Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 785435, 8 pages
http://dx.doi.org/10.1155/2014/785435

Research Article

On Software Defect Prediction Using Machine Learning

Jinsheng Ren,[1] Ke Qin,[1] Ying Ma,[2] and Guangchun Luo[1]

[1] *University of Electronic Science and Technology of China, Chengdu 611731, China*
[2] *Xiamen University of Technology, Xiamen 361024, China*

Correspondence should be addressed to Ke Qin; qinke@uestc.edu.cn

Received 19 October 2013; Revised 2 January 2014; Accepted 16 January 2014; Published 23 February 2014

Academic Editor: Chin-Yu Huang

This paper mainly deals with how kernel method can be used for software defect prediction, since the class imbalance can greatly reduce the performance of defect prediction. In this paper, two classifiers, namely, the asymmetric kernel partial least squares classifier (AKPLSC) and asymmetric kernel principal component analysis classifier (AKPCAC), are proposed for solving the class imbalance problem. This is achieved by applying kernel function to the asymmetric partial least squares classifier and asymmetric principal component analysis classifier, respectively. The kernel function used for the two classifiers is Gaussian function. Experiments conducted on NASA and SOFTLAB data sets using F-measure, Friedman's test, and Tukey's test confirm the validity of our methods.

1. Introduction

Software defect prediction is an essential part of software quality analysis and has been extensively studied in the domain of software-reliability engineering [1–5]. However, as pointed out by Menzies et al. [2] and Seiffert et al. [4], the performance of defect predictors can be greatly degraded by class imbalance problem of the real-world data sets. Here the "class imbalanced" means that the majority of defects in a software system are located in a small percentage of the program modules. Current approaches to solve the class imbalance problem can be roughly categorized into two ways: in a data-level way or algorithm-level way, as reported in [4]. The literature [4] shows that the algorithm-level method AdaBoost almost always outperforms even the best data-level methods in software defect prediction. AdaBoost is a typical adaptive algorithm which has received great attention since Freund and Schapire's proposal [6]. Adaboost attempts to reduce the bias generated by majority class data, by updating the weights of instances dynamically according to the errors in previous learning. Some other studies improved dimension reduction methods for the class imbalanced problem by means of partial least squares (PLS) [7], linear discriminant analysis (LDA) [8], and principle component analysis (PCA) [9, 10]. Although PLS was not inherently designed for problems of classification and discrimination, it is widely used in many areas that need class proclaimation. The authors of [7] reported that rarely will PLS be followed by an actual discriminant analysis on the scores and rarely is the classification rule given a formal interpretation. Still this method often produces nice separation. Based on the previous work, recently, Qu et al. investigated the effect of PLS in unbalanced pattern classification. It is reported that, beyond dimension reduction, PLS is proved to be superior to generate favorable features for classification. Thereafter, they proposed an asymmetric partial least squares (APLS) classifier to deal with the class imbalance problem. They illustrated that APLS outperforms other algorithms because it can extract favorable features for unbalanced classification. As for the PCA, it is an effective linear transformation, which maps high-dimensional data to a lower dimensional space. Based on the PCA, the authors of [11] proposed kernel principal component analysis (KPCA) which can perform nonlinear mapping $\Phi(x)$ to transform an input vector to a higher dimensional feature space, where kernel function $\Phi(x)$ is introduced to reduce computation for mapping the data nonlinearly into a feature space. Then linear PCA is used in this feature space.

While both APLS and KPCA are of great value, they have their own disadvantages. For example, the APLS classifier is a bilinear classifier, in which the dimension is mapped to a bilinear subspace, which is, to some degree, obscure and not

easy to implement. The KPCA regression model does not consider the correlation between principal components and the class attribution. PCA dimension reduction is affected inevitably by asymmetric distribution. In this paper, we propose two kernel-based learning methods to solve the class imbalance problem, called asymmetric kernel partial least squares classifier (AKPLSC) and asymmetric kernel principal component analysis classifier (AKPCAC), respectively. The former is able to nonlinearly extract the favorable features and retrieve the loss caused by class imbalance problem, while the latter is more adaptive to imbalance data sets.

It is not out of place to explain the relationship between this paper and our previous papers [12, 13]. The AKPLSC and AKPCAC were firstly proposed in [12, 13], respectively. However, recently, we found some errors when we proceeded to our work. And due to these errors, the AKPCAC and AKPLSC proposed in [12, 13] show superiority only in part of the data sets. We carefully rectified the source code and then tested the AKPCAC and AKPLSC again on the whole data sets by means of statistical tools, such as Friedman's test and Tukey's test. The outcomes show that our classifiers indeed outperform the others, namely, APLSC, KPCAC, AdaBoost, and SMOTE. We carefully examine the theory and experimental results and then form this paper in more detail.

