Information and Software Technology xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software defect prediction based on kernel PCA and weighted extreme learning machine[☆]

Zhou Xu[a,b], Jin Liu[a,f,*], Xiapu Luo[b], Zijiang Yang[c], Yifeng Zhang[a], Peipei Yuan[d], Yutian Tang[b], Tao Zhang[e]

[a] *School of Computer Science, Wuhan University, Wuhan, China*
[b] *Department of Computing, The Hong Kong Polytechnic University, Hong Kong*
[c] *Department of Computer Science, Western Michigan University, Michigan, USA*
[d] *School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China*
[e] *College of Computer Science and Technology, Harbin Engineering University, China*
[f] *Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.*

ARTICLE INFO

Keywords:
Feature extraction
Nonlinear mapping
Kernel principal component analysis
Weighted extreme learning machine

ABSTRACT

Context: Software defect prediction strives to detect defect-prone software modules by mining the historical data. Effective prediction enables reasonable testing resource allocation, which eventually leads to a more reliable software.
Objective: The complex structures and the imbalanced class distribution in software defect data make it challenging to obtain suitable data features and learn an effective defect prediction model. In this paper, we propose a method to address these two challenges.
Method: We propose a defect prediction framework called *KPWE* that combines two techniques, i.e., *Kernel Principal Component* Analysis (*KPCA*) and *Weighted Extreme Learning Machine* (*WELM*). Our framework consists of two major stages. In the first stage, KPWE aims to extract representative data features. It leverages the KPCA technique to project the original data into a latent feature space by nonlinear mapping. In the second stage, KPWE aims to alleviate the class imbalance. It exploits the WELM technique to learn an effective defect prediction model with a weighting-based scheme.
Results: We have conducted extensive experiments on 34 projects from the PROMISE dataset and 10 projects from the NASA dataset. The experimental results show that KPWE achieves promising performance compared with 41 baseline methods, including seven basic classifiers with KPCA, five variants of KPWE, eight representative feature selection methods with WELM, 21 imbalanced learning methods.
Conclusion: In this paper, we propose KPWE, a new software defect prediction framework that considers the feature extraction and class imbalance issues. The empirical study on 44 software projects indicate that KPWE is superior to the baseline methods in most cases.

1. Introduction

Software testing is an important part of software development life cycle for software quality assurance [1,2]. Defect prediction can assist the quality assurance teams to reasonably allocate the limited testing resources by detecting the potentially defective software modules (such as classes, files, components) before releasing the software product. Thus, effective defect prediction can save testing cost and improve software quality [3–5].

The majority of existing researches leverages various machine learning techniques to build defect prediction methods. In particular, many classification techniques have been used as defect prediction models, such as decision tree [6], Naive Bayes [7], random forest [8,9], nearest neighbor [10], support vector machine [11,12], neural network [13–15], logistic regression [16], and ensemble methods [17,18]. Since irrelevant and redundant features in the defect data may degrade the performance of the classification models, different feature selection methods have been applied to select an optimal feature subset for defect prediction[19]. These methods can be roughly divided into three categories: the filter-based feature ranking methods, wrapper-based feature subset evaluation methods, and extraction-based feature transformation methods, such as *Principal Component Analysis (PCA)* [20].

1.1. Motivation

Selecting optimal features that can reveal the intrinsic structures of the defect data is crucial to build effective defect prediction mod-

☆ Fully documented templates are available in the elsarticle package on CTAN.
* Corresponding author.
E-mail address: jinliu@whu.edu.cn (J. Liu).

https://doi.org/10.1016/j.infsof.2018.10.004
Received 21 September 2017; Received in revised form 26 August 2018; Accepted 5 October 2018
Available online xxx
0950-5849/© 2018 Elsevier B.V. All rights reserved.

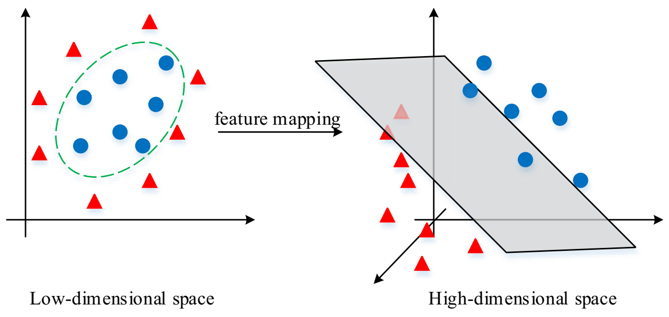


Fig. 1. An example of the merit of feature mapping.

els. The filter-based and wrapper-based feature selection methods only select a subset of the original features without any transformation [21]. However, such raw features may not properly represent the essential structures of raw defect data [22]. Being a linear feature extraction method, PCA has been widely used to transform the raw features to a low-dimensional space where the features are the linear combinations of the raw ones [23–26]. PCA performs well when the data are linearly separable and follow a Gaussian distribution, whereas the real defect data may have complex structures that can not be simplified in a linear subspace [27,28]. Therefore, the features extracted by PCA are usually not representative, and cannot gain anticipated performance for defect prediction [19,29]. To address this issue, we exploit KPCA [30], a non-linear extension of PCA, to project the original data into a latent high-dimensional feature space in which the mapped features can properly characterize the complex data structures and increase the probability of linear separability of the data. When the original data follow an arbitrary distribution, the mapped data by KPCA obey an approximate Gaussian distribution. Fig. 1 shows the merit of the feature mapping, where the data are linearly inseparable within the low-dimensional space but linearly separable within the high-dimensional space. Existing studies have shown that KPCA outperforms PCA [31,32].

Although many classifiers have been used for defect prediction, Lessmann et al. [33] suggested that the selection of classifiers for defect prediction needs to consider additional criteria, such as computational efficiency and simplicity, because they found that there are no significant performance differences among most defect prediction classifiers. Moreover, class imbalance is prevalent in defect data in which the non-defective modules usually outnumber the defective ones. It makes most classifiers tend to classify the minority samples (i.e., the defective modules) as the majority samples (i.e., the non-defective modules). However, existing defect prediction methods did not address this problem well, thus leading to unsatisfactory performance. In this work, we exploit *Single-hidden Layer Feedforward Neural networks (SLFNs)* called *Weighted Extreme Learning Machine (WELM)* [34] to overcome this challenge. WELM assigns higher weights to defective modules to emphasize their importance. In addition, WELM is efficient and convenient since it only needs to adaptively set the number of hidden nodes while other parameters are randomly generated instead being tuned through iterations like traditional neural networks [35].

In this paper, we propose a new defect prediction framework called KPWE that leverages the two aforementioned techniques: KPCA and WELM. This framework consists of two major stages. First, KPWE exploits KPCA to map original defect data into a latent feature space. The mapped features in the space can well represent the original ones. Second, with the mapped features, KPWE applies WELM to build an efficient and effective defect prediction model that can handle imbalanced defect data.

We conduct extensive experiments on 44 software projects from two datasets (PROMISE dataset and NASA dataset) with four indicators, i.e., F-measure, G-measure, MCC, and AUC. On average, KPWE achieves average F-measure, G-measure, MCC, and AUC values of 0.500, 0.660, 0.374, and 0.764 on PROMISE dataset, of 0.410, 0.611, 0.296 and 0.754 on NASA dataset, and of 0.480, 0.649, 0.356, and 0.761 across 44 projects of the two datasets. We compare KPWE against 41 baseline methods. The experimental results show that KPWE achieves significantly better performance (especially in terms of F-measure, MCC, and AUC) compared with all baseline methods.

1.2. Organization

The remainder of this paper is organized as follows. Section 2 presents the related work. In Section 3, we describe the proposed method in detail. Section 4 elaborates the experimental setup. In Section 5, we report the experimental results of performance verification. Section 6 discusses the threats to validity. In Section 7, we draw the conclusion.

2. Related work

2.1. Feature selection for defect prediction

Some recent studies have investigated the impact of feature selection methods on the performance of defect prediction. Song et al. [4] suggested that feature selection is an indispensable part of a general defect prediction framework. Menzies et al. [7] found that Naive Bayes classifier with Information Gain based feature selection can get good performances over 10 projects from the NASA dataset. Shivaji et al. [36,37] studied the performance of filter-based and wrapper-based feature selection methods for bug prediction. Their experiments showed that feature selection can improve the defect prediction performance even remaining 10% of the original features. Wold et al. [20] investigated four filter-based feature selection methods on a large telecommunication system and found that the Kolmogorov–Smirnov method achieved the best performance. Gao et al. [38] explored the performance of their hybrid feature selection framework based on seven filter-based and three feature subset search methods. They found that the reduced features would not adversely affect the prediction performance in most cases. Chen et al. [39] modelled the feature selection as a multi-objective optimization problem: minimizing the number of selected features and maximizing the defect prediction performance. They conducted experiments on 10 projects from PROMISE dataset and found that their method outperformed three wrapper-based feature selection methods. However, their method was less efficient than two wrapper-based methods. Catal and Diri [40] conducted an empirical study to investigate the impact of the dataset size, the types of feature sets and the feature selection methods on defect prediction. To study the impact of feature selection methods, they first utilized a *Correlation-based Feature Selection (CFS)* method to obtain the relevant features before training the classification models. The experiments on five projects from NASA dataset showed that the random forest classifier with CFS performed well on large project datasets and the Naive Bayes classifier with CFS worked well on small projects datasets. Xu et al. [19] conducted an extensive empirical comparison to investigate the impact of 32 feature selection methods on defect prediction performance over three public defect datasets. The experimental results showed that the performances of these methods had significant differences on all datasets and that PCA performed the worst. Ghotra et al. [41] extended Xu et al.'s work and conducted a large-scale empirical study to investigate the defect prediction performance of 30 feature selection methods with 21 classification models. The experimental results on 18 projects from NASA and PROMISE datasets suggested that correlation-based filter-subset feature selection method with best-first search strategy achieved the best performance among all other feature selection methods on majority projects.

2.2. Various classifiers for defect prediction

Various classification models have been applied to defect prediction. Malhotra [42] evaluated the feasibility of seven classification models for defect prediction by conducting a systematic literature review on the studies that published from January 1991 to October 2013. They discussed the merits and demerits of the classification models and found that they were superior to traditional statistical models. In addition, they suggested that new methods should be developed to further improve the defect prediction performance. Malhotra [43] used the statistical tests to compare the performance differences among 18 classification models for defect prediction. They performed the experiments on seven Android software projects and stated that these models have significant differences while support vector machine and voted perceptron model did not perform well. Lessmann et al. [33] conducted an empirical study to investigate the effectiveness of 21 classifiers on NASA dataset. The results showed that the performances of most classifiers have no significant differences. They suggested that some additional factors, such as the computational overhead and simplicity, should be considered when selecting a proper classifier for defect prediction. Ghotra et al. [44] expanded Lessmann's experiment by applying 31 classifiers to two versions of NASA dataset and PROMISE dataset. The results showed that these classifiers achieved similar results on the noisy NASA dataset but different performance on the clean NASA and the PROMISE datasets. Malhotra and Raje [45] investigated the performances of 18 classifiers on six projects with object-oriented features and found that Naive Bayes classifier achieved the best performance. Although some researchers introduced KPCA into defect prediction [46–48] recently, they aimed at building asymmetrical prediction models with the kernel method by considering the relationship between principal components and the class labels. In this work, we leverage KPCA as a feature selection method to extract representative features for defect prediction. In addition, Mesquita et al. [49] proposed a method based on ELM with reject option (i.e., IrejoELM) for defect prediction. The results were good because they abandoned the modules that have contradictory decisions for two designed classifiers. However, in practice, such modules should be considered.

