defect prediction where the cost information is used in both the feature selection stage and the classification stage. The cost-sensitive feature selection aims to select features that are associated with the interesting class (i.e., defect-prone module), and the cost-sensitive classification deems to make the SDP classifier not dominated by the majority class (i.e., not-defect-prone module). The above two stages are used to solve the class imbalance problem in SDP. The flowchart in Fig. 1 illustrates a general architecture of the proposed TSCS method.

As shown in Fig. 1, the historical data, including various software metrics captured from software systems, are divided into two groups: the training data set, and the test data set. These data are pre-processed before being fed into the following feature selection and classification algorithms. In the second stage, cost-sensitive feature selection algorithms are applied to the training data to find the optimal features, and thus the dimension can be reduced. The next step is to train the cost-sensitive classification models based on the training data set with selected features. Finally, the learned model is evaluated on the test data set.

The proposed TSCS is a general method because any type of cost-sensitive feature selection method can be used in the feature selection stage, and any kind of cost-sensitive classifiers can be used in the classification stage. However, to the best of our knowledge, few works have been done on cost-sensitive feature selection. In the following, we develop three novel cost-sensitive feature selection algorithms by considering different costs for different errors in the feature selection stage.

C. Cost-Sensitive Feature Selection

Assume we have a set of samples $\mathbf{X} = [x_1, \ldots, x_N]$ $(x_i \in R^d)$, where N is the number of samples, and d is the feature dimension. Let f_{ri} denote the rth feature of sample x_i. Denote $\mu_r = (1/N)\sum_{i=1}^{N} f_{ri}$ as the mean of the rth feature among all samples, and N_k as the number of samples belonging to the kth class. Denote $\mathbf{M} = \{(x_i, x_j)|x_i \text{ and } x_j \text{ belong to the same class}\}$ as the must-link constraints set, and $\mathbf{C} = \{(x_i, x_j)|x_i \text{ and } x_j \text{ belong to different classes}\}$ as the cannot-link constraints set.

1) CSVS: Similar to Variance score, we assume that the variance of a good feature in the out-group class should be larger than that of the in-group classes. Thus, the Cost-Sensitive Variance Score (CSVS) of the rth feature denoted as $CSVS_r$, which should be maximized, is defined as

$$CSVS_r = \frac{\frac{(c-1)}{N_c}\sum_{j=1}^{N_c}\phi(c)(f_{rj}-\mu_r)^2}{\sum_{i=1}^{c-1}\frac{1}{N_i}\sum_{j=1}^{N_i}\phi(i)(f_{rj}-\mu_r)^2} \tag{2}$$

where N_i is the sample number of the ith class, and $\phi(i)$ is the importance value of the ith class that can be obtained from (1).

2) CSLS: To facilitate balancing the variance and locality preserving ability of features, we define the Cost-Sensitive Laplacian Score (CSLS) of the rth feature, denoted as $CSLS_r$, as

$$CSLS_r = \sum_{i=1}^{N}\sum_{j=1}^{N}[-cost(y_i, y_j)](f_{ri}-f_{rj})^2 S_{ij} - \gamma\sum_{i=1}^{N}\phi(y_i)(f_{ri}-\mu_r)^2 D_{ii} \tag{3}$$

where $\mathbf{D}$ is a diagonal matrix with element $D_{ii}\sum_j S_{ij}$, and S_{ij} is defined by the neighborhood relationship between two samples x_i and x_j as

$$S_{ij} = \begin{cases} e^{-\frac{\|x_i - x_j\|^2}{t}}, & if\ x_i\ and\ x_j\ are\ k\ neighbors \\ 0, & otherwise \end{cases} \tag{4}$$

where t is a constant that is set to be the average distance between training samples in this paper. And the term 'x_i and x_j are k neighbors' means that x_i is among the k nearest neighbors of x_j, or x_j is among the k nearest neighbors of x_i.

In (3), y_i, and $y_j (y_i, y_j \in \{1, \ldots, c\})$ are class labels for samples x_i, and x_j, respectively. The term $cost(y_i, y_j)$ denotes the cost of classifying a sample from the y_ith class to the y_jth classs, which can be obtained from the cost matrix given in Table I. The

TABLE II
DETAILED INFORMATION OF MDP DATA SETS AFTER PRE-PROCESSING

Data Set	Language	System	# Modules	% Defective
CM1	C	Spacecraft instrument	505	9.50
KC2	Java	Storage management for ground data	522	20.50
MW1	C	Db	403	7.96
PC1	C	Flight software for earth orbiting satellite	1107	6.87
PC2	C		5589	0.41
PC3	C		1563	10.24
PC4	C		1458	12.21

term $\phi(y_i)$ denotes the importance value of class y_i defined in (1). The regularization coefficient γ is used to trade off the contribution of the two terms in (3). As $\phi(y_i)$ is usually larger than $cost(y_i, y_j)$, we set $\gamma < 1$ in this work. In (3), we assume important features have smaller $CSLS$ values.