2. State of the Art

In software defect prediction, $L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)\} \subset X \times Y$ denotes the labeled example set with size ℓ and $U = \{x_{\ell+1}, x_{\ell+2}, \ldots, x_{\ell+u}\} \subset X$ denotes the unlabeled example set with size u. For labeled examples, $Y = \{+1, -1\}$, the defective modules are labeled "+1" and the nondefective modules are labeled "−1". Software defect data sets are highly imbalanced; that is, the examples of the minority class (defective modules) are heavily underrepresented in comparison to the examples of majority class (nondefective modules). Thereby, lots of algorithms are proposed to cope with this problem, as will be seen below.

2.1. Software Defect Predictor Related to Partial Least Squares. Linear partial least squares (PLS) [7] is an effective linear transformation, which performs the regression on the subset of extracted latent variables. Kernel PLS [14] first performs nonlinear mapping, $\Phi : \{x_i\}_{i=1}^n \in \mathbb{R}^N \rightarrow \Phi(x) \in \mathbb{F}$, to project an input vector to a higher dimensional feature space, in which the linear PLS is used.

Given the center M, the radius of the class region r, and the parameter of overlapping η, the relationship of the two classes can be expressed as $M_{+1} - M_{-1} = \eta(r_{+1} - r_{-1})$. The parameter η indicates the level of overlapping between the region of the two classes (the smaller the value of η is, the higher the overlapping will be).

APLSC can be expressed as $\widehat{Y} = \text{sign}(\sum_{i=1}^k m_i t_i - b)$, which is derived from the regression model of the linear PLS, $\widehat{y} = \sum_{i=1}^k m_i t_i$, where k is the number of the latent variables, t_i is the ith score vector of testing data, m_i indicates the direction of ith score, and the bias b is equal to $m_1(M_{+1} - r_{+1}\eta)$.

APLSC suffers from the high overlapping, especially when the data sets are nonlinear separable [15]. A suggestion of solving such overlapping problem is by using a kernel method. Kernel PLS [14] corresponds to solving the eigenvalue equation as follows:

$$\Phi\Phi^T\Psi\Psi^T\tau = \lambda\tau, \tag{1}$$

where Φ and Ψ denote the matrix of mapped X-space data $\Phi(x)$ and the matrix of mapped Y-space data $\Psi(y)$ in the feature space $\mathbb{F}$, respectively. The nonlinear feature selection methods can reduce the overlapping level of the two classes, but the class imbalance problem makes them fail to distinguish the minority class [15]. In order to retrieve the loss caused by class imbalance problem, we want to get the bias $\widehat{b}$ of the kernel PLS classification, KPLSC [14].

Different from the APLSC, the kernel PLS regression is $\widehat{y} = \sum_{i=1}^\ell \alpha_i \kappa(x_i, x)$, where ℓ is the size of labeled example set, $\kappa(x_i, x)$ is a kernel function, and α_i is dual regression coefficient. Consequently, we may combine the APLSC and kernel PLS so that we get the asymmetric kernel PLS, as will be seen in Section 3.1.

2.2. Kernel Principal Component Analysis Classifier for Software Defect Prediction. Principal Component Analysis (PCA) [10] is an effective linear transformation, which maps high-dimensional data to a lower dimensional space. Kernel principal component analysis (KPCA) [11] first performs nonlinear mapping $\Phi(x)$ to transform an input vector to a higher dimensional feature space. And then linear PCA is used in this feature space.

For both of the algorithms demonstrated in [10, 11], the input data are centralized in the original space and the transformed high-dimensional space; that is, $\sum_{i=1}^\ell x_i = 0$ and $\sum_{i=1}^\ell \Phi(x_i) = 0$, where ℓ is the number of the labeled data and x_i is the ith instance of the data set. In the proceeding of PCA, the correlation matrix $C = (1/\ell)\sum_{i=1}^\ell x_i x_i'$ should be diagonalized, while, in KPCA, the correlation matrix $C^\Phi = (1/\ell)$ $\sum_{i=1}^\ell \Phi(x_i)\Phi(x_i)'$ should be diagonalized. It is equal to solving the eigenvalue problem $\lambda V = C^\Phi V$, where λ is an eigenvalue and V is a matrix of eigenvectors in KPCA. It can also be written as $n\lambda\alpha = K\alpha$, where $K = nC^\Phi$ is the kernel matrix.

The kernel principal component regression algorithm has been proposed by Rosipal et al. [11]. The standard regression model in the transformed feature space can be written as

$$f(x) = \sum_{k=1}^p w_k \beta(x)_k + b, \tag{2}$$

where p is the number of components, w_k is the kth primal regression coefficient, and b is the regression bias. Consider $\beta(x)_k = V_k\Phi(x)$, where V_k is the kth eigenvector of V. V and Λ are the eigenvectors and eigenvalues of the correlation matrix, respectively.