2.3. Class imbalanced learning for defect prediction

Since class imbalance issue can hinder defect prediction techniques to achieve satisfactory performance, researchers have proposed different imbalanced learning methods to mitigate such negative effects. Sampling based methods and cost-sensitive based methods are the most studied imbalanced learning methods for defect prediction.

For the sampling based imbalanced learning methods, there are two main sampling strategies to balance the data distribution. One is to decrease the number of non-defective modules (such as under-sampling technique), the other is to increase the number of the defective modules with redundant modules (such as over-sampling technique) or synthetic modules (such as *Synthetic Minority Over-sampling Technique, SMOTE*). Kamei et al. [50] investigated the impact of four sampling methods on the performance of four basic classification models. They conducted experiments on two industry legacy software systems and found that these sampling methods can benefit linear and logistic models but were not helpful to neural network and classification tree models. Bennin et al. [51] assessed the statistical and practical significance of six sampling methods on the performance of five basic defect prediction models. Experiments on 10 projects indicated that these sampling methods had statistical and practical effects in terms of some performance indicators, such as Pd, Pf, G-mean, but had no effect in terms of AUC. Bennin et al. [52] explored the impact of a configurable parameter (i.e, the percentage of defective modules) in seven sampling methods on the performance of five classification models. The experimental results showed that this parameter can largely impact the performance (except AUC) of studied prediction models. Due to the contradictory conclusions of previous empirical studies about which imbalanced learning methods performed the best in the context of defect prediction models, Tantithamthavorn et al. [53] conducted a large-scale empirical experiment on 101 project versions to investigate the impact of four popularly-used sampling techniques on the performance and interpretation of seven classification models. The experimental results explained that these sampling methods increased the completeness of Recall indicator but had no impact on the AUC indicator. In addition, the sampling based imbalanced learning methods were not conducive to the understanding towards the interpretation of the defect prediction models.

The cost-sensitive based imbalanced learning methods alleviate the differences between the instance number of two classes by assigning different weights to the two types of instances. Khoshgottar et al. [54] proposed a cost-boosting method by combining multiple classification models. Experiments on two industrial software systems showed that the boosting method was feasible for defect prediction. Zheng [55] proposed three cost-sensitive boosting methods to boost neural networks for defect prediction. Experimental results showed that threshold-moving-based boosting neural networks can achieve better performance, especially for object-oriented software projects. Liu et al. [56] proposed a novel two-stage cost-sensitive learning method by utilizing cost information in the classification stage and the feature selection stage. Experiments on seven projects of NASA dataset demonstrated its superiority compared with the single-stage cost-sensitive classifiers and cost-blind feature selection methods. Siers and Islam [57] proposed two cost-sensitive classification models by combining decision trees to minimize the classification cost for defect prediction. The experimental results on six projects of NASA dataset showed the superiority of their methods compared with six classification methods. The WELM technique used in our work belongs to this type of imbalanced learning methods.

3. KPWE: The new framework

The new framework consists of two stages: feature extraction and model construction. This section first describes how to project the original data into a latent feature space using the nonlinear feature transformation technique KPCA, and then presents how to build the WELM model with the extracted features by considering the class imbalance issue.

3.1. Feature extraction based on KPCA

In this stage, we extract representative features with KPCA to reveal the potentially complex structures in the defect data. KPCA uses a nonlinear mapping function φ to project each raw data point within a low-dimensional space into a new point within a high-dimensional feature space *F*.

Given a dataset $\{x_i, y_i\}, i = 1, 2, \ldots, n$, where $x_i = [x_{i1}, x_{i2}, \ldots, x_{im}]^{\mathrm{T}} \in \Re^m$ denotes the feature set and $y_i = [y_{i1}, y_{i2}, \ldots, y_{ic}]^{\mathrm{T}} \in \Re^c$ ($c = 2$ in this work) denotes the label set. Assuming that each data point x_i is mapped into a new point $\varphi(x_i)$ and the mapped data points are centralized, i.e.,

$$\frac{1}{n}\sum_{i=1}^{n}\varphi(x_i) = 0 \tag{1}$$

The covariance matrix **C** of the mapped data is:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}\varphi(x_i)\varphi(x_i)^{\mathrm{T}} \tag{2}$$

To perform the linear PCA in *F*, we diagonalize the covariance matrix *C*, which can be treated as a solution of the following eigenvalue problem

$$\mathbf{CV} = \lambda\mathbf{V}, \tag{3}$$

where λ and **V** denote the eigenvalues and eigenvectors of *C*, respectively.

Since all solutions **V** lie in the span of the mapped data points $\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n)$, we multiply both sides of Eq. (3) by $\varphi(x_l)^T$ as

$$\varphi(x_l)^{\mathrm{T}}\mathbf{CV} = \lambda\varphi(x_l)^{\mathrm{T}}\mathbf{V}, \forall l = 1, 2, \ldots, n \tag{4}$$

Meanwhile, there exist coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ that linearly express the eigenvectors **V** of **C** with $\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n)$, i.e.,

$$\mathbf{V} = \sum_{j=1}^{n}\alpha_j\varphi(x_j) \tag{5}$$

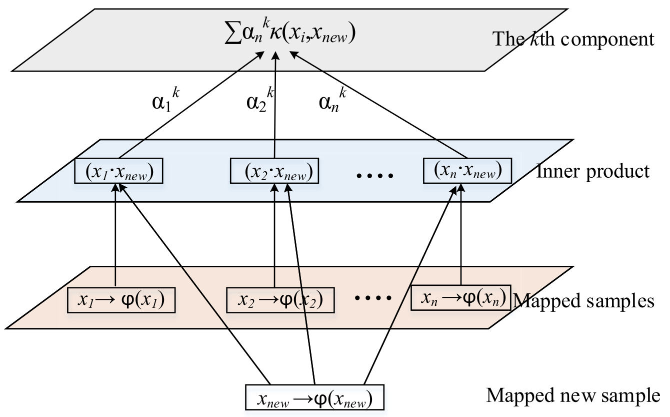

$\sum \alpha_n^k \kappa(x_i, x_{new})$
The kth component
α_1^k
α_2^k
α_n^k
$(x_1 \cdot x_{new})$
$(x_2 \cdot x_{new})$
$(x_n \cdot x_{new})$
Inner product
$x_1 \rightarrow \varphi(x_1)$
$x_2 \rightarrow \varphi(x_2)$
$x_n \rightarrow \varphi(x_n)$
Mapped samples
$x_{new} \rightarrow \varphi(x_{new})$
Mapped new sample

Fig. 2. Feature extraction with KPCA.

Eq. (4) can be rewritten as following formula by substituting Eqs. (2) and (5) into it

$$\frac{1}{n}\varphi(x_l)^{\mathrm{T}} \sum_{i=1}^{n} \varphi(x_i)\varphi(x_i)^{\mathrm{T}} \sum_{j=1}^{n} \alpha_j \varphi(x_j) = \lambda \varphi(x_l)^{\mathrm{T}} \sum_{j=1}^{n} \alpha_j \varphi(x_j) \tag{6}$$

Let the kernel function $\kappa(x_i, x_j)$ be

$$\kappa(x_i, x_j) = \varphi(x_i)^{\mathrm{T}} \varphi(x_j) \tag{7}$$

Then Eq. (6) is rewritten as

$$\frac{1}{n} \sum_{l=1,i=1}^{n} \kappa(x_l, x_i) \sum_{i=1,j=1}^{n} \alpha_j \kappa(x_i, x_j) = \lambda \sum_{l=1,j=1}^{n} \alpha_j \kappa(x_l, x_j) \tag{8}$$

Let the kernel matrix $\mathbf{K}$ with size $n \times n$ be

$$\mathbf{K}_{i,j} = \kappa(x_i, x_j) \tag{9}$$

Then Eq. (8) is rewritten as

$$\mathbf{K}^2 \alpha = n\lambda \mathbf{K}\alpha, \tag{10}$$

where $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]^{\mathrm{T}}$.

The solution of Eq. (10) can be obtained by solving the eigenvalue problem

$$\mathbf{K}\alpha = n\lambda\alpha \tag{11}$$

for nonzero eigenvalues λ and corresponding eigenvectors $\boldsymbol{\alpha}$. As we can see, all the solutions of Eq. (11) satisfy Eq. (10).

As mentioned above, we first assume that the mapped data points are centralized. If they are not centralized, the Gram matrix $\tilde{K}$ be used to replace the kernel matrix $\mathbf{K}$ as

$$\tilde{\mathbf{K}} = \mathbf{K} - 1_n\mathbf{K} - \mathbf{K}1_n + 1_n\mathbf{K}1_n, \tag{12}$$

where 1_n denotes the $n \times n$ matrix with all values equal to $1/n$.

Thus, we just need to solve the following formula

$$\tilde{\mathbf{K}}\alpha = n\lambda\alpha \tag{13}$$

To extract the nonlinear principal components of a new test data point $\varphi(x_{new})$, we can compute the projection of the kth kernel component by

$$\mathbf{V}^k \cdot \varphi(x_{new}) = \sum_{i=1}^{n} \alpha_i^k \varphi(x_i)^{\mathrm{T}} \varphi(x_{new}) = \sum_{i=1}^{n} \alpha_i^k \kappa(x_i, x_{new}) \tag{14}$$

Fig. 2 depicts the process of KPCA for feature extraction. KPCA simplifies the feature mapping by calculating the inner product of two data points with kernel function instead of calculating the $\varphi(x_i)$ explicitly. Various kernel functions, such as Gaussian Radial Basic Function (RBF) kernel and polynomial kernel, can induce different nonlinear mapping. The RBF kernel is commonly used in image retrieval and pattern recognition domains [58,59] that is defined as

$$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{15}$$

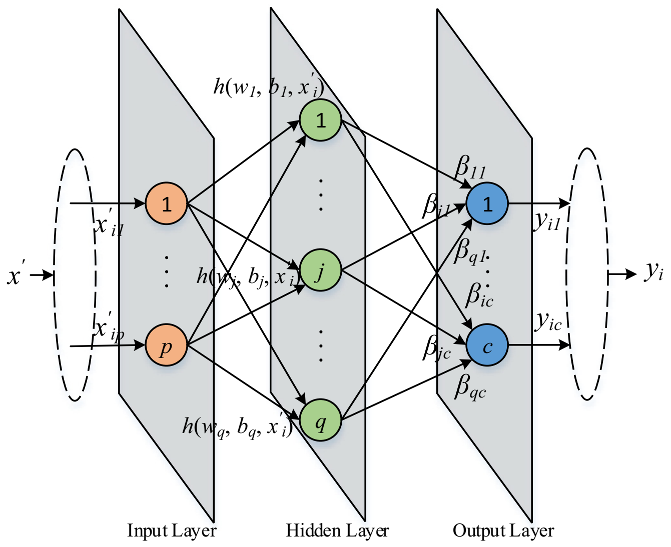


Fig. 3. The architecture of ELM.

where $\|\cdot\|$ denotes the l_2 norm and $2\sigma^2 = \omega$ denotes the width of the Gaussian RBF function.