3) CSCS: Taking the cost information into consideration, we formulate the Cost-Sensitive Constraint Score (CSCS) of the rth feature denoted as $CSCS_r$, which should be minimized, as

$$CSCS_r = \sum_{(x_i,x_j)\in\mathbf{M}} \phi(y_i)(f_{ri} - f_{rj})^2 - \lambda \sum_{(x_i,x_j)\in\mathbf{C}} cost(y_i, y_j)(f_{ri} - f_{rj})^2 \quad (5)$$

where $\mathbf{M}$ is a set of pair-wise must-link constraints, $\mathbf{C}$ is a set of pair-wise cannot-link constraints, and λ is a regularization coefficient to trade-off the two terms in (5). Because $\phi(y_i)$ is usually larger than $cost(y_i, y_j)$, and the distance between samples in the same class is typically smaller than that in different classes, we set $\lambda < 1$ in this work.

IV. EXPERIMENTS

A. Data Sets

The data sets used in this study come from the public NASA Metrics Data Program (MDP) repository [19], making our proposed methods repeatable and verifiable. These data sets, including CM1, KC2, MW1, PC1, PC2, PC3, and PC4, belong to several NASA projects, which are also used in [82]. Similar to [56], we pre-process the data from each data set. The characteristics of these data sets after pre-processing are shown in Table II.

B. Performance Measurements

For better evaluating the performances in the cost-sensitive learning scenarios, the *Total-cost* of misclassification, which is a general measurement for cost-sensitive learning [49], [63]–[67], is used as one primary evaluation criterion in our experiments. On the other hand, as shown in Table III, the classification results can be represented by the confusion matrix with two rows and two columns reporting the number of true positives (*TP*), false positives (*FP*), false negatives (*FN*), and true negatives (*TN*).

TABLE III
DEFECT PREDICTION CONFUSION MATRIX

		Actual	
		Defect-prone	Not-defect-prone
Predicted	Defect-prone	*TP*	*FP*
	Not-defect-prone	*FN*	*TN*

From the confusion matrix, $Sensitivity$ and $Accuracy$ can be defined as

$$Sensitivity = \frac{TP}{(TP + FN)} \quad (6)$$

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \quad (7)$$

where $Sensitivity$ measures the proportion of defect-prone modules correctly classified, and $Accuracy$ measures the proportion of samples correctly classified among the whole population. In addition to the *Total-cost*, we also adopt the $Sensitivity$ and $Accuracy$ of the classification results as evaluation measures.

C. Experiment Design

In the first group of experiments, we compare the two-stage cost-sensitive learning method, i.e., the proposed TSCS, with existing single-stage cost-sensitive classifiers on MDP data sets. Among various existing cost-sensitive classifiers, the cost-sensitive back propagation neural network (CSNN) method with the threshold moving technique has been shown effective to build cost-sensitive SDP models [17]. Thus, we use CSNN with the threshold-moving strategy as the single-stage cost-sensitive classifier in this work. In the proposed TSCS, three cost-sensitive feature selection algorithms (CSVS, CSLS, and CSCS) are used in the feature selection stage, and CSNN is used in the classification stage. Also, the back propagation neural network (NN) [80] is used as a cost-blind classifier in the experiments. In the second group of experiments, we evaluate the proposed TSCS method, and the proposed cost-sensitive feature selection methods (i.e., CSVS, CSLS, and CSCS) in SDP, with comparison to conventional methods.

A 10-fold cross-validation strategy is used to compute the *Total-cost*, $Sensitivity$, and $Accuracy$ on the test set. To be

TABLE IV
RESULTS OF TWO-STAGE AND SINGLE-STAGE COST-SENSITIVE LEARNING METHODS ON MDP DATA SETS

Data Set	CSNN	Proposed TSCS Methods		
		CSVS+CSNN	CSLS+CSNN	CSCS+CSNN
Total-cost: Misclassification cost				
CM1	55.00±6.60	46.45±1.31(16)	**43.95±4.57(5)**	45.80±4.65(16)
KC2	30.68±2.83	27.43±1.38(9)	**27.37±1.43(11)**	27.83±1.85(3)
MW1	18.87±0.99	**14.88±0.94(5)**	17.82±0.94(14)	16.75±0.71(9)
PC1	95.56±1.85	82.15±0.65 (18)	**81.05±1.53(12)**	86.95±2.03(19)
PC2	33.05±5.90	30.40±4.40(1)	**29.70±4.50(28)**	32.05±6.20(1)
PC3	152.00±8.06	135.70±8.66(5)	**135.05±7.72(12)**	136.80±5.15(19)
PC4	132.00±5.80	**72.00±2.92(7)**	79.70±2.15(36)	77.60±2.14(31)
Sensitivity (%): Accuracy of defect-prone class				
CM1	47.22±0.86	61.11±1.89	**70.00±0.56**	59.44±0.56
KC2	76.55±2.51	81.42±1.31	81.18±1.47	**81.77±1.79**
MW1	41.00±0.42	**54.50±0.32**	45.42±0.32	49.83±0.24
PC1	55.67±1.73	56.00±0.94	**60.67±1.14**	52.00±2.95
PC2	13.75±2.79	6.25±0.13	**26.25±0.12**	17.50±0.20
PC3	62.19±0.39	**73.13±0.68**	69.38±1.03	70.63±0.76
PC4	83.43±0.26	**91.71±0.54**	85.29±0.68	86.86±0.71
Accuracy(%): Total accuracy				
CM1	75.81±1.53	77.60±0.42	74.44±0.56	**79.34±0.19**
KC2	73.07±1.64	74.07±0.59	**74.82±0.68**	72.55±0.68
MW1	85.12±1.66	**87.93±0.43**	85.06±0.59	85.72±0.60
PC1	83.28±1.93	**83.73±1.93**	82.01±2.23	83.46±5.07
PC2	99.20±0.13	**99.63±0.11**	99.19±0.20	99.20±0.30
PC3	78.53±0.71	75.80±0.39	**78.80±0.18**	77.24±0.39
PC4	84.35±0.72	82.23±1.09	**85.00±0.25**	84.40±0.16