2.3. Data Set. There are many data sets for machine learning test, such as the UCI [16] and the PROMISE data repository [17] (since the contributors maintain these data sets continuously, the metrics listed in Table 1 may vary at different times). What we are using in this paper are the latest

Table 1: Data sets.

Project	Modules	Attributes	Size (loc)	%Defective	Description
ar3	63	30	5,624	11.11	Embedded controller
ar4	107	30	9,196	18.69	Embedded controller
ar5	36	30	2,732	22.22	Embedded controller
cm1	327	38	14,763	12.84	Spacecraft instrument
kc1	2109	21	42,965	15.46	Storage management
kc2	521	22	19,259	20.54	Storage management
kc3	194	40	7,749	18.56	Storage management
mw1	253	38	8341	10.67	A zero gravity experiment
pc1	705	38	25,924	8.65	Flight software

Table 2: Metrics used in our experiment.

Type	Number	Metric
Loc	5	Halstead's count of blank lines; McCabe's line count of code; Halstead's line count; Halstead's count of lines of comments; line count of code and comment
McCabe	3	Cyclomatic complexity; essential complexity; design complexity
Halstead	12	Unique operators; unique operands; total operators; total operands; total operators and operands; volume; program length; difficulty; intelligence; effort; volume on minimal implementation; time estimator
BranchCount	1	BranchCount
Others	18	Global data complexity; cyclomatic density; decision count; decision density; global data density; essential density; design density; loc executable; parameter count; percent comments; normalized cyclomatic complexity; modified condition count multiple condition count; node count; maintenance severity; condition count; global data complexity; call pairs; edge count

ones updated in June 2012. They are different from the data set that we used in our previous papers [12, 13]), which is a data collection from real-world software engineering projects. The choice that which data set should be used depends on the area of the machine learning where it will be applied. In this paper, the experimental data sets come from NASA and SOFTLAB, which can be obtained from PROMISE [17], as shown in Tables 1 and 2. These software modules are developed in different languages, at different sites by different teams, as shown in Table 1. The SOFTLAB data sets (ar3, ar4, and ar5) are drawn from three controller systems for a washing machine, a dishwasher, and a refrigerator, respectively. They are all written in C. The rests are from NASA projects. They are all written in C++, except for kc3, which is written in JAVA. All the metrics are computed according to [17].

3. Design the Asymmetric Classifiers Based on Kernel Method

3.1. The Asymmetric Kernel Partial Least Squares Classifier (AKPLSC). As we illustrated in Section 2.1, APLSC can be expressed as $\widehat{Y} = \text{sign}(\sum_{i=1}^{k} m_i t_i - b)$ and the kernel PLS regression is $\widehat{y} = \sum_{i=1}^{\ell} \alpha_i \kappa(x_i, x)$; thus the AKPLSC can be well characterized as

$$\widehat{Y} = \text{sign}\left(\sum_{i=1}^{\ell} \alpha_i \kappa\left(x_i, x\right) - \widehat{b}\right), \tag{3}$$

where α_i is dual regression coefficient, which can be obtained from kernel PLS, as shown in Algorithm 1 and $\widehat{b}$ is the bias of the classifier.

Since kernel PLS put most of the information on the first dimension, the bias in the AKPLSC can be computed similarly as [15]

$$\widehat{b} = c_1 * \left(M_{+1} - r_{+1}\eta\right) = c_1 * \frac{M_{+1} r_{-1} + M_{-1} r_{+1}}{r_{-1} + r_{+1}}, \tag{4}$$

where c_1 indicates the direction of the first score τ_1 and the centers (M_{+1}, M_{-1}) and radiuses (r_{+1}, r_{-1}) are computed based on τ_1, which can be obtained from (1). Then we move the origin to the center of mass by employing data centering, as reported in [14]:

$$K = K - \frac{1}{\ell} JJ'K - \frac{1}{\ell} KJJ' + \frac{1}{\ell^2}\left(J'KJ\right)JJ', \tag{5}$$

where J is the a vector with all elements that are equal to 1. After data centering, the AKPLSC can be described as shown in Algorithm 1.