To eliminate the underlying noise in the data, when performing the PCA in the latent feature space F, we maintain the most important principal components that capture at least 95% of total variances of the data according to their cumulative contribution rates [60]. Finally, the data are mapped into a p-dimensional space.

After completing feature extraction, the original training data are transformed to a new dataset $\{x'_i, y_i\} \in \Re^p \times \Re^c$ $(i = 1, 2, \ldots, n)$.

3.2. ELM

Before formulizing the WELM, we first introduce the basic ELM. With the mapped dataset $\{x'_i, y_i\} \in \Re^p \times \Re^c$ $(i = 1, 2, \ldots, n)$, the output of the generalized SLFNs with q hidden nodes and activation function $\mathbf{h}(x')$ is formally expressed as

$$o_i = \sum_{k=1}^{q} \beta_k h_k(x'_i) = \sum_{k=1}^{q} \beta_k h(w_k, b_k, x'_i), \tag{16}$$

where $i = 1, 2, \ldots, n$, $w_k = [w_{k1}, w_{k2}, \ldots, w_{kp}]^{\mathrm{T}}$ denotes the input weight vector connecting the input nodes and the kth hidden node, b_k denotes the bias of the k-th hidden node, $\beta_k = [\beta_{k1}, \beta_{k2}, \ldots, \beta_{kc}]^{\mathrm{T}}$ denotes the output weight vector connecting the output nodes and the kth hidden node, and o_i denotes the expected output of the ith sample. The commonly-used activation functions in ELM include sigmoid function, Gaussian RBF function, hard limit function, and multiquadric function [61,62]. Fig. 3 depicts the basic architecture of ELM.

Eq. (16) can be equivalently rewritten as

$$\mathbf{H}\beta = \mathbf{O}, \tag{17}$$

where $\mathbf{H}$ is called the hidden layer output matrix of the SLFNs and is defined as

$$\mathbf{H} = \mathbf{H}(w_1, \ldots, w_q, b_1, \ldots, b_q, x'_1, \ldots, x'_n) = \begin{bmatrix} \mathbf{h}(x'_1) \\ \vdots \\ \mathbf{h}(x'_n) \end{bmatrix} = \begin{bmatrix} h(w_1, b_1, x'_1) & \cdots & h(w_q, b_q, x'_1) \\ \vdots & \ddots & \vdots \\ h(w_1, b_1, x'_n) & \cdots & h(w_q, b_q, x'_n) \end{bmatrix}_{n \times q}, \tag{18}$$

where the ith row of $\mathbf{H}$ denotes the output vector of the hidden layer with respect to input sample x'_i, and the kth column of $\mathbf{H}$ denotes the output vector of the kth hidden node with respect to the input samples $x'_1, x'_2, \ldots, x'_n$.

β denotes the weight matrix connecting the hidden layer and the output layer, which is defined as

$$\beta = \begin{bmatrix} \beta_1^{\mathrm{T}} \\ \vdots \\ \beta_q^{\mathrm{T}} \end{bmatrix}_{q\times c} \tag{19}$$

O denotes the expected label matrix, and each row represents the output vector of one sample. **O** is defined as

$$\mathbf{O} = \begin{bmatrix} o_1^{\mathrm{T}} \\ \vdots \\ o_n^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} o_{11} & \cdots & o_{1c} \\ \vdots & \ddots & \vdots \\ o_{n1} & \cdots & o_{nc} \end{bmatrix}_{n\times c} \tag{20}$$

Since the target of training SLFNs is to minimize the output error, i.e., approximating the input samples with zero error as follows

$$\textstyle\sum_{i=1}^{n} \|o_i - y_i\| = \|\mathbf{O} - \mathbf{Y}\| = 0 \tag{21}$$

where $\mathbf{Y} = \begin{bmatrix} y_1^{\mathrm{T}} \\ \vdots \\ y_n^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1c} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nc} \end{bmatrix}_{n\times c}$ denotes the target output matrix.

Then, we need to solve the following formula

$$\mathbf{H}\beta = \mathbf{Y} \tag{22}$$

Huang et al. [35,63] proved that, for ELM, the weights w_k of the input connection and the bias b_k of the hidden layer node can be randomly and independently designated. Once these parameters are assigned, Eq. (22) is converted into a linear system and the output weight matrix $\boldsymbol{\beta}$ can be analytically determined by finding the least-square solution of the linear system, i.e.,

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{Y}\| \tag{23}$$

The optimal solution of Eq. (23) is

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y} = (\mathbf{H}^{T}\mathbf{H}) \tag{24}$$

where **H**† denotes the Moore–Penrose generalized inverse of the hidden layer output matrix **H** [64,65]. The obtained $\hat{\beta}$ can ensure minimum training error, get optimal generalization ability and avoid plunging into local optimum since $\hat{\beta}$ is unique [35]. This solution can also be obtained with Karush–Kuhn–Tucker (KKT) theorem [66].

Finally, we get the classification function of ELM as

$$f(x') = \mathbf{h}(x')\hat{\beta} = \mathbf{h}(x')\mathbf{H}^{\dagger}\mathbf{Y} \tag{25}$$

3.3. Model construction based on WELM

For imbalanced data, to consider the different importance of the majority class samples (i.e., defective modules) and the minority class samples (i.e., non-defective modules) when building the ELM classifier, we define a $n\times n$ diagonal matrix **W**, whose diagonal element $\mathbf{W_{ii}}$ denotes the weight of training sample x_i'. More precisely, if x_i' belongs to the majority class, the weight $\mathbf{W_{ii}}$ is relatively lower than the sample that belongs to the minority class. According to the KKT theorem, Eq. (24) is rewritten as

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y} = (\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{26}$$

Then, Eq. (25) becomes

$$f(x') = \mathbf{h}(x')\hat{\beta} = \mathbf{h}(x')(\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{27}$$

There are mainly two schemes for assigning the weights to the samples of the two classes as follows [34]:

$$\mathbf{W1} = \mathbf{W_{ii}} = \begin{cases} 1/n_P & \text{if } x_i' \in \text{minority class} \\ 1/n_N & \text{if } x_i' \in \text{majority class} \end{cases}, \tag{28}$$

or

$$\mathbf{W2} = \mathbf{W_{ii}} = \begin{cases} 0.618/n_P & \text{if } x_i' \in \text{minority class} \\ 1/n_N & \text{if } x_i' \in \text{majority class} \end{cases}, \tag{29}$$

where **W1** and **W2** denote two weighting schemes, n_P and n_N indicate the number of samples of the minority and majority class, respectively. The golden ratio of 0.618:1 between the majority class and the minority class in scheme **W2** represents the perfection in nature [67].

4. Experimental setup

In this section, we elaborate the experimental setup, including the *Research Questions (RQs)*, benchmark datasets, the performance indicators, and the experimental design.

4.1. Research questions

We design the following five research questions to evaluate our KPWE method.

RQ1: How efficient are ELM and WELM?

As the computational cost is an important criterion to select the appropriate classifier for defect prediction in practical application [33,63], this question is designed to evaluate the efficiency of ELM and its variant WELM compared with some typical classifiers.

RQ2: How effective is KPWE compared with basic classifiers with KPCA?

Since our method KPWE combines feature transformation and an advanced classifier, this question is designed to explore the effectiveness of this new classifier compared against the typical classifiers with the same process of feature extraction. We use the classic classifiers in RQ1 with KPCA as the baseline methods.

RQ3: Is KPWE superior to its variants?

Since the two techniques KPCA and WELM used in our method are variants of the linear feature extraction method PCA and the original ELM respectively, this question is designed to investigate whether our method is more effective than other combinations of these four techniques. To answer this question, we first compare KPWE against the baseline methods that combine WELM with PCA (short for PCAWELM) and none feature extraction (short for WELM). It can be used to investigate the different performance among the methods using non-linear, linear and none feature extraction for WELM. Then, we compare KPWE against the baseline methods that combine ELM with KPCA, PCA, and none feature extraction (short for KPCAELM, PCAELM, and ELM, respectively). It can be used to compare the performance of our method against its downgraded version methods that do not consider the class imbalance issue. All these baseline methods are treated as the variants of KPWE.

RQ4: Are the selected features by KPCA more effective for performance improvement than that by other feature selection methods?

To obtain the representative features of the defect data, previous researches [19,41] used various feature selection methods to select an optimal feature subset to replace the original set. This question is designed to investigate whether the features extracted by KPCA are more effective in improving the defect prediction performance than the features selected by other feature selection methods. To answer this question, we select some classic filter-based feature ranking methods and wrapper-based feature subset selection methods with the same classifier WELM for comparison.

RQ5: Is the prediction performance of KPWE comparable to that of other imbalanced learning methods?

Since our method KPWE is customized to address the class imbalance issue for software defect data, this question is designed to study whether our method can achieve better or at least comparable performance than existing imbalanced learning methods. To answer this question, we employ several sampling-based, ensemble learning-based, and cost-sensitive-based imbalanced learning methods for comparison.

Table 1
Statistics of the PROMISE dataset.

Projects	# M	# D	(%)D	Projects	# M	# D	(%)D
ant−1.3	125	20	16.00	lo4j−1.0	135	34	25.19
ant−1.4	178	40	22.47	log4j−1.1	109	37	33.94
ant−1.5	293	32	10.92	lucene−2.0	195	91	46.67
ant−1.6	351	92	26.21	poi−2.0	314	37	11.78
ant−1.7	745	166	22.28	prop−6	660	66	10.00
arc	234	27	11.54	redaktor	176	27	15.34
camel−1.0	339	13	3.83	synapse−1.0	157	16	10.19
camel−1.2	608	216	35.53	synapse−1.1	222	60	27.03
camel−1.4	872	145	16.63	synapse−1.2	256	86	33.59
camel−1.6	965	188	19.48	tomcat	858	77	8.97
ivy−1.4	241	16	6.64	velocity−1.6	229	78	34.06
ivy−2.0	352	40	11.36	xalan−2.4	723	110	15.21
jedit−3.2	272	90	33.09	xalan−2.5	803	387	48.19
jedit−4.0	306	75	24.51	xalan−2.6	885	411	46.44
jedit−4.1	312	79	25.32	xerces-init	162	77	47.53
jedit−4.2	367	48	13.08	xerces−1.2	440	71	16.14
jedit−4.3	492	11	2.24	xerces−1.3	453	69	15.23

4.2. Benchmark dataset

We conduct extensive experiments on 34 projects taken from an open-source PROMISE data repository,[1] which have been widely used in many defect prediction studies [44,68,69]. These projects include open-source projects (such as 'ant' project), proprietary projects (such as 'prop' project) and academic projects (such as 'redaktor' project). Each module in the projects includes 20 object-oriented features and a dependent variable that denotes the number of defects in the module. These features are collected by Jureczko and Madeyski, Spinellis with Ckjm tool [70,71]. We label the module as 1 if it contains one or more defects. Otherwise, we label it as 0. In this work, we just select a subset from PROMISE data repository as our benchmark dataset. The selection criteria are that: First, to ensure a certain amount of training set and test set, we filter out the projects that have less than 100 modules. Second, since our method KPWE is designed to address the imbalanced defect data where the non-defective modules outnumber the defective ones, we only consider the projects whose defective ratios are lower than 50%. As a result, 34 versions of 15 projects are selected and used in this study. To investigate the generalization of our method to other datasets, we further conduct experiments on ten projects from NASA dataset which is cleaned by Shepperd et al. [27]. Since there are two cleaned versions (D′ and D″) of NASA dataset, in this work, we use the D″ version as our benchmark dataset as in previous work [44].