specific, the whole data set is first randomly partitioned into ten equally sized subsets. Each time, one of these subsets is retained as the test data while the other nine subsets are used as the training data. To ensure a low bias of random partitioning, the cross-validation process is repeated ten times. For each performance measure, the mean is computed from the results of these ten runs.

For the proposed CSLS and CSCS methods, the parameters γ and λ are both set to be 0.5. Following the work in [79], we use equal numbers of must-link and cannot-link constraints in the experiments. To be specific, for each data set, a total of 100 pair-wise constraints including 50 must-link and 50 cannot-link constraints are used. For fair comparison, the conventional CS method and the proposed CSCS method share the same pool of pair-wise constraints. The fixed costs, i.e. $C_{\mathrm{IO}} = 2$ and $C_{\mathrm{OI}} = 10$, are used in the experiments. We also discuss the influence of different cost ratios on the experimental results, with more details given in Section IV-F.

D. Two-Stage vs. Single-Stage Cost-Sensitive Learning

In this subsection, we perform classification experiments by comparing the proposed two-stage cost-sensitive learning (i.e., TSCS) method to the existing single-stage cost-sensitive learning (i.e., CSNN) method. The experimental results are reported in Table IV, where numbers in the bracket represent the optimal features determined by the lowest *Total-cost* achieved by a specific feature selection method. In addition, Fig. 2 plots the results vs. different numbers of selected features achieved by the proposed TSCS method and the CSNN method on CM1 and KC2 data sets.

From Table IV, one can see that the proposed TSCS methods (including CSVS + CSNN, CSLS + CSNN, and CSCS + CSNN) consistently achieve lower *Total-cost* than CSNN on seven data sets. At the same time, the proposed CSLS + CSNN method usually performs better than the proposed CSVS + CSNN and CSCS + CSNN methods in

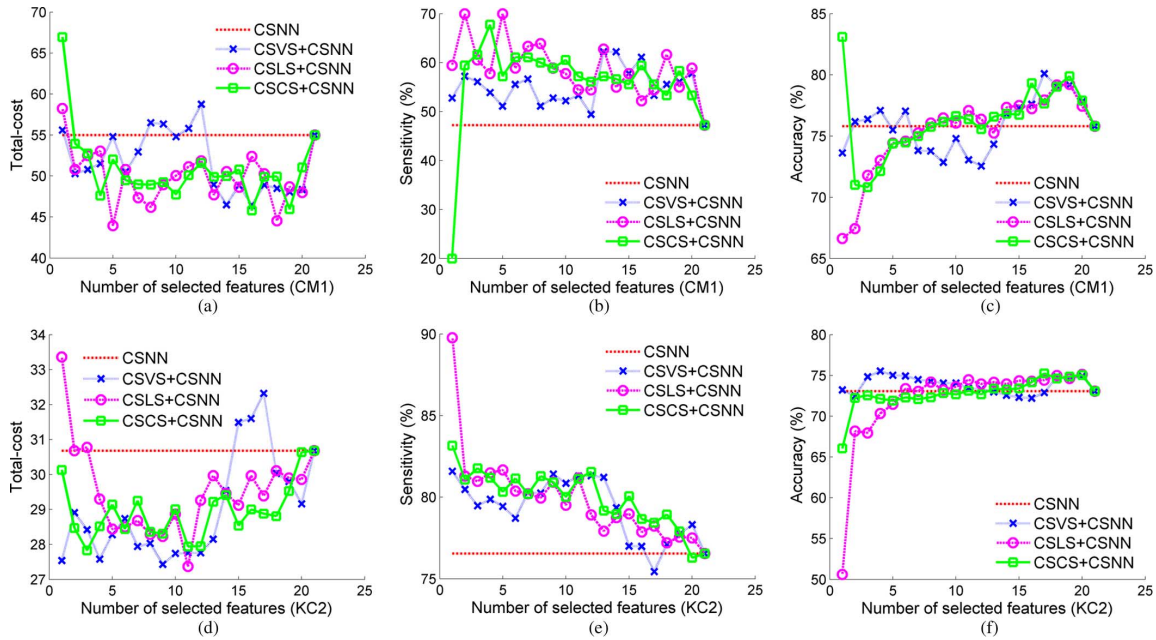

Fig. 2. Classification results vs. different number of selected features on CM1 and KC2 data sets achieved by the proposed TSCS (i.e. CSVS + CSNN, CSLS + CSNN, and CSCS + CSNN) methods, and the CSNN method.