3.2. The Asymmetric Kernel Principal Component Analysis Classifier (AKPCAC). The KPCA regression model does not consider the correlation between principal components and the class attribution. PCA dimension reduction is inevitably

Input: Labeled and unlabeled data sets, L and U; number of components, k.
Output: Asymmetric Kernel Partial Least Squares Classifier, H;
Method:
(1) $K_{ij} = \kappa\left(x_i, x_j\right), i, j = 1, \ldots, \ell, x_i, x_j \in L$;
(2) $K_1 = K, \widehat{Y} = Y$, where K is the kernel matrix, Y is the label vector.
(3) **for** $j = 1, \ldots, k$ **do**
(4) $\beta_j = \beta_j / \left\|\beta_j\right\|$, where β_j is a projection direction.
(5) **repeat**
(6) $\beta_j = \widehat{Y}\widehat{Y}' K_j \beta_j$
(7) $\beta_j = \beta_j / \left\|\beta_j\right\|$
(8) **untill** convergence
(9) $\tau_j = K_j \beta_j$, where τ_j is the score
(10) $c_j = \widehat{Y}' \tau_j / \left\|\tau_j\right\|^2$, where c_j is the direction of the score
(11) $\widehat{Y} = \widehat{Y} - \tau_j c_j'$, where $\widehat{Y}$ is the deflation of Y
(12) $K_{j+1} = (I - \tau_j \tau_j' / \left\|\tau_j\right\|^2) K_j (I - \tau_j \tau_j' / \left\|\tau_j\right\|^2)$
(13) **end for**
(14) $B = [\beta_1, \ldots, \beta_k], T = [\tau_1, \ldots, \tau_k]$
(15) $\alpha = B(T'KB)^{-1} T'Y$, where α is the vector of dual regression coefficients
(16) Calculate $\widehat{b}$ according to (4);
(17) $H(x) = \operatorname{sign}\left(\sum_{i=1}^{\ell} \alpha_i \kappa\left(x_i, x\right) - \widehat{b}\right), x \in U$;
(18) **return** H;
End Algorithm AKPLSC.

Algorithm 1: AKPLSC.

affected by asymmetric distribution [15]. We analyze the effect of class imbalance on KPCA. Considering the class imbalance problem, we propose an asymmetric kernel principal component analysis classifier (AKPCAC), which retrieves the loss caused by this effect.

Suppose that $S_b^\Phi = \sum_{i=1}^2 n_i (\overline{u_i} - \overline{u})(\overline{u_i} - \overline{u})'$ denotes the between-class scatter matrix and $S_w^\Phi = \sum_{i=1}^2 \sum_{j=1}^{n_i} (\Phi(x_j^i) - \overline{u_i})(\Phi(x_j^i) - \overline{u_i})'$ denotes the within-class scatter matrix, where $\overline{u_i} = (1/n_i) \sum_{j=1}^{n_i} \Phi(x_j^i)$ is class-conditional mean vector, $\overline{u}$ is mean vector of total instances, $\Phi(x_j^i)$ is the jth instances in the ith class, and n_i is the number of instances of the ith class. The total noncentralized scatter matrix in the form of kernel matrix is

$$\begin{aligned} K &= \sum_{i=1}^{2} \sum_{j=1}^{\ell} \left(\Phi\left(x_j^i\right) - \overline{u}\right)\left(\Phi\left(x_j^i\right) - \overline{u}\right)' \\ &= \sum_{i=1}^{2} \sum_{j=1}^{\ell} \left(\Phi\left(x_j^i\right) - \overline{u_i} + \overline{u_i} - \overline{u}\right)\left(\Phi\left(x_j^i\right) - \overline{u_i} + \overline{u_i} - \overline{u}\right)' \\ &= S_w^\Phi + S_b^\Phi + \sum_{i=1}^{2} \sum_{j=1}^{\ell} \left(\Phi\left(x_j^i\right) - \overline{u_i}\right)\left(\overline{u_i} - \overline{u}\right)' \\ &\quad + \sum_{i=1}^{2} \sum_{j=1}^{\ell} \left(\overline{u_i} - \overline{u}\right)\left(\Phi\left(x_j^i\right) - \overline{u_i}\right)'. \end{aligned} \tag{6}$$

The third term of (6) can be rewritten as

$$\begin{aligned} &\sum_{i=1}^{2} \sum_{j=1}^{\ell} \left(\Phi\left(x_j^i\right) - \overline{u_i}\right)\left(\overline{u_i} - \overline{u}\right)' \\ &\quad = \sum_{i=1}^{2} \left(\sum_{j=1}^{\ell} \left(\Phi\left(x_j^i\right) - \overline{u_i}\right)\right)\left(\overline{u_i} - \overline{u}\right)' \\ &\quad = \sum_{i=1}^{2} \left(\sum_{j=1}^{n_i} \Phi\left(x_j^i\right) - n_i \overline{u_i}\right)\left(\overline{u_i} - \overline{u}\right)'. \end{aligned} \tag{7}$$

Note that $n_i \overline{u_i} = \sum_{j=1}^{n_i} \Phi(x_j^i)$. Then the third term and fourth term of (6) are equal to zero. Thus, we have the relation $K = S_b^\Phi + S_w^\Phi = S_b^\Phi + P\Sigma_P^\Phi + N\Sigma_N^\Phi$, where P is the number of positive instances, N is the number of negative instances, Σ_P^Φ is the positive covariance matrices, and Σ_N^Φ is the negative covariance matrix. Since class distribution has a great impact on S_w^Φ, the class imbalance also impacts the diagonalization problem of KPCA.