Tables 1 and 2 summarize the basic information of the two datasets, including the number of features (# F), the number of modules (# M), the number of defective modules (# D) and the defect ratios (% D). Note that we do not report the number of features for the projects in PROMISE dataset since all of them contain 20 features. In addition, for PROMISE dataset, the feature descriptions and corresponding abbreviations are presented in Table 3 (CC is the abbreviations of Cyclomatic Complexity). For NASA dataset, Table 4 depicts the common features among the 10 projects and Table 5 tabulates the other specific features for each project with symbol $\surd$.

4.3. Performance indicators

We use F-measure, G-measure, *Matthews Correlation Coefficient (MCC)* and *Area Under the ROC Curve (AUC)* to measure the performance of KPWE, because they are widely used in defect prediction [44,69,72,73]. The first three indicators can be deduced by some simpler binary classification metrics as listed in Table 6.

[1] http://openscience.us/repo/defect/ck/.

Possibility of detection (*pd*) or *recall* is defined as the ratio of the number of defective modules that are correctly predicted to the total number of defective modules.

Possibility of false alarm (*pf*) is defined as the ratio of the number of defective modules that are incorrectly predicted to the total number of non-defective modules.

Precision is defined as the ratio of the number of defective modules that are correctly predicted to the total number of defective modules that are correctly and incorrectly predicted.

F-measure, a trade-off between *recall* and *precision*, is defined as

$$\text{F-measure} = \frac{2 * recall * precision}{recall + precision}. \tag{30}$$

G-measure, a trade-off between *pd* and *pf*, is defined as

$$\text{G-measure} = \frac{2 * pd * (1 - pf)}{pd + (1 - pf)}. \tag{31}$$

MCC, a comprehensive indicator by considering *TP, TN, FP*, and *FN*, is defined as

$$\text{MCC} = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}. \tag{32}$$

AUC calculates the area under a ROC curve which depicts the relative trade-off between *pd* (the y-axis) and *pf* (the x-axis) of a binary classification. Different from the above three indicators which are based on the premise that the threshold of determining a sample as positive class is 0.5 by default, the value of AUC is independent of the decision threshold. More specifically, given a threshold, we can get a point pair (*pd,pf*) and draw the corresponding position in the two-dimension plane. For all possible thresholds, we can get a set of such point pairs. The ROC curve is made up by connecting all these points. The area under this curve is used to evaluate the classification performance.

The greater values of the four indicators indicate better prediction performance.

4.4. Experimental design

We perform substantial experiments to evaluate the effectiveness of KPWE. In the feature extraction phase, we choose the Gaussian RBF as the kernel function for KPCA since it usually exhibits better performances in many applications [58,59,74]. In terms of the parameter ω, i.e., the width of the Gaussian kernel (as defined in Section 3.1), we empirically set a relatively wide range as $\omega = 10^2, 20^2, \ldots, 100^2$. In the model construction phase, we also choose the Gaussian RBF as the activation function for WELM because it is the preferred choice in many applications [59,75]. Since the number of hidden nodes q is far less than the number of training sample n [35], we set the number of hidden nodes from 5 to n with an increment of 5. So, for each project, there are $2n(10 \times \frac{n}{5})$ combinations of ω and q in total. For the weighting scheme of **W**, we adopt the second scheme **W2** as described in Section 3.3. For each project, we use the 50:50 split with stratified sampling to constitute the training and test set. More specifically, we utilize stratified sampling to randomly select 50% instances as the training set and the remaining 50% instances as the test set. The stratified sampling strategy guarantees the same defect ratios of the training set and test set which conforms to the actual application scenario. In addition, such sampling setting helps reduce sampling biases [76]. The 50:50 split and stratified sampling are commonly used in previous defect prediction studies [22,77–79]. To mitigate the impact of the random division treatment on the experimental results and produce a general conclusion, we repeat this process 30 times on each project by reshuffling the module order. Therefore, for each parameter combination, we run KPWE 30 times and record the average indicator values. Finally, the optimal combination of parameters ω and q is determined by the best average F-measure value. In this work, we report the average values of the four indicators on 30-rounds experiments.

Table 2
Statistics of the NASA dataset.

Projects	# F	# M	# D	(%)D	Projects	# F	# M	# D	(%)D
CM1	37	327	42	12.84	MW1	37	251	25	9.96
KC1	21	1162	294	25.30	PC1	37	696	55	7.90
KC3	39	194	36	18.56	PC3	37	1073	132	12.30
MC1	38	1847	36	1.95	PC4	37	1276	176	13.79
MC2	39	125	44	35.20	PC5	38	1679	459	27.34

Table 3
The feature description and abbreviation for PROMISE dataset.

1. Weighted Methods per Class (WMC)	11. Measure of Functional Abstraction (MFA)
2. Depth of Inheritance Tree (DIT)	12. Cohesion Among Methods of Class (CAM)
3. Number of Children (NOC)	13. Inheritance Coupling (IC)
4. Coupling between Object Classes (CBO)	14. Coupling Between Methods (CBM)
5. Response for a Class (RFC)	15. Average Method Complexity (AMC)
6. Lack of Cohesion in Methods (LOCM)	16. Afferent Couplings (Ca)
7. Lack of Cohesion in Methods (LOCM3)	17. Efferent Couplings (Ce)
8. Number of Public Methods (NPM)	18. Greatest Value of CC (Max_CC)
9. Data Access Metric (DAM)	19. Arithmetic mean value of CC (Avg_CC)
10. Measure of Aggregation (MOA)	20. Lines of Code (LOC)

Table 4
The description of the common feature for NASA dataset.

1. Line count of code	11. Halstead_Volume
2. Count of blank lines	12. Halstead_Level
3. Count of code and comments	13. Halstead_Difficulty
4. Count of comments	14. Halstead_Content
5. Line count of executable code	15. Halstead_Effort
6. Number of operators	16. Halstead_Error_Estimate
7. Number of operands	17. Halstead_Programming_Time
8. Number of unique operators	18. Cyclomatic_Complexity
9. Number of unique operands	19. Design_Complexity
10. Halstead_Length	20. Essential_Complexity

Table 6
Basic indicators for defect prediction.

	Predicted as defective	Predicted as defective-free
Actual defective	*TP*	*FN*
Actual defective-free	*FP*	*TN*
pd (*recall*)		$\frac{TP}{TP+FN}$
pf		$\frac{FP}{FP+TN}$
precision		$\frac{TP}{TP+FP}$

4.5. Statistical test method

To statistically analyze the performance between our method KPWE and other baseline methods, we perform the non-parametric Frideman test with the Nemenyi's post-hoc test [80] at significant level 0.05 over all projects. The Friedman test evaluates whether there exist statistically significant differences among the average ranks of different methods. Since Friedman test is based on performance ranks of the methods, rather than actual performance values, therefore it makes no assumptions on the distribution of performance values and is less susceptible to outliers [33,81]. The test statistic of the Friedman test can be calculated as follows:

$$\tau_{\chi^2} = \frac{12N}{L(L+1)}\left(\sum_{j=1}^{L} AR_j^2 - \frac{L(L+1)^2}{4}\right), \tag{33}$$

where N denotes the total number of the projects, L denotes the number of methods needed to be compared, $AR_j = \frac{1}{N}\sum_{i=1}^{N} R_i^j$ denotes the average rank of method j on all projects and R_i^j denotes the rank of jth method on the ith project. τ_{χ^2} obeys the χ^2 distribution with $L-1$ degree of freedom [82]. Since the original Friedman test statistic is too conservative, its variant τ_F is usually used to conduct the statistic test.

Table 5
The specific features for each project of NASA dataset.

Features	CM1	KC1	KC3	MC1	MC2	MW1	PC1	PC3	PC4	PC5
21. Number_of_lines	√		√	√	√	√	√	√	√	√
22. Cyclomatic_Density	√		√	√	√	√	√	√	√	√
23. Branch_Count	√	√	√	√	√	√	√	√	√	√
24. Essential_Density	√		√	√	√	√	√	√	√	√
25. Call_Pairs	√		√	√	√	√	√	√	√	√
26. Condition_Count	√		√	√	√	√	√	√	√	√
27. Decision_Count	√		√	√	√	√	√	√	√	√
28. Decision_Density	√		√		√	√	√	√	√	
29. Design_Density	√		√	√	√	√	√	√	√	√
30. Edge_Count	√		√	√	√	√	√	√	√	√
31. Global_Data_Complexity			√	√	√					√
32. Global_Data_Density			√	√	√					√
33. Maintenance_Severity	√		√	√	√	√	√	√	√	√
34. Modified_Condition_Count	√		√	√	√	√	√	√	√	√
35. Multiple_Condition_Count	√		√	√	√	√	√	√	√	√
36. Node_Count	√		√	√	√	√	√	√	√	√
37. Normalized_CC	√		√	√	√	√	√	√	√	√
38. Parameter_Count	√		√	√	√	√	√	√	√	√
39. Percent_Comments	√		√	√	√	√	√	√	√	√

Table 7
The parameter settings of the used machine learning classifiers.

Classifier	Parameter settings
NB	Estimator: kernel estimator
RF	Number of generated tree: 10, Number of variables for random feature selection: 2
BP	Layer: 3, Learning rate: 0.1, Maximal number of iterations: 2000, Tolerant error: 0.004
SVM	Kernel function: Gaussian RBF, Kernel parameter: $2^{-10}, 2^{-9}, , 2^{4}$, Cost parameter: $2^{-2}, 2^{-1}, 2^{12}$
NN	Number of neighbors used: 1
LR	The distribution used: normal
CART	The minimal number of observations per tree leaf: 1

Table 8
Training Time of classifiers on promise dataset (in Seconds).