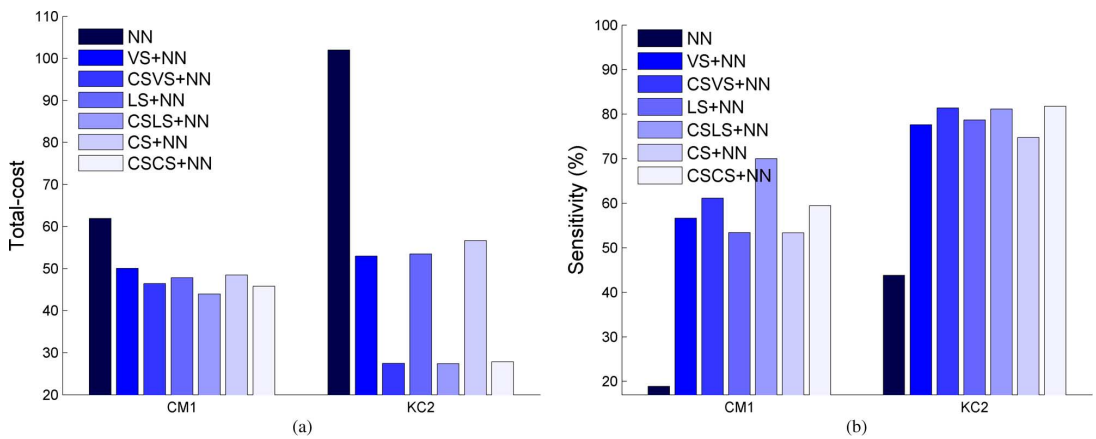

Fig. 3. Comparison of performances of different feature selection methods using NN classifier on CM1 and KC2 data sets.

reducing the *Total-cost*. In terms of $Sensitivity$, the three proposed TSCS methods usually outperform CSNN, especially on the CM1, KC2, and PC3 data sets. As for $Accuracy$, the proposed TSCS methods achieve competitive results compared to CSNN. It is worth noting that, on the PC2 data set which is extremely class imbalanced, the proposed CSLS + CSNN method achieves lower *Total-cost* as well as higher $Sensitivity$ than other methods. These results indicate that, compared to single-stage cost-sensitive classifiers, the proposed TSCS methods considering cost information in both feature selection and classification stages provide better solutions to deal with the class imbalance problem in SDP.

From Fig. 2(a) and (d), one can see that the proposed TSCS methods (i.e., CSVS + CSNN, CSLS + CSNN, and CSCS + CSNN) using less than one third of the features can achieve a lower *Total-cost* than CSNN on two data sets. For example, on the CM1 data set, the proposed CSLS + CSNN using only two selected features perform better than CSNN. From Fig. 2(b) and (e), it can be seen that, on both CM1 and KC2 data sets, the $Sensitivity$ achieved by the proposed TSCS using less than three features are significantly higher than that of CSNN. From Fig. 2(c) and (f), one can see that, in terms of $Accuracy$, the proposed TSCS and the CSNN methods achieve comparable results in most cases, even if the former uses a reduced number of features than the latter. These results again validate the advantages of our proposed TSCS methods over a conventional single-stage cost-sensitive classifier.

E. Cost-Sensitive vs. Cost-Blind Learning

In this subsection, we evaluate the proposed TSCS, and three cost-sensitive feature selection methods in SDP, with comparison to conventional methods including VS + NN, LS + NN, CS + NN, VS + CSNN, LS + CSNN, and CS + CSNN. The classification results achieved by cost-sensitive and cost-blind feature selection methods on CM1 and KC2 data sets, using a cost-blind classifier (i.e., NN), are shown in Fig. 3.

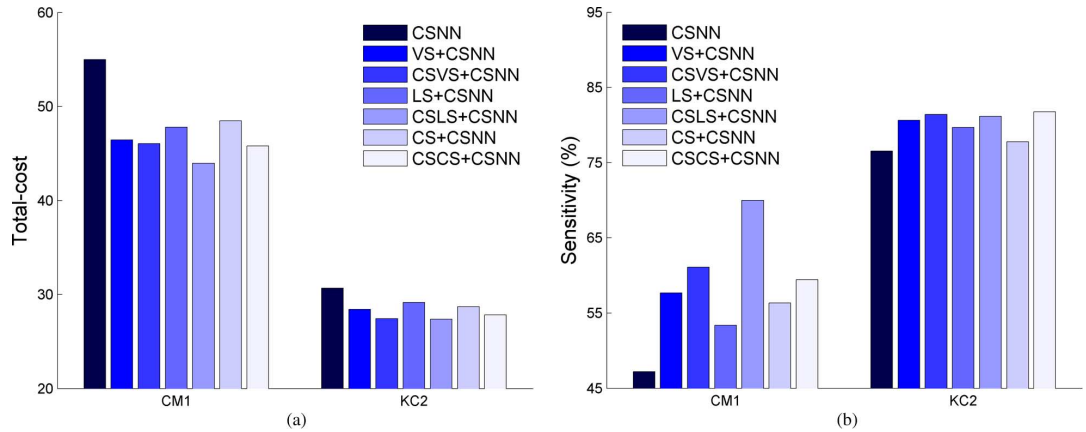

Fig. 4. Comparison of performances of different feature selection methods using CSNN classifier on CM1 and KC2 data sets.