In order to combat the class imbalance problem, we propose the AKPCAC, based on kernel method. It considers the correlation between principal components and the class distribution. The imbalance ratio can be denoted as $\sum_{i=1}^{\ell} I(y_i, +1) / \sum_{i=1}^{\ell} I(y_i, -1) = P/N$, which is the probabilities of the positive instances to the negative instances of training data, where $I(\cdot)$ is an indicator function: $I(x, y) = 1$ if $x = y$, zero otherwise. We assume that future test examples are drawn from the same distribution, so the imbalance ratio

Input: The set of Labeled samples, $L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)\}$;
The set of unlabeled samples, $U = \{x_{\ell+1}, x_{\ell+2}, \ldots, x_{\ell+u}\}$;
Output: Kernel Principal Component Classifier, H;
Method:
(1) $K_{ij} = \kappa(x_i, x_j), i, j = 1, \ldots, \ell$;
(2) $K = K - (1/\ell)JJ'K - 1/\ell KJJ' + 1/\ell^2(J'KJ)JJ'$, where J is a vector with all elements equal to 1.
(3) $[V, \Lambda] = \text{eig}(K)$;
(4) $\alpha = \sum_{j=1}^{p}(1/\lambda_j)(V_j'Y^s)V_j$; where $Y^s = \{y_1, y_2, \ldots, y_\ell\}$ is the label vector
(5) Calculate $\widehat{b}$, $H(x)$ according to (9), (10);
(6) **return** H;
End Algorithm APPCC.

Algorithm 2: AKPCAC.

of the training data is the same as that of the test data. Then, we have

$$\frac{\sum_{i=1}^{\ell}\left(\widehat{y}_i - \widehat{b}\right) I\left(y_i, +1\right)}{\sum_{i=1}^{\ell}\left(\widehat{y}_i - \widehat{b}\right) I\left(y_i, -1\right)} = \frac{\sum_{i=1}^{\ell} I\left(y_i, +1\right)}{\sum_{i=1}^{\ell} I\left(y_i, -1\right)} = \frac{P}{N}, \tag{8}$$

where $\widehat{b}$ is the bias of the classifier and $\widehat{y}_i$ is the regression result of x_i. $\widehat{y}_i$ can be computed by regression model equation (2). Note that the regression is conducted on the p principal components. Solving this one variable equation, we get

$$\widehat{b} = \frac{N\left(\sum_{i=1}^{\ell}\left(\widehat{y}_i I\left(y_i, +1\right)\right)\right) - P\left(\sum_{i=1}^{\ell}\left(\widehat{y}_i I\left(y_i, -1\right)\right)\right)}{N^2 - P^2}. \tag{9}$$

Based on principal components, (9) describes the detail deviation of the classifier. This deviation may be caused by class imbalance, noise, or other unintended factors. In order to retrieve the harmful effect, we compensate this deviation. By transforming the regression model (2), the classifier model can be written as

$$\begin{aligned} H(x) &= \text{sign}\left(\sum_{k=1}^{p} w_k \beta(x)_k + \widehat{b}\right) \\ &= \text{sign}\left(\sum_{k=1}^{p} w_k \sum_{i=1}^{\ell} \alpha_i^k \kappa\left(x_i, x\right) + \widehat{b}\right) \\ &= \text{sign}\left(\sum_{i=1}^{\ell} c_i \kappa\left(x_i, x\right) + \widehat{b}\right), \end{aligned} \tag{10}$$

where $\{c_i = \sum_{k=1}^{p} w_k \alpha_i^k\}, i = 1, 2, \ldots, \ell$.

AKPCAC is summarized in Algorithm 2. Since the AKPCAC was firstly studied for reducing the effect of class imbalance for classification, it inherently has the advantage of kernel method, which can conduct quite general dimensional feature space mappings. In this paper, again, we have illustrated how the unreliable dimensions based on KPCA can be removed; thereafter, the imbalance problem based on the PCA has also been solved.

4. Experimental Result

The experiments are conducted under the data set from NASA and SOFTLAB. The Gaussian kernel function $K(x, y) = \exp(-\|x - y\|^2)$ is adopted for the performance investigation for both AKPLSC and AKPCAC. The efficiency is evaluated by F-measure and Friedman's test, as will be explained presently.