Projects	NB	RF	LR	CART	BP	SVM	ELM	WELM
ant	0.085	0.181	0.019	0.030	2.933	8.089	0.008	**0.003**
arc	0.084	0.174	0.040	0.016	8.444	3.651	**0.003**	0.006
camel	0.084	0.171	0.050	0.050	9.050	21.985	0.061	**0.004**
ivy	0.086	0.168	0.014	0.020	6.222	5.233	0.006	**0.002**
jedit	0.100	0.168	0.032	0.034	4.414	7.869	0.008	**0.007**
log4j	0.066	0.150	0.007	0.014	0.465	2.181	**0.000**	**0.000**
lucene	0.088	**0.000**	0.073	0.004	0.666	7.887	0.006	0.003
poi	0.085	**0.000**	0.043	0.005	0.663	10.196	0.004	0.003
prop-6	0.086	0.170	0.144	0.056	11.793	14.179	0.042	**0.003**
redaktor	0.082	0.171	0.044	0.023	0.645	2.793	**0.000**	**0.000**
synapse	0.081	0.170	0.021	0.020	5.761	3.912	0.003	**0.000**
tomcat	0.082	0.206	0.023	0.058	6.267	21.958	0.055	**0.005**
velocity	0.087	0.170	0.012	0.017	14.742	4.154	0.003	**0.000**
xalan	0.080	0.223	0.024	0.077	6.836	26.410	0.028	**0.011**
xerces	0.084	0.192	0.026	0.039	3.898	8.112	**0.006**	0.008

Table 9
Training time of classifiers on nasa dataset (in Seconds).

Projects	NB	RF	LR	CART	BP	SVM	ELM	WELM
CM1	**0.004**	0.175	1.902	0.094	8.551	6.960	0.030	0.061
KC1	0.014	0.294	0.027	0.112	5.316	88.619	0.176	**0.005**
KC3	0.004	0.167	1.755	0.07	18.519	3.996	**0.003**	0.040
MC1	0.662	0.263	2.939	0.204	131.473	95.002	0.309	**0.108**
MC2	0.669	0.15	1.065	0.049	1.791	2.696	**0.003**	0.036
MW1	0.629	0.152	1.848	0.053	86.102	4.585	**0.006**	0.031
PC1	0.643	0.198	2.151	0.115	3.285	20.158	**0.041**	0.054
PC3	0.681	0.257	0.424	0.218	127.702	50.87	0.147	**0.061**
PC4	0.630	0.261	2.658	0.216	53.239	65.151	0.09	**0.073**
PC5	0.666	0.351	0.246	0.438	113.32	179.318	0.283	**0.087**

Table 10
Average indicator values of KPWE and seven basic classifiers with KPCA on two datasets and across all projects.

Dataset	Indicator	KPNB	KPNN	KPRF	KPLR	KPCART	KPBP	KPSVM	KPWE
PROMISE	F-measure	0.426	0.423	0.361	0.410	0.396	0.419	0.391	**0.500**
	G-measure	0.525	0.523	0.360	0.453	0.484	0.478	0.376	**0.660**
	MCC	0.284	0.257	0.235	0.292	0.222	0.260	0.280	**0.374**
	AUC	0.699	0.624	0.696	0.716	0.630	0.672	0.648	**0.764**
NASA	F-measure	0.354	0.336	0.267	0.325	0.315	0.352	0.310	**0.410**
	G-measure	0.476	0.477	0.264	0.387	0.425	0.429	0.287	**0.611**
	MCC	0.248	0.216	0.201	0.234	0.176	0.242	0.230	**0.296**
	AUC	0.708	0.596	0.693	0.698	0.606	0.684	0.655	**0.754**
ALL	F-measure	0.410	0.403	0.340	0.391	0.377	0.403	0.372	**0.480**
	G-measure	0.513	0.512	0.338	0.438	0.471	0.467	0.355	**0.649**
	MCC	0.276	0.248	0.228	0.279	0.212	0.256	0.269	**0.356**
	AUC	0.701	0.618	0.695	0.712	0.625	0.675	0.650	**0.761**

τ_F is calculated as the following formula:

$$\tau_F = \frac{(N-1)\tau_{\chi^2}}{N(L-1)-\tau_{\chi^2}}. \quad (34)$$

τ_F obeys the F-distribution with $L-1$ and $(L-1)(N-1)$ degrees of freedom. Once τ_F value is calculated, we can compare τ_F against critical values[2] for the F distribution and then determine whether to accept or reject the null hypothesis (i.e., all methods perform equally on the projects).

If the null hypothesis is rejected, it means that the performance differences among different methods are nonrandom, then a so-called Ne-

[2] http://www.socr.ucla.edu/applets.dir/f_table.html.

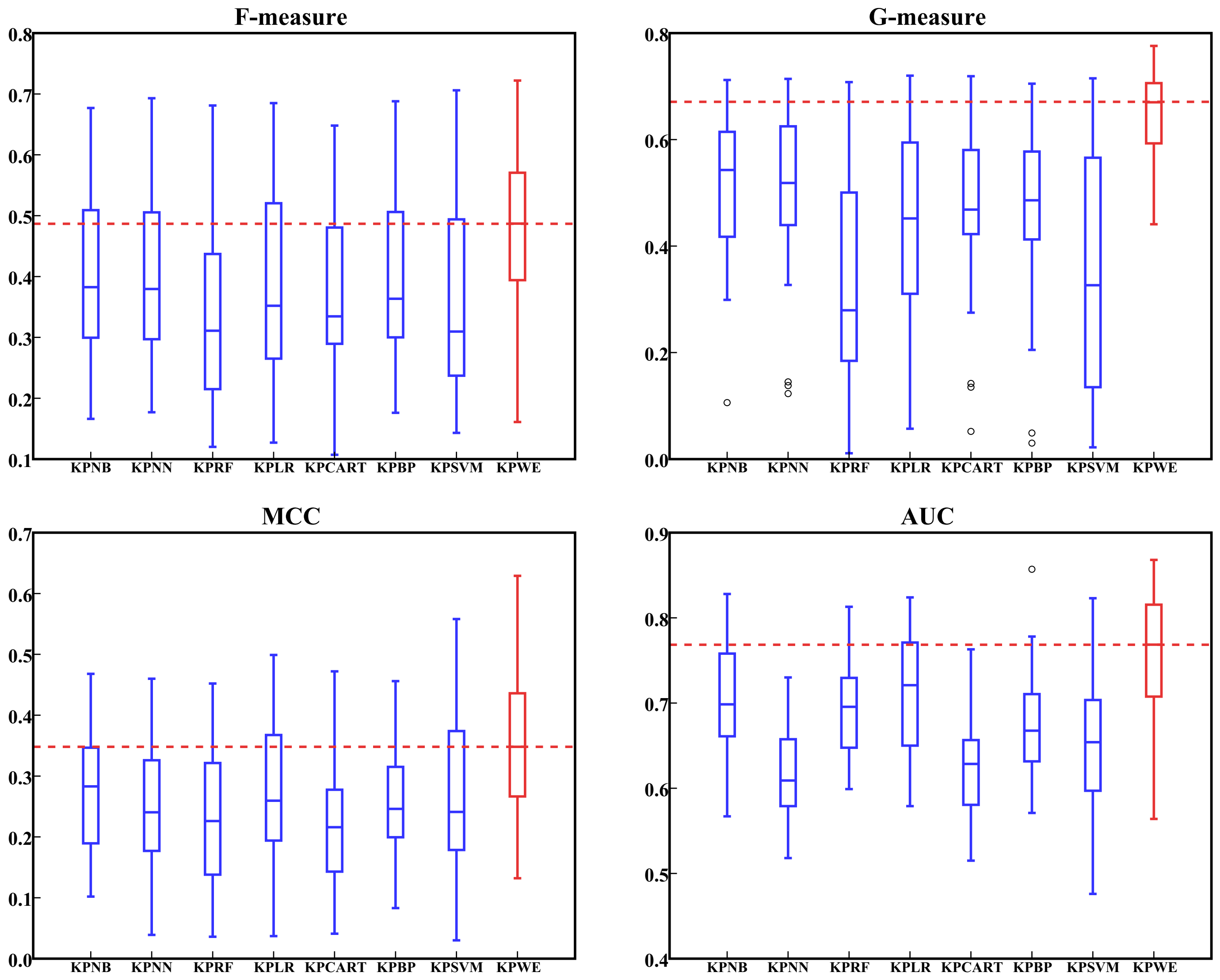


Fig. 4. Box-plots of four indicators for KPWE and seven basic classifiers with KPCA across all 44 projects.

Table 11
Average indicator values of KPWE and its five variants with WELM on two datasets and across all projects.

Dataset	Indicator	ELM	PCAELM	KPCAELM	WELM	PCAWELM	KPWE
PROMISE	F-measure	0.382	0.388	0.467	0.374	0.385	**0.500**
	G-measure	0.470	0.486	0.567	0.556	0.571	**0.660**
	MCC	0.174	0.183	0.342	0.182	0.200	**0.374**
	AUC	0.617	0.624	0.702	0.629	0.639	**0.745**
NASA	F-measure	0.322	0.324	0.365	0.330	0.333	**0.410**
	G-measure	0.458	0.451	0.475	0.550	0.550	**0.611**
	MCC	0.164	0.164	0.263	0.184	0.188	**0.296**
	AUC	0.612	0.611	0.679	0.626	0.629	**0.754**
ALL	F-measure	0.369	0.374	0.444	0.364	0.373	**0.480**
	G-measure	0.468	0.478	0.546	0.555	0.566	**0.649**
	MCC	0.172	0.179	0.324	0.183	0.197	**0.356**
	AUC	0.616	0.621	0.697	0.628	0.637	**0.747**

menyi's post-hoc test is performed to check which specific method differs significantly [33]. For each pair of methods, this test uses the average rank of each method and checks whether the rank difference exceeds a *Critical Difference (CD)* which is calculated with the following formula:

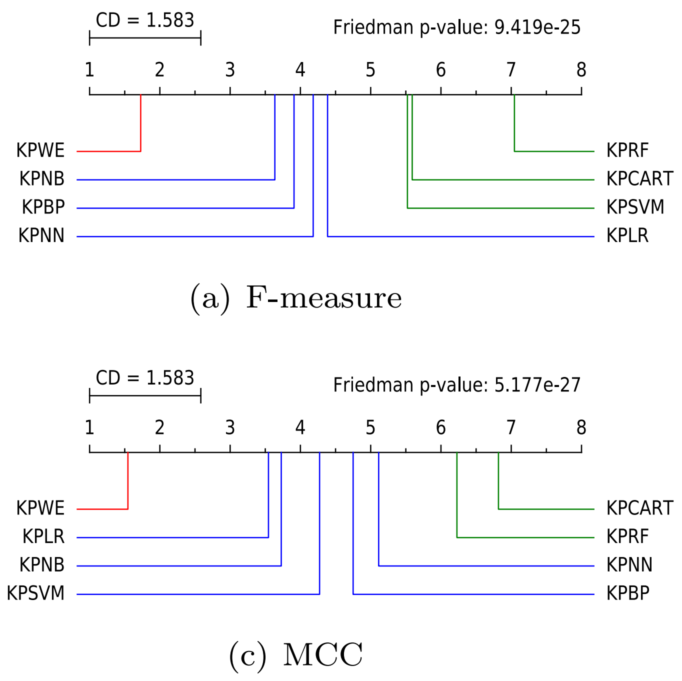
$$CD = q_{\alpha,L}\sqrt{\frac{L(L+1)}{6N}}, \quad (35)$$

where $q_{\alpha,\,L}$ is a critical value that related to the number of methods L and the significance level α. The critical values are available online.[3] The Frideman test with the Nemenyi's post-hoc test is widely used in previous studies [33,81,83–88].