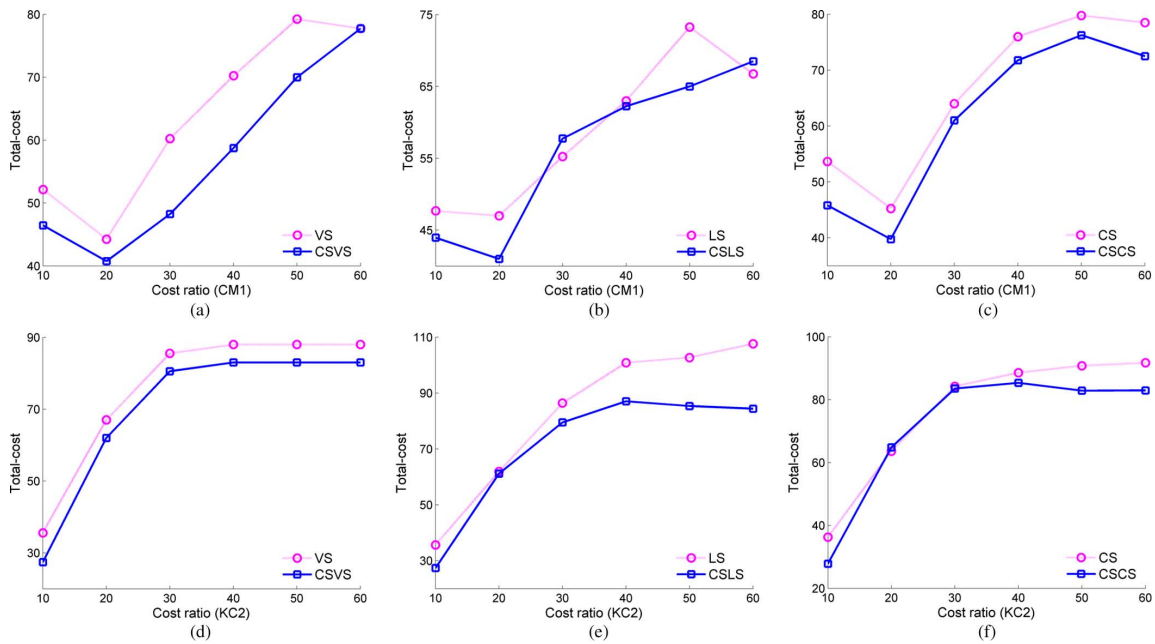

Fig. 5. Total-cost vs. different cost ratios for different methods on CM1 and KC2 data sets.

As can be seen from Fig. 3, compared to NN, both the cost-sensitive and cost-blind feature selection methods can significantly reduce the *Total-cost* and improve the $Sensitivity$ on two data sets, demonstrating that feature selection is an important step for the SDP problem. On the other hand, one can see that the proposed CSVS + NN, CSLS + NN, and CSCS + NN methods usually outperform VS+NN, LS+NN, and CS+NN, respectively.

The experimental results achieved by different feature selection methods using a cost-sensitive classifier (i.e., CSNN) are given in Fig. 4. From Fig. 4, one can find a similar trend as in Fig. 3. Specifically, feature selection can help improve the performance of SDP, compared to CSNN without any feature selection stage. Also, the proposed cost-sensitive feature selection methods usually perform better than cost-blind feature selection methods in *Total-cost* and $Sensitivity$ on the two data sets we used. These results further validate the efficacy of using cost information in both the feature selection and the classification stages.

F. Discussion

1) Influence of Cost Ratio: In the above experiments, the cost ratios (i.e., $C_{\mathrm{OI}}/C_{\mathrm{IO}}$) are given by users, which reflect the users' intention on the trade-off between different types of errors. Now, we investigate the influence of different cost ratios on the performances of different methods. To be specific, the proposed TSCS (i.e., CSVS + CSNN, CSLS + CSNN and CSCS + CSNN) methods are compared to conventional single-stage cost-sensitive learning methods (i.e., VS + CSNN, LS + CSNN, and CS+CSNN) by using different cost ratios. We first set $C_{\mathrm{IO}} = 2$, and then select the cost ratio from {5, 10, 15, 20, 25, 30}. In Fig. 5, we plot the *Total-cost* achieved by different methods with different cost ratios.

As can be seen from Fig. 5, the *Total-cost* achieved by our proposed cost-sensitive feature selection algorithms (including CSVS, CSLS, and CSCS) rises slowly with the increase of the cost ratios on both CM1 and KC2 data sets. In most cases, the proposed TSCS perform better than single-stage cost-sensitive methods in reducing the overall cost of misclassification.