4.1. Validation Method Using F-Measure. F-measure method is widely used for assessing a test's accuracy. It considers both the precision P and the recall R to compute the score. P is defined as the number of correct results divided by the number of all returned results. R is the number of correct results divided by the number of results that should have been returned. For the clarity of this paper, we give a short explanation of the F-measure as below. Obviously, there are four possible outcomes of a predictor:

(1) TP: true positives are modules classified correctly as defective modules;

(2) FP: false positives refer to nondefective modules incorrectly labeled as defective;

(3) TN: true negatives correspond to correctly classified nondefective modules;

(4) FN: false negatives are defective modules incorrectly classified as nondefective.

Thereby, the precision is defined as $P = \text{TP}/(\text{TP} + \text{FP})$ and the recall is $R = \text{TP}/(\text{TP} + \text{FN})$.

The general formula of the F-measure is

$$F_\beta = \frac{\left(1 + \beta^2\right) PR}{\beta^2 P + R}, \tag{11}$$

where β is a positive real number. According to the definition of P and R, (11) can be rewritten as

$$F_\beta = \frac{\left(1 + \beta^2\right) \text{TP}}{\left(1 + \beta^2\right) \text{TP} + \beta^2 \text{FN} + \text{FP}}. \tag{12}$$

Generally, there are 3 commonly used F-measures: F_1 (which is a balance of P and R), $F_{0.5}$ (which puts more

Table 3: Statistical F-measure (mean ± std) values of six classifiers on all data sets. The value of the winner method of each row is emphasized in bold.

	APLSC	KPCAC	AdaBoost	SMOTE	AKPCAC	AKPLSC
ar3	0.498 (0.062)	0.394 (0.115)	0.357 (0.026)	0.415 (0.003)	**0.569 (0.052)**	0.500 (0.009)
ar4	0.421 (0.023)	0.422 (0.058)	0.412 (0.013)	0.457 (0.000)	**0.474 (0.024)**	0.421 (0.001)
ar5	0.526 (0.058)	0.562 (0.055)	0.435 (0.012)	0.555 (0.001)	**0.625 (0.025)**	0.553 (0.022)
cm1	**0.254 (0.033)**	0.069 (0.031)	0.234 (0.007)	0.237 (0.000)	0.089 (0.031)	0.235 (0.002)
kc1	0.408 (0.005)	0.309 (0.006)	0.325 (0.005)	0.428 (0.000)	**0.462 (0.009)**	0.433 (0.235)
kc2	0.431 (0.021)	0.433 (0.014)	**0.534 (0.002)**	0.520 (0.000)	0.523 (0.011)	0.480 (0.036)
kc3	0.382 (0.036)	0.231 (0.025)	0.386 (0.002)	0.340 (0.000)	0.234 (0.045)	**0.412 (0.000)**
mw1	0.285 (0.011)	0.276 (0.080)	0.224 (0.002)	0.290 (0.000)	**0.371 (0.037)**	0.268 (0.025)
pc1	0.215 (0.019)	0.212 (0.025)	0.354 (0.001)	0.387 (0.000)	0.254 (0.015)	**0.398 (0.001)**

emphasis on P than R), and F_2 (which weights R higher than P). In this paper, $F_1 = 2PR/(P + R)$ is used to evaluate the efficiency of different classifiers. The F-measure can be interpreted as a weighted average of the precision and recall. It reaches its best value at 1 and worst score at 0.

We compare the F-measure values of different predictors including AKPLSC, AKPCAC, APLSC [15], KPCAC [11], AdaBoost [4], and SMOTE [18]. The results are listed in Table 3. For each data set, we perform a 10 × 5-fold cross validation.

From the table we may see clearly that the AKPLSC and the AKPCAC are superior than the other 4 classifiers, which validates our contributions of this paper.

4.2. Validation Method Using Friedman's Test and Tukey's Test. The Friedman test is a nonparametric statistical test developed by Friedman [19, 20]. It is used to detect differences in algorithms/classifiers across multiple test attempts. The procedure involves ranking each block (or row) together and then considering the values of ranks by columns. In this section, we present a multiple AUC value comparison among the six classifiers using Friedman's test.

At first, we make two hypotheses:

H_0: the six classifiers have equal classification probability;

H_1: at least two of them have different probability distribution.

In order to determine which hypothesis should be rejected, we compute the statistic:

$$F_r = \frac{12}{bk(k+1)} \sum_{i=1}^{k} R_i^2 - 3b(k+1), \tag{13}$$

where b is the number of blocks (or rows), k is the number of classifiers, and R_i is the summation of ranks of each column. The range of rejection for null hypothesis is $F_r > \chi_\alpha^2$. In our experiment, the degree of freedom is $k - 1 = 5$ and we set $\alpha = 0.05$; thus $F_r = 18.9683 > 11.0705$, which implies that H_0 should be rejected.