[3] http://www.cin.ufpe.br/~fatc/AM/Nemenyi_critval.pdf.

Table 12
Average indicator values of KPWE and eight feature selection methods with WELM on two datasets and across all projects.

Dataset	Indicator	CS	FS	IG	ReF	NBWrap	NNWrap	LRWrap	RFWrap	KPWE
PROMISE	F-measure	0.347	0.415	0.349	0.415	0.427	0.435	0.425	0.431	**0.500**
	G-measure	0.482	0.574	0.482	0.574	0.588	0.605	0.582	0.597	**0.660**
	MCC	0.139	0.257	0.142	0.255	0.271	0.283	0.271	0.277	**0.374**
	AUC	0.590	0.680	0.588	0.674	0.688	0.692	0.689	0.690	**0.764**
NASA	F-measure	0.297	0.360	0.301	0.366	0.353	0.378	0.365	0.369	**0.410**
	G-measure	0.510	0.568	0.515	0.578	0.573	0.603	0.581	0.591	**0.611**
	MCC	0.152	0.247	0.157	0.243	0.228	0.265	0.242	0.252	**0.296**
	AUC	0.618	0.685	0.606	0.685	0.681	0.688	0.679	0.679	**0.754**
ALL	F-measure	0.336	0.403	0.338	0.404	0.410	0.422	0.411	0.417	**0.480**
	G-measure	0.488	0.572	0.490	0.575	0.585	0.604	0.582	0.595	**0.649**
	MCC	0.142	0.255	0.145	0.252	0.261	0.279	0.265	0.271	**0.356**
	AUC	0.596	0.681	0.592	0.676	0.686	0.691	0.687	0.688	**0.761**



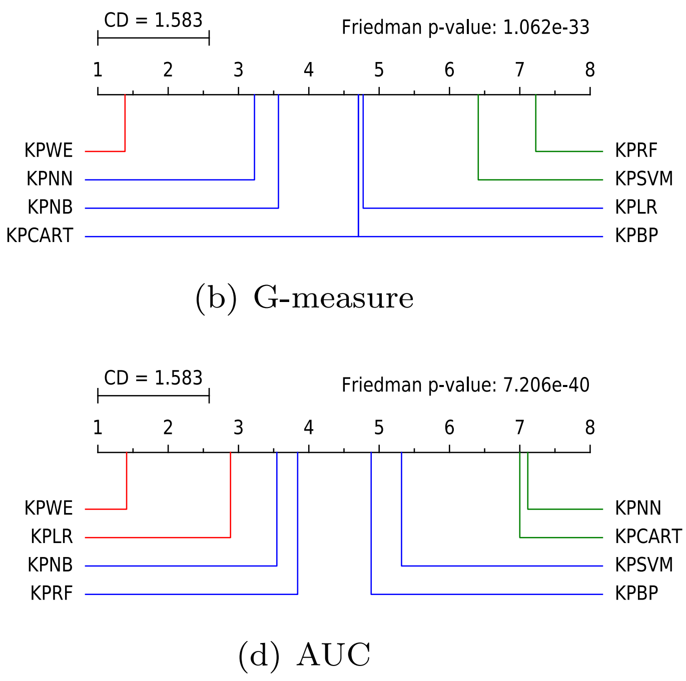


Fig. 5. Comparison of KPWE against seven basic classifiers with KPCA using Friedman test and Nemenyi's post-hoc test in terms of four indicators.

However, the main drawback of post-hoc Nemenyi test is that it may generate overlapping groups for the methods that are compared, not completely distinct groups, which means that a method may belong to multiple significantly different groups [44,88]. In this work, we utilize the strategy in [88] to address this issue. More specifically, under the assumption that the distance (i.e., the difference between two average ranks) between the best average rank and the worst rank is 2 times larger than CD value, we divide the methods into three non-overlapping groups: (1) The method whose distance to the best average rank is less than CD belongs to the top rank group; (2) The method whose distance to the worst average rank is less than CD belongs to the bottom rank group; (3) The other methods belong to the middle rank group. And if the distance between the best average rank and the worst rank is larger than 1 time but less than 2 times CD value, we divide the methods into 2 non-overlapping groups: The method belongs to the top rank group (or bottom rank group) if its average rank is closer to the best average rank (or the worst average rank). In addition, if the distance between the best average rank and the worst rank is less than CD value, all methods belong to the same group. Using this strategy, the generating groups are non-overlapping significantly different.

5. Performance evaluation

5.1. Answer to RQ1: the efficiency of ELM, WELM and some classic classifiers.

Since many previous defect prediction studies applied classic classifiers as prediction models [33,44], in this work, we choose seven representative classifiers, including *Naive Bayes (NB), Nearest Neighbor (NN), Random Forest (RF), Logistic Regression (LR), Classification and Regression Tree (CART), Back Propagation neural networks (BP)* and *Support Vector Machine (SVM)*, and compare their efficiency with ELM and WELM.

The parameter settings of the classifiers are detailed as follows. For NB, we use the kernel estimator that achieves better F-measure values on most projects through our extensive experiments. For RF, we set the number of generated trees to 10, the number of variables for random feature selection to 2, and do not limit the maximum depth of the trees, as suggested in [11]. BP is implemented using the neural networks toolbox in MATLAB with a three-layered and fully-connected network architecture. The learning rate is initialized to 0.1. Since how to select an optimal number of hidden nodes is still an open question [89], we conduct extensive experiments on the benchmark dataset and find that BP

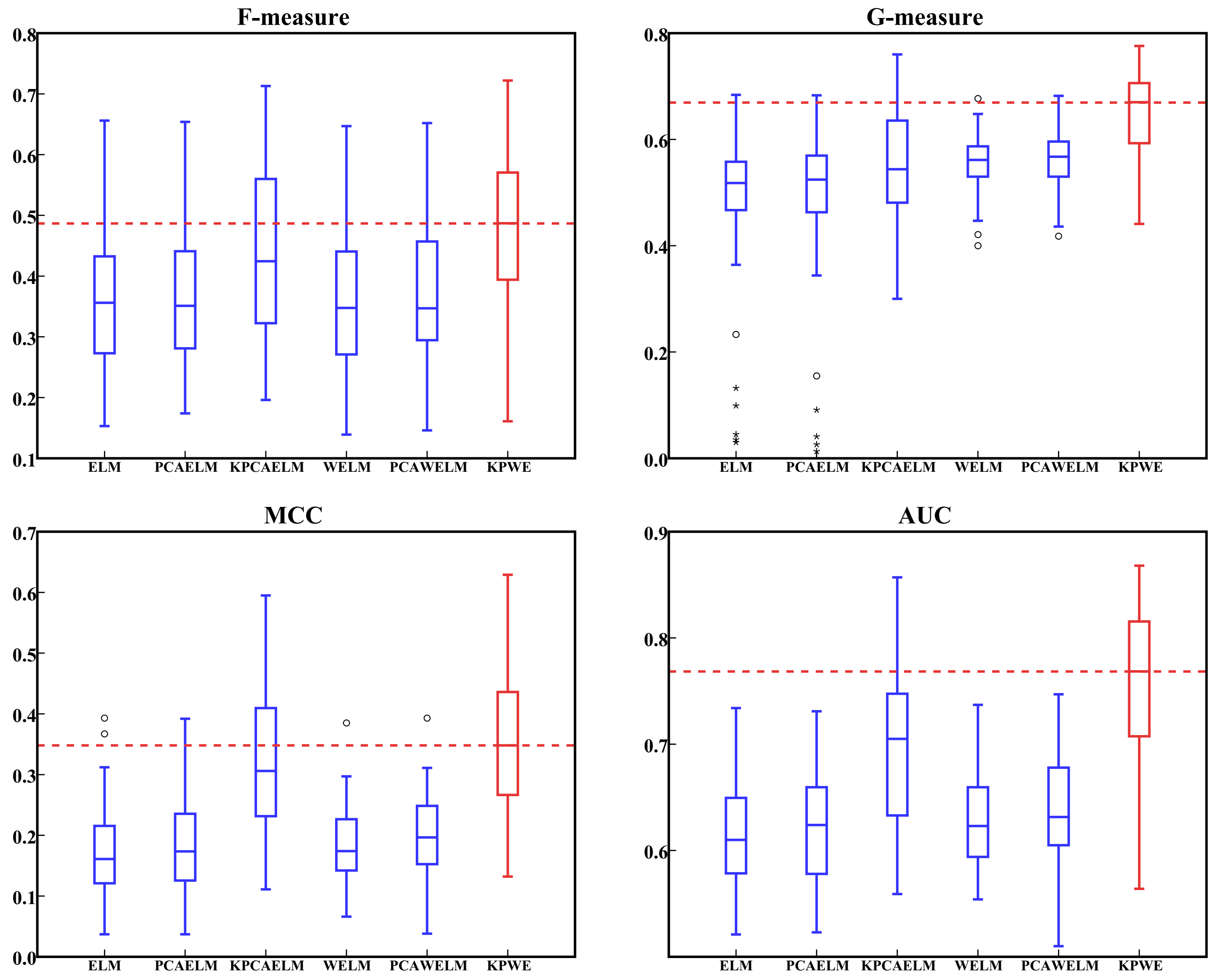


Fig. 6. Box-plots of four indicators for KPWE and its variants on NASA dataset.

can achieve the best F-measure with less than 80 hidden nodes on the vast majority of the projects. Thus we set the number of hidden nodes from 5 to 80 with an increment of 5. The algorithm terminates when the number of iterations is above 2000 or the tolerant error is below 0.004. Other network parameters is set with the default values. The optimal number of hidden nodes is determined based on the best F-measure.

For SVM, we also choose the Gaussian RBF as the kernel function, and set the kernel parameter $\omega_{SVM} = 2^{-10}, 2^{-9}, \dots, 2^{4}$ while cost parameter $C = 2^{-2}, 2^{-1}, \dots, 2^{12}$ as suggested in [90]. Similarly, the optimal parameter combination is obtained according to the best performance through the grid search. For other classifiers, we use the default parameter values. Table 7 tabulates the parameter setting of the seven basic classifiers. The experiments are conducted on a workstation with a 3.60 GHz Intel i7-4790 CPU and 8.00 GB RAM.