2) Threats to Validity: The proposed TSCS learning method is a general method, where any other kinds of cost-sensitive feature selection algorithms, and cost-sensitive classifiers can be used in the feature selection stage, and the classification stage, respectively. In the current study, we adopt CSNN as the cost-sensitive classifier, because it has been recently shown effective for addressing the class-imbalance problem in SDP [17]. In addition, to the best of our knowledge, no previous studies have addressed using cost information in the feature selection stage. Accordingly, we proposed three filter-type cost-sensitive feature selection methods, i.e., CSVS, CSLS, and CSCS, which are more suitable for large-scale software systems than other (e.g., wrapper-type) feature selection methods.

In the current experiments, we used public software defect prediction data sets from NASA projects. However, due to different pre-processing methods, there might be different results between our current study and others. In addition, as the NASA MDP data sets have been questioned by Gray *et al.* [83] recently, other software defect data could be used to further validate our proposed methods.

V. Conclusion

To address the class-imbalance and high-dimensional data problems of software defect prediction, we propose a two-stage cost-sensitive learning (TSCS) method, where the cost information is utilized not only in the classification stage but also in the feature selection stage. We also develop three cost-sensitive feature selection methods, called CSVS, CSLS, and CSCS, by incorporating the cost information into conventional feature selection algorithms. Experimental results demonstrate that the proposed TSCS methods outperform single-stage cost-sensitive learning methods, while the proposed cost-sensitive feature selection methods perform better than conventional cost-blind feature selection methods.

Acknowledgment

The authors would like to thank the editor and all the referees for their useful comments and contributions for the improvement of this paper. Daoqiang Zhang is the corresponding author for this paper.

References

[1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Trans. Software Eng.*, vol. 37, pp. 356–370, 2011.

[2] B. T. Compton and C. Withrow, "Prediction and control of ADA software defects," *J. Syst. Software*, vol. 12, pp. 199–207, 1990.

[3] J. Munson and T. M. Khoshgoftaar, "Regression modelling of software quality: Empirical investigation," *J. Electro. Mater.*, vol. 19, pp. 106–114, 1990.

[4] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Data mining for predictors of software quality," *Int. J. Softw. Eng. Know.*, vol. 9, pp. 547–563, 1999.

[5] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, "Assessing predictors of software defects," in *Proc. Workshop on Predictive Software Models*, Chicago, IL, USA, 2004.

[6] C. Wohlin and P. Runeson, "Defect content estimations from review data," in *Proc. 20th Int. Conf. Software Eng.*, Washington, DC, USA, 1998, pp. 400–409.

[7] S. V. Wiel and L. Votta, "Assessing software designs using capture-recapture methods," *IEEE Trans. Software Eng.*, vol. 19, pp. 1045–1054, 1993.

[8] P. Runeson and C. Wohlin, "An experimental evaluation of an experience-based capture-recapture method in software code inspections," *Empir. Softw. Eng.*, vol. 3, pp. 381–406, 1998.

[9] L. C. Briand, K. E. Emam, and B. G. Freimut, "A comparison and integration of capture-recapture models and the detection profile method," in *Proc. 9th Int. Symp. Softw. Reliab. Eng.*, Paderborn, 1998.

[10] L. C. Briand, K. E. Emam, B. G. Freimut, and O. Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content," *IEEE Trans. Software Eng.*, vol. 26, pp. 518–540, 2000.

[11] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *IEEE Trans. Software Eng.*, vol. 32, pp. 69–82, 2006.

[12] C.-P. Chang, C.-P. Chu, and Y.-F. Yeh, "Integrating in-process software defect prediction with association mining to discover defect pattern," *Inf. Software Tech.*, vol. 51, pp. 375–384, 2009.

[13] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, pp. 1806–1817, 2012.

[14] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. I. Hudepohl, "Classification-tree models of software-quality over multiple releases," *IEEE Trans. Rel.*, vol. 49, pp. 4–11, 2000.

[15] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning, "Application of neural networks for predicting defects," *Annal. Software Eng.*, vol. 1, pp. 141–154, 1995.

[16] T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud, "Application of neural networks to software quality modeling of software quality," *Int. J. Rel. Qual. Safety Eng.*, vol. 6, pp. 902–909, 1997.

[17] J. Zheng, "Cost-sensitive boosting neural network for software defect prediction," *Expert Syst. Appl.*, vol. 37, pp. 4537–4543, 2010.

[18] T. M. Khoshgoftaar and N. Seliya, "Analogy-based practical classification rules for software quality estimation," *Empir. Softw. Eng.*, vol. 8, pp. 325–350, 2003.

[19] M. Chapman, P. Callis, and W. Jackson, 2004, Metrics Data Program [Online]. Available: http://mdp.ivv.nasa.gov

[20] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, pp. 476–493, 1994.

[21] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*. : Prentice Hall, 1994.

[22] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE Trans. Rel.*, vol. 61, pp. 516–525, 2012.

[23] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Software Pract. Exper.*, vol. 41, pp. 579–606, 2011.