Friedman's test just tells us that at least two of the classifiers have different performance, but it does not give any implication which one performs best. In this case, a post hoc test should be proceeded. Actually, there are many post hoc tests such as LSD (Fisher's least significant difference), SNK (Student-Newman-Keuls), Bonferroni-Dunn's test, Tukey's test, and Nemenyi's test, which is very similar to the Tukey test for ANOVA. In this paper, the Tukey test [21] is applied.

Tukey's test is a single-step multiple comparison procedure and statistical test. It is used in conjunction with an ANOVA to find means that are significantly different from each other. It compares all possible pairs of means and is based on a studentized range distribution.

Tukey's test involves two basic assumptions:

(1) the observations being tested are independent;

(2) there is equal within-group variance across the groups associated with each mean in the test.

Obviously, our case satisfies the two requirements.

The steps of the Tukey multiple comparison with equal sample size can be summarized in Algorithm 3.

In this paper, we set $\alpha = 0.05$. Since we compare 6 classifiers over 9 data sets, then $n = 54$, $p = 6$, $v = n - p = 48$, and $n_t = 9$. $q_\alpha(p, v) \approx 4.2$, which can be found from the studentized range statistic table. Now the only problem to find the value of ω is to determine s and MSE. This can be calculated accordingly as

$$\begin{gathered}
\mathrm{SY} = \sum_{i=1}^{n} y_i, \qquad \mathrm{MY} = \frac{\mathrm{SY}}{n}, \qquad \mathrm{YS} = \sum_{i=1}^{n} y_i^2, \\
\mathrm{CM} = \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} = \frac{\mathrm{SY}^2}{n}, \\
\mathrm{SS} = \sum_{i=1}^{n} y_i^2 - \mathrm{CM} = \mathrm{YS} - \mathrm{CM}, \qquad \mathrm{SST} = \sum_{i=1}^{p} \frac{T_i^2}{n_i} - \mathrm{CM}, \\
\mathrm{SSE} = \mathrm{SS} - \mathrm{SST}, \qquad \mathrm{MSE} = \frac{\mathrm{SSE}}{n - p},
\end{gathered} \tag{14}$$

where y_i is the corresponding AUC value in Table 4 and T_i is the AUC summation of each column. Now we have the results: MSE = 0.0051, $s = 0.0713$, and $\omega = 0.0998$. The

Input: α, p, v, s, MSE, n_t and samples. The meaning of these parameters is: α is an error rate. p is the number of means, $s = \sqrt{\text{MSE}}$, v is the degree of freedom related with MSE. n_t is the number of observations of each sample.
Output: The minimum significant difference ω and a deduction;
Method:
(1) Choose a proper error rate α;
(2) Calculate the statistic ω according to

$$\omega = q_\alpha(p, v)\frac{s}{\sqrt{n_t}},$$

where $q_\alpha(p, v)$ is the critical value of Studentized range statistic, which can be found from any statistics textbooks.
(3) Compute and rank all the p means
(4) Draw a deduction based on the ranks in the confidence level $(1 - \alpha)$.
End Algorithm Tukey Multiple Comparison.

Algorithm 3: Tukey's multiple comparison: equal sample size.

Table 4: Comparison of AUC between six classifiers. The ranks in the parentheses are used in computation of the Friedman test.

	APLSC	KPCAC	AdaBoost	SMOTE	AKPCAC	AKPLSC
ar3	0.626 (3)	0.588 (5)	0.580 (6)	0.590 (4)	0.699 (1)	0.682 (2)
ar4	0.563 (5)	0.600 (3)	0.555 (6)	0.610 (2)	0.671 (1)	0.579 (4)
ar5	0.626 (5)	0.710 (2)	0.614 (6)	0.651 (3)	0.722 (1)	0.640 (4)
cm1	0.611 (4)	0.724 (1)	0.589 (5)	0.550 (6)	0.681 (2)	0.650 (3)
kc1	0.682 (4)	0.592 (6)	0.627 (5)	0.700 (3)	0.800 (1)	0.768 (2)
kc2	0.591 (6)	0.601 (5)	0.796 (1)	0.635 (3)	0.732 (2)	0.610 (4)
kc3	0.598 (5)	0.569 (6)	0.698 (2)	0.612 (4)	0.658 (3)	0.713 (1)
mw1	0.587 (5)	0.602 (4)	0.534 (6)	0.654 (2)	0.725 (1)	0.639 (3)
pc1	0.692 (6)	0.718 (5)	0.769 (3)	0.753 (4)	0.841 (2)	0.882 (1)
Sum of ranks	$T_1 = 43$	$T_2 = 37$	$T_3 = 40$	$T_4 = 31$	$T_5 = 14$	$T_6 = 24$

Table 5: Tukey's test result for the six classifiers.