Since NN is a lazy classifier that does not need to build a model with the training set in advance, it has no training time [91]. Tables 8 and 9 present the training times of ELM, WELM and the baseline classifiers on PROMISE dataset and NASA dataset, respectively. Note that the value 0 means the training time of the classifier is less than 0.0005 s. For the project with multiple versions, we only report the average training time across the versions. From Table 8, we observe that, on PROMISE dataset, the training time of WELM, less than 0.01 s on 14 projects, is lower than the baseline classifiers on most projects. More specifically, the training time of NB, RF, LR, and CART, less than 0.3 s, is a little bit longer than that of ELM and WELM except for the time of RF on project lucene and poi, while the training time of ELM and WELM are much shorter than that of BP and SVM. In particular, WELM runs nearly 200 (for poi) to 30,000 (for velocity) times faster than BP while 600 (for arc) to 8500 (for velocity) times faster than SVM. The training time between ELM and WELM has a slight difference. From Table 9, we find that, on NASA dataset, WELM takes less than 0.1 seconds to finish training a model on 9 projects. ELM and WELM run faster than the six classifiers except for NB on CM1 project. Particularly, WELM runs 50 (for MC2) to 2700 (for MW1) times faster than BP while 100 (for KC3) to 17,000 (for KC1) times faster than SVM.

Discussion: The short training time of ELM and WELM is due to the following reasons. First, the weights of the input layer and the bias of the hidden layer in ELM are randomly assigned without iterative learning. Second, the weights of the output layer are solved by an inverse operation without iteration. They empower ELM to train the model quickly. Since WELM only adds one step for assigning different weights to the defective and non-defective modules when building the model, it introduces little additional computation cost. Therefore, the training time of ELM and that of WELM are very similar. The superiority of the training

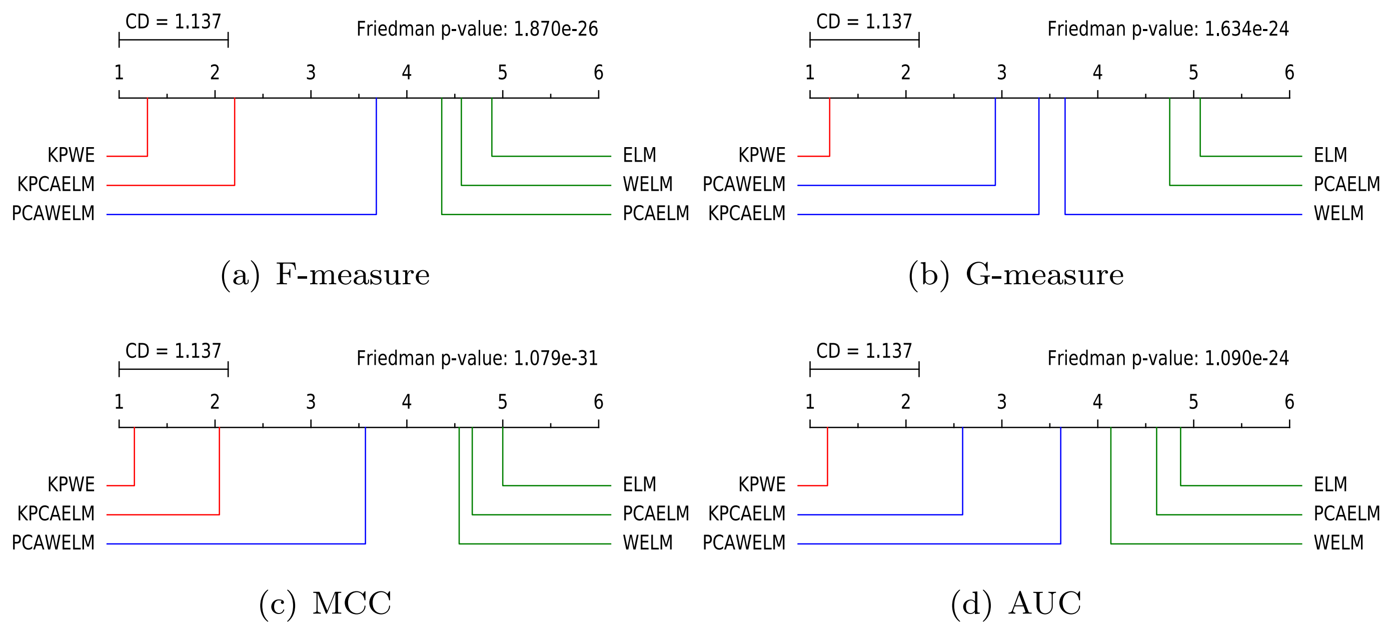


Fig. 7. Comparison of KPWE against its five variants with Friedman test and Nemenyi's post-hoc test in terms of four indicators.

speed of ELM and WELM will be more significant when they are applied to larger datasets.

Summary: Compared with the basic classifiers, ELM and WELM are more efficient to train the prediction model, especially towards BP and SVM, whereas the differences of the efficiency between ELM, WELM and other classifiers are small.

5.2. *Answer to RQ2: the prediction performance of KPWE and the basic classifiers with KPCA.*

Table 10 presents the average indicator values of KPWE and the seven baseline methods on PROMISE dataset, NASA dataset, and across all 44 projects of the two datasets. Fig. 4 depicts the box-plots of four indicators for the eight methods across all 44 projects. The detailed results, including the optimal kernel parameter, the number of hidden nodes, the performance value for each indicator on each project and the corresponding standard deviation for all research questions are available on our online supplementary materials.[4] From Table 10 and Fig. 4, we have the following observations.

First, from Table 10, the results show that our method KPWE achieves the best average performance in terms of all indicators on two datasets and across all 44 projects. More specifically, across all 44 projects, the average F-measure value (0.480) by KPWE yields improvements between 17.1% (for KPNB) and 41.2% (for KPRF) with an average improvement of 25.1%, the average G-measure value (0.649) by KPWE gains improvements between 26.5% (for KPNB) and 92.0% (for KPRF) with an average improvement of 50.4%, the average MCC value (0.356) by KPWE achieves improvements between 27.6% (for KPLR) and 67.9% (for KPCART) with an average improvement of 42.2%, and the average AUC value (0.761) gets improvements between 6.9% (for KPLR) and 23.1% (for KPNN) with an average improvement of 14.2% compared against the seven classic classifiers with KPCA.

Second, Fig. 4 demonstrates that the median values of all four indicators by KPWE are superior to that by the seven baseline methods across all 44 projects. In particular, the median AUC by KPWE is even higher than or similar to the maximum AUC by KPNN, KPCART, and KPBP.

[4] https://sites.google.com/site/istkpwe.

Third, Fig. 5 visualizes the results of the Friedman test with Nemenyi's post-hoc test for KPWE and the seven baseline methods in terms of the four indicators. Groups of the methods that are significantly different are with different colors. The results of the Friedman test show that the *p* values are all less than 0.05, which means that there exist significant differences among the eight methods in terms of all four indicators. The results of the post-hoc test show that KPWE always belongs to the top rank group in terms of all indicators. In addition, KPLR belongs to the top rank group in terms of AUC. These observations indicate that KPWE performs significantly better than the seven baseline methods expect for the KPLR method in terms of AUC.

Discussion: Among all the methods that build prediction models with the features extracted by KPCA, KPWE outperforms the baseline methods because it uses an advanced classifier that considers the class imbalance in the defect data while traditional classifiers could not well copy with the imbalanced data.

Summary: Our method KPWE performs better than KPCA with the seven basic classifiers. On average, compared with the seven baseline methods, KPWE achieves 24.2%, 47.3%, 44.3%, 14.4% performance improvement in terms of the four indicators respectively over PROMISE dataset, 28.1%, 63.6%, 35.6%, 14.2% performance improvement in terms of the four indicators respectively over NASA dataset, and 25.1%, 50.4%, 42.2%, 14.2% performance improvement in terms of the four indicators respectively across all 44 projects.

5.3. *Answer to RQ3: the prediction performance of KPWE and its variants.*

Table 11 presents the average indicator values of KPWE and its five variants on PROMISE dataset, NASA dataset, and across all 44 projects of the two datasets. Fig. 6 depicts the box-plots of four indicators for the six methods across all 44 projects. From Table 11 and Fig. 6, we have the following findings.

First, from Table 11, the results show that our method KPWE achieves the best average performance in terms of all indicators on two datasets and across all 44 projects. More specifically, across all 44 projects, the average F-measure value (0.480) by KPWE yields improvements between 8.1% (for KPCAELM) and 31.9% (for WELM) with an average improvement of 25.4%, the average G-measure value (0.649) by KPWE gains improvements between 14.7% (for PCAWELM) and 38.7%

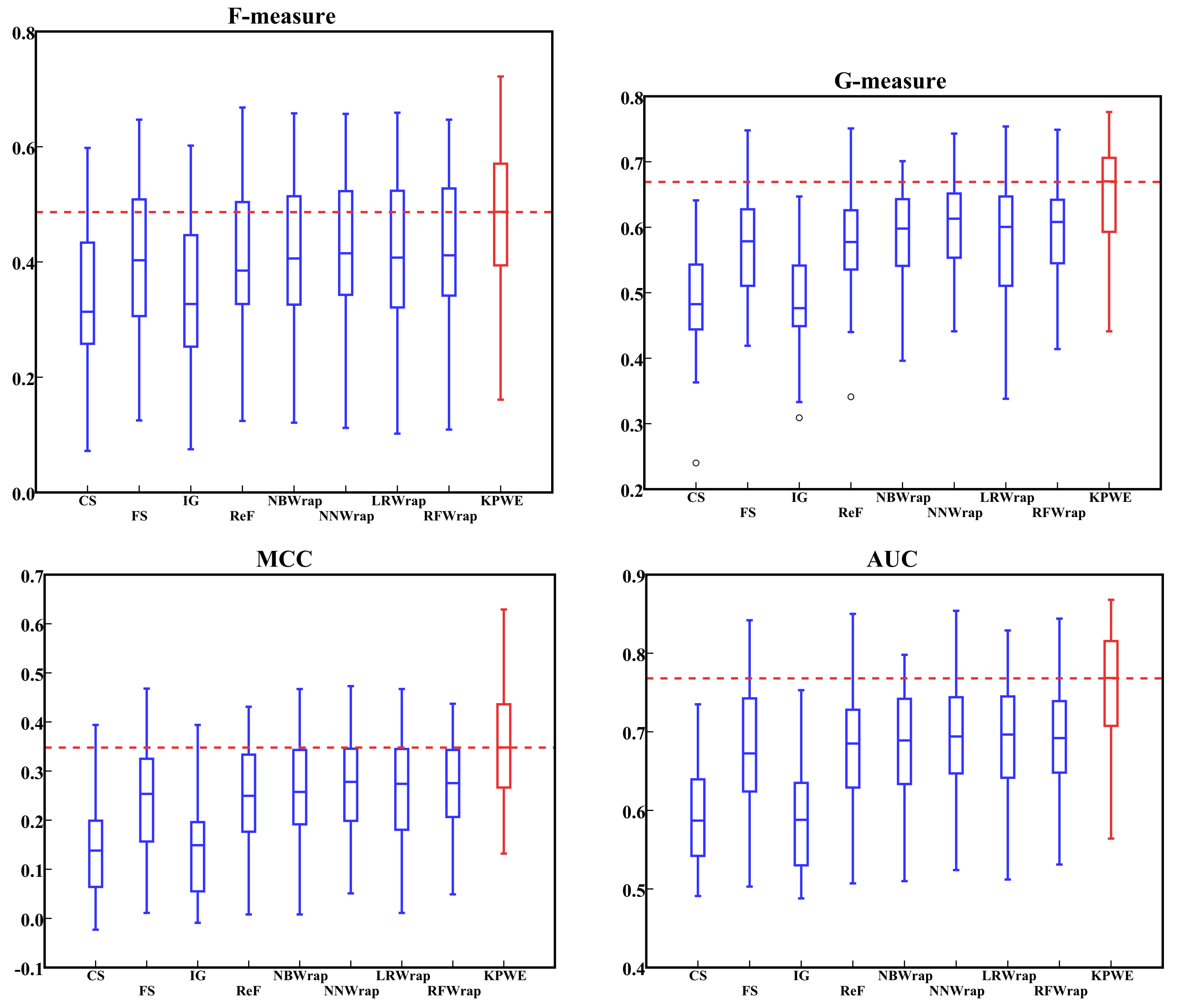


Fig. 8. Box-plots of four indicators for KPWE and eight feature selection methods with WELM across all 44 projects.