[24] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, "An empirical study of feature ranking techniques for software quality prediction," *Int. J. Softw. Eng. Know.*, vol. 22, pp. 161–183, 2012.

[25] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[26] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artif. Intell.*, vol. 97, pp. 245–271, 1997.

[27] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proc. 8th IEEE Int. Conf. Inform. Reuse Integration*, Las Vegas, NV, USA, 2007, pp. 667–672.

[28] T. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Inf. Syst. Front.*, pp. 1–22, 2013.

[29] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Trans. Software Eng.*, vol. 14, pp. 1462–1477, 1988.

[30] S. Dick and A. Kandel, "Data mining with resampling in software metrics databases," in *Artificial Intelligence Methods in Software Testing*, M. Last, A. Kandel, and H. Bunke, Eds. Singapore: World Scientific, 2004, pp. 175–208.

[31] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Proc. North Amer. Fuzzy Inf. Process. Soc.*, San Diego, CA, 2007, pp. 69–72.

[32] T. M. Khoshgoftaar and J. V. Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, pp. 1283–1294, 2009.

[33] C. X. Ling, S. Sheng, T. Bruckhaus, and N. H. Madhavji, "Predicting software escalations with maximum ROI," in *Proc. IEEE Int. Conf. Data Mining*, Houston, TX, USA, 2005, pp. 717–720.

[34] T. M. Khoshgoftaar, E. Geleyn, L. Nguyen, and L. Bullard, "Cost-sensitive boosting in software quality modeling," in *Proc. 7th IEEE Int. Symp. High Assurance Syst. Eng.*, Tokyo, Japan, 2002, pp. 51–60.

[35] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proc. 27th Int. Conf. Software Eng.*, St. Louis, MO, USA, 2005, pp. 580–586.

[36] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Software Eng.*, vol. 22, pp. 751–761, 1996.

[37] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high-risk software components," *J. Syst. Software*, vol. 55, pp. 301–320, 2001.

[38] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. 30th Int. Conf. Software Eng.*, Leipzig, Germany, 2008, pp. 181–190.

[39] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in *9th Int. Workshop on Principles of Software Evolution*, Dubrovnik, Croatia, 2007, pp. 11–18.

[40] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. 27th Int. Conf. Software Eng.*, St. Louis, MO, USA, 2005, pp. 284–292.

[41] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *Proc. 21st IEEE Int. Conf. Software Maintenance*, 2005, pp. 263–272.

[42] S. Kim, Z. T. J. Whitehead, and A. Zeller, "Predicting faults from cached history," in *Proc. 29th Int. Conf. Software Eng.*, Washington, DC, USA, 2007, pp. 489–498.

[43] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Trans. Rel.*, vol. 56, pp. 223–236, 2007.

[44] K. Ganesan, T. M. Khoshgoftaar, and E. B. Allen, "Case-based software quality prediction," *Int. J. Softw. Eng. Know.*, vol. 10, pp. 139–152, 2000.

[45] S. S. Gokhale and M. R. Lyu, "Regression tree modeling for the prediction of software quality," in *Proc. 3rd Int. Conf. Reliab. and Quality in Design*, CA, USA, 1997, pp. 31–36.

[46] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharya, and G. D. Richardson, "Predictive modeling techniques of software quality from software measures," *IEEE Trans. Software Eng.*, vol. 18, pp. 979–987, 1992.

[47] T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya, "A comparative study of pattern recognition techniques for quality evaluation of telecommunications software," *IEEE J. Sele. Area. Comm.*, vol. 12, pp. 279–291, 1994.

[48] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of defect-proneness by random forests," in *Proc. 15th Int. Symp. Software Rel. Eng.*, 2004.

[49] P. D. Turney, "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm," *J. Artif. Intell. Res.*, vol. 2, pp. 369–409, 1995.

[50] T. M. Khoshgoftaar, M. P. Evett, E. B. Allen, and P. D. Chien, "An application of genetic programming to software quality prediction," in *Comput. Intell. Software Eng.*, River Edge, NJ, USA, 1998, pp. 176–195.

[51] T. M. Khoshgoftaar and Y. Liu, "A multi-objective software quality classification model using genetic programming," *IEEE Trans. Rel.*, vol. 56, pp. 237–245, 2007.

[52] S. Dick and A. Sadia, "Fuzzy clustering of open-source software quality data: A case study of mozilla," in *Proc. Int. Joint. Conf. Neural Networks*, Vancouver, BC, Canada, 2006, pp. 4089–4096.

[53] N. Seliya and T. M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semisupervised clustering," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 37, pp. 201–211, 2007.

[54] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Software*, vol. 81, pp. 649–660, 2008.

[55] Y. Bo and L. Xiang, "A study on software reliability prediction based on support vector machines," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manag.*, Singapore, 2007, pp. 1176–1180.

[56] 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*. Berlin, Germany: Springer-Verlag, 2009, vol. 43, pp. 223–234.

[57] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empir. Softw. Eng.*, vol. 17, pp. 62–74, 2012.

[58] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, pp. 429–449, 2002.