	APLSC	KPCAC	SMOTE	AdaBoost	AKPLSC	AKPCAC
Rank	1	2	3	4	5	6
Mean: T_i	0.6196	0.6338	0.6394	0.6402	0.6848	0.7254
Difference: $T_{i+1} - T_i$	—	0.0142	0.0057	0.0008	0.0446	0.0407

means comparison is listed in Table 5. From this table we can see clearly the following.

(1) The difference $T_6 - T_1 = 0.1058$ is greater than the critical value ω, which hints that the AKPCAC is significantly better than the APLSC.

(2) But compared to the rest, except the APLSC, the two newly proposed methods have no significant difference.

(3) Nevertheless, the AKPCAC and AKPLSC have the largest and second largest means, which implies that both indeed outperform the rest, although insignificantly.

(4) Compared to the AKPLSC, the AKPCAC is slightly more powerful, which supports our claim that the AKPCAC is more adaptive to dimensional feature space mappings over imbalanced data sets.

(5) The deduction is made in the confidence level $(1 - 0.05)$.

5. Conclusion

In this paper, we introduce kernel-based asymmetric learning for software defect prediction. To eliminate the negative effect of class imbalance problem, we propose two algorithms called the asymmetric kernel partial least squares classifier and the asymmetric kernel principal component analysis classifier. The former one is derived from the regression model of linear PLS, while the latter is derived from kernel PCA method. The AKPLSC can extract feature information in a nonlinear way and retrieve the loss caused by class imbalance. The AKPCAC is more adaptive to dimensional feature space mappings over imbalanced data sets and has a better performance. F-measure, Friedman's test, and a post hoc test using Tukey's

method are used to verify the performance of our algorithms. Experimental results on NASA and SOFTLAB data sets validate their effectiveness.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work of this paper was supported by the National Natural Science Foundation of China (Grant no. 61300093) and the Fundamental Research Funds for the Central Universities in China (Grant no. ZYGX2013J071). The authors are extremely grateful to the anonymous referees of the initial version of this paper, for their valuable comments. The present version incorporates all the changes requested. Their comments, thus, significantly improved the quality of this present paper.

References

[1] T. M. Khoshgoftaar, E. B. Allen, and J. Deng, "Using regression trees to classify fault-prone software modules," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 455–462, 2002.

[2] 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.

[3] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[4] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics A*, vol. 39, no. 6, pp. 1283–1294, 2009.

[5] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE '04)*, pp. 417–428, November 2004.

[6] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the 13th International Conference on Machine Learning*, pp. 148–156, 1996.

[7] M. Barker and W. Rayens, "Partial least squares for discrimination," *Journal of Chemometrics*, vol. 17, no. 3, pp. 166–173, 2003.

[8] J.-H. Xue and D. M. Titterington, "Do unbalanced data have a negative effect on LDA?" *Pattern Recognition*, vol. 41, no. 5, pp. 1558–1571, 2008.

[9] X. Jiang, "Asymmetric principal component and discriminant analyses for pattern classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 931–937, 2009.

[10] I. T. Jolliffe, *Principal Component Analysis*, Springer, New York, NY, USA, 1986.

[11] R. Rosipal, M. Girolami, L. J. Trejo, and A. Cichocki, "Kernel PCA for feature extraction and de-noising in nonlinear regression," *Neural Computing and Applications*, vol. 10, no. 3, pp. 231–243, 2001.

[12] Y. Ma, G. Luo, and H. Chen, "Kernel based asymmetric learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. E-95-D, no. 1, pp. 267–270, 2012.

[13] Y. Ma, G. Luo, and H. Chen, "Kernel based asymmetric learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. E-95-D, no. 1, pp. 267–270, 2012.

[14] R. Rosipal, L. J. Trejo, and B. Matthews, "Kernel PLS-SVC for linear and nonlinear classification," in *Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pp. 640–647, August 2003.

[15] H.-N. Qu, G.-Z. Li, and W.-S. Xu, "An asymmetric classifier based on partial least squares," *Pattern Recognition*, vol. 43, no. 10, pp. 3448–3457, 2010.

[16] K. Bache and M. Lichman, "UCI Machine Learning Repository," University of California, School of Information and Computer Science, Irvine, Calif, USA, 2013 http://archive.ics.uci.edu/ml/.

[17] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The PROMISE Repository of empirical software engineering data," West Virginia University, Department of Computer Science, 2012, http://promisedata.googlecode.com/.

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

[19] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, no. 32, pp. C675–C701, 1937.

[20] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Annals of Mathematical Statistics*, vol. 11, pp. 86–92, 1940.

[21] M. William and S. Terry, *Statistics for Engineering and the Sciences*, Pearson, London, UK, 5th edition, 2006.