[59] J. Lu and Y. P. Tan, "Cost-sensitive subspace learning for face recognition," *IEEE Trans. Inf. Foren. Sec*, vol. 8, pp. 510–519, 2013.

[60] Y. Zhang and Z. Zhou, "Cost-sensitive face recognition," *IEEE Trans. Pattern Anal.*, vol. 32, pp. 1758–1769, 2010.

[61] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Min. Knowl. Disc.*, vol. 1, pp. 291–316, 1997.

[62] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special issue on learning from imbalanced data sets," *SIGKDD Explorations Newslett.*, vol. 6, pp. 1–6, 2004.

[63] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, pp. 63–77, 2006.

[64] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," in *Proc. 5th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, San Diego, California, USA, 1999, pp. 155–164.

[65] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. 17th Int. Joint Conf. Artif. Intell.*, Seattle, WA, USA, 2001, pp. 973–978.

[66] Z.-H. Zhou and X.-Y. Liu, "On multi-class cost-sensitive learning," in *Proc. 21st National Conf. Artificial Intelligence*, 2006, pp. 567–572.

[67] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *Proc.10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Seattle, WA, USA, 2004, pp. 3–11.

[68] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok, "Toward cost-sensitive modeling for intrusion detection and response," *J. Computer Security*, vol. 10, pp. 5–22, 2002.

[69] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recogn.*, vol. 40, pp. 3358–3378, 2007.

[70] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Proc. 3th IEEE Int. Conf. Data Mining*, 2003, pp. 435–442.

[71] S. Viaene and G. Dedene, "Cost-sensitive learning and decision making revisited," *Eur. J. Operational Res.*, vol. 166, pp. 212–220, 2005.

[72] W. Fan and S. J. Stolfo, "AdaCost: Misclassification cost-sensitive boosting," in *Proc. 16th Int. Conf. Machine Learning*, 1999, pp. 97–105.

[73] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "A comparative study of data sampling and cost sensitive learning," in *Proc. IEEE Int. Conf. Data Mining Workshops*, Washington, DC, USA, 2008, pp. 46–52.

[74] A. R. Webb*, Statistical Pattern Recognition*. : Wiley, 2002.

[75] I. Guyon, S. Gunn, M. Nikravesh, and Z. L. *, Feature Extraction: Foundations and Applications*. Berlin, Germany: Springer-Verlag New York, Inc., 2006.

[76] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, pp. 273–324, 1997.

[77] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: Criteria of max-dependency, max-relevance and min-redundancy," *IEEE Trans. Pattern Anal.*, vol. 27, pp. 1226–1238, 2005.

[78] D. Zhang, S. Chen, and Z. Zhou, "Constraint Score: A new filter method for feature selection with pairwise constraints," *Pattern Recogn.*, vol. 41, pp. 1440–1451, 2008.

[79] D. Sun and D. Zhang, "Bagging constraint score for feature selection with pairwise constraints," *Pattern Recogn.*, vol. 43, pp. 2106–2118, 2010.

[80] C. M. Bishop*, Neural Networks for Pattern Recognition*. London, U.K.: Oxford University Press, 1995.

[81] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[82] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 32, pp. 2–13, 2007.

[83] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the NASA metrics data program data sets for automated software defect prediction," in *Proc. 15th Annual Conf. Evaluation and Assessment in Software Engineering*, Durham, 2011, pp. 96–103.

Mingxia Liu received the B.S. degree, and M.S. degree from Shandong Normal University, China, in 2003, and 2006, respectively. She joined the School of Information Science and Technology in Taishan University as a Lecturer in 2006. She is currently a Ph.D. candidate in Computer Science from Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China. Her research interests include machine learning, pattern recognition, and software engineering.

Linsong Miao received the B.S. degree from Nanjing University of Information Science and Technology in 2009, and M.S. degree from NUAA, China, in 2012. His research interests include software engineering and machine learning.

Daoqiang Zhang received the B.S. degree, and Ph.D. degree in Computer Science from Nanjing University of Aeronautics and Astronautics (NUAA), China, in 1999, and 2004, respectively. He joined the Department of Computer Science and Engineering of NUAA as a Lecturer in 2004, and is a professor at present. His research interests include machine learning, pattern recognition, data mining, and medical image analysis. In these areas, he has published over 100 scientific articles in refereed international journals such as Neuroimage, Pattern Recognition, Artificial Intelligence in Medicine, IEEE Trans. Neural Networks; and conference proceedings such as IJCAI, AAAI, SDM, ICDM. He was nominated for the National Excellent Doctoral Dissertation Award of China in 2006, won the best paper award at the 9th Pacific Rim International Conference on Artificial Intelligence (PRICAI'06), and was the winner of the best paper award honorable mention of Pattern Recognition Journal 2007. He has served as a program committee member for several international and native conferences such as IJCAI, SDM, CIKM, PAKDD, PRICAI, and ACML, etc. He is a member of the Machine Learning Society of the Chinese Association of Artificial Intelligence (CAAI), and the Artificial Intelligence & Pattern Recognition Society of the China Computer Federation (CCF).