(for ELM) with an average improvement of 25.0%, the average MCC value (0.356) by KPWE achieves improvements between 9.9% (for KPCAELM) and 107.0% (for ELM) with an average improvement of 78.2%, and the average AUC value (0.761) gets improvements between 9.2% (for KPCAELM) and 23.5% (for ELM) with an average improvement of 19.2% compared with the five variants.

Second, Fig. 6 shows that KPWE outperforms the five variants in terms of the median values of all indicators across all 44 projects. In particular, the median G-measure by KPWE is higher than or similar to the maximum G-measure (do not consider the noise points) by the baseline methods except for PCAWELM, the median MCC by KPWE is higher than the maximum MCC by ELM, WELM and PCAWELM, and the median AUC by KPWE is higher than the maximum AUC by the baseline methods except for PCAWELM.

Third, Fig. 7 visualizes the results of the Friedman test with Nemenyi's post-hoc test for KPWE and its five variants in terms of the four indicators. The *p* values of the Friedman test are all less than 0.05, which means that there exist significant differences among the six methods in terms of all four indicators. The results of the post-hoc test show that KPWE also always belongs to the top rank 1 group in terms of all indicators. In addition, KPCAELM belong to the top rank 1 group in terms of F-measure and MCC. These observations indicate that in terms of G-measure and AUC, KPWE significantly performs better than the five variants, whereas in terms of F-measure and MCC, KPWE does not perform significantly better than KPCAELM.

Discussion: On the one hand, KPWE and KPCAELM are superior to PCAWELM and PCAELM in terms of all four indicators respectively, on the other hand, KPWE and KPCAELM perform better than WELM and ELM, respectively on both datasets, all these mean that the features extracted by the nonlinear method KPCA are beneficial to ELM and WELM for the improvement of defect prediction performance compared against the raw features or the features extracted by linear method PCA. Moreover, KPWE, PCAWELM and WELM are superior to KPCAELM, PCAELM and ELM respectively which denotes that WELM is more appropriate to the class imbalanced defect data than ELM.

Summary: KPWE precedes its five variants. On average, compared with the five downgraded variants, KPWE achieves 26.1%, 25.4%, 84.2%, 19.2% performance improvement in terms of the four indicators respectively over PROMISE dataset, 22.7%, 23.9%, 58.4%,

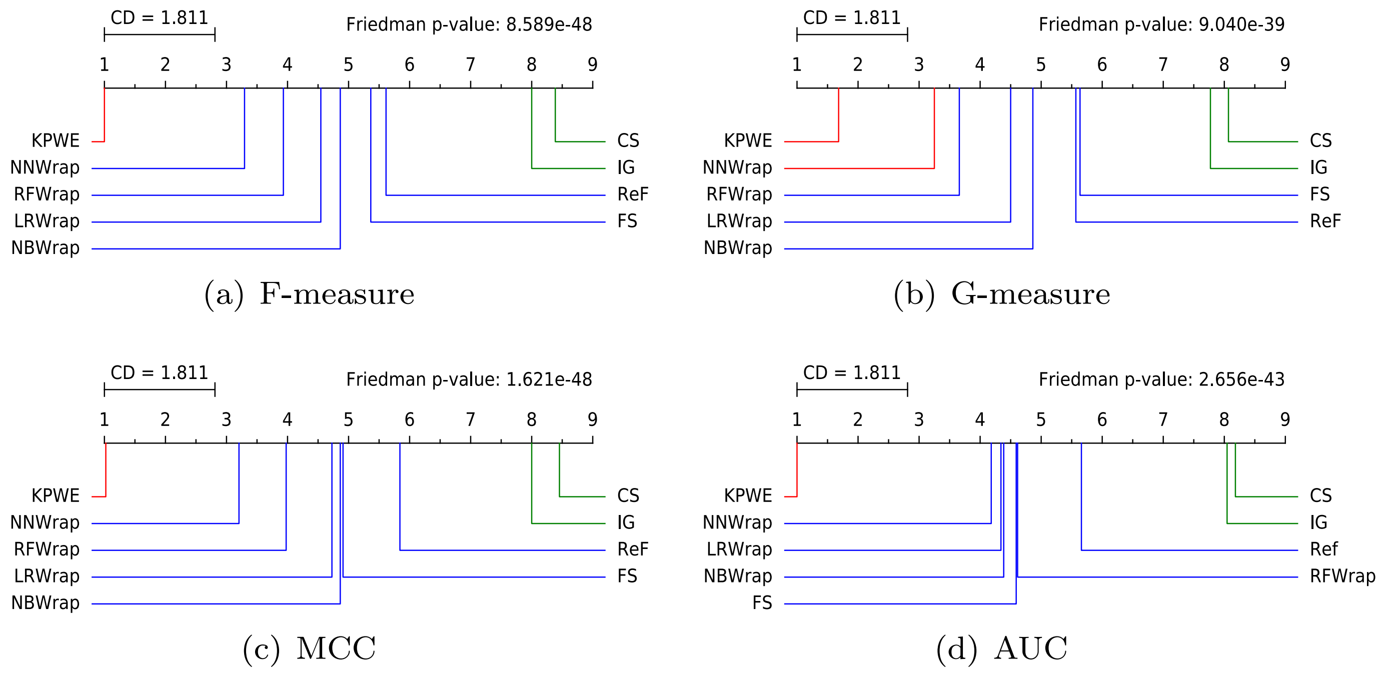


Fig. 9. Comparison of KPWE against the eight feature selection based baseline methods with Friedman test and Nemenyi's post-hoc test in terms of four indicators.

19.6% performance improvement in terms of the four indicators respectively over NASA dataset, and 25.4%, 25.0%, 78.2%, 19.2% performance improvement in terms of the four indicators respectively across all 44 projects.

5.4. Answer to RQ4: the prediction performance of KPWE and other feature selection methods with WELM.

Here, we choose eight representative feature selection methods, include four filter-based feature ranking methods and four wrapper-based feature subset selection methods, for comparison. The filter-based methods are *Chi-Square (CS), Fish Score (FS), Information Gain (IG)* and *ReliefF (ReF)*. The first two methods are both based on statistics, the last two are based on entropy and instance, respectively. These methods have been proven to be effective for defect prediction [19,92]. For wrapper-based methods, we choose four commonly-used classifiers (i.e., NB, NN, LR, and RF) and F-measure to evaluate the performance of the selected feature subset. The four wrapper methods are abbreviated as NBWrap, NNWrap, LRWrap, and RFWrap. Following the previous work [19,38], we set the number of selected features to $\lceil log_2 m \rceil$, where m is the number of original features.

Table 12 presents the average indicator values of KPWE and eight feature selection methods with WELM on PROMISE dataset, NASA dataset, and across all 44 projects of the two datasets. Fig. 8 depicts the box-plots of four indicators for the nine methods across all 44 projects. Some findings are observed from Table 12 and Fig. 8 as follows.

First, from Table 12, the results show that our method KPWE achieves the best average performance in terms of all indicators on two datasets and across all 44 projects. More specifically, across all 44 projects, the average F-measure value (0.480) by KPWE yields improvements between 13.7% (for NNWrap) and 42.9% (for CS) with an average improvement of 23.2%, the average G-measure value (0.649) by KPWE gains improvements between 7.5% (for NNWrap) and 33.0% (for CS) with an average improvement of 16.4%, the average MCC value (0.356) by KPWE achieves improvements between 27.6% (for NNWrap) and 150.7% (for CS) with an average improvement of 63.4%, and the average AUC value (0.761) gets improvements between 10.1% (for NNWrap) and 27.7% (for CS) with an average improvement of 15.4% compared with eight feature selection methods with WELM.

Second, Fig. 8 manifests that superiority of KPWE compared with the eight baseline methods in terms of the median values of all four indicators across all 44 projects. In particular, the median AUC by KPWE is higher than the maximum AUC by CS and IG. In addition, we can also observe that the performance of the four wrapper-based feature subset selection methods are generally better than the filter-based feature subset selection methods, which is consistent with the observation in previous study [19].

Third, Fig. 9 visualizes the results of the Friedman test with Nemenyi's post-hoc test for KPWE and the eight feature selection based baseline methods in terms of the four indicators. There exist significant differences among the nine methods in terms of all four indicators since the p values of the Friedman test are all less than 0.05. The results of the post-hoc test illustrate that KPWE always belongs to the top rank group in terms of all indicators. In addition, NNWrap belongs to the top rank group in terms of G-measure. These observations show that KPWE performs significantly better than the eight baseline methods expect for the NNWrap method in terms of G-measure.

Discussion: The reason why the features extracted by KPCA are more effective is that, the eight feature selection methods only select a subset of original features that are not able to excavate the important information hidden behind the raw data, whereas KPCA can eliminate the noise in the data and extract the intrinsic structures of the data that are more helpful to distinguish the class labels of the modules.

Summary: KPWE outperforms the eight feature selection methods with WELM. On average, compared with the eight baseline methods, KPWE achieves 24.3%, 18.6%, 71.0%, 16.0% performance improvement in terms of the four indicators respectively over PROMISE dataset, 18.5%, 8.5%, 38.3%, 13.7% performance improvement in terms of the four indicators respectively over NASA dataset, and 23.2%, 16.4%, 63.4%, 15.4% performance improvement in terms of the four indicators respectively across all 44 projects.

5.5. Answer to RQ5: the prediction performance of KPWE and other imbalanced learning methods.

Here, we employ 12 classic imbalanced learning methods based on data sampling strategies. These methods first use *Random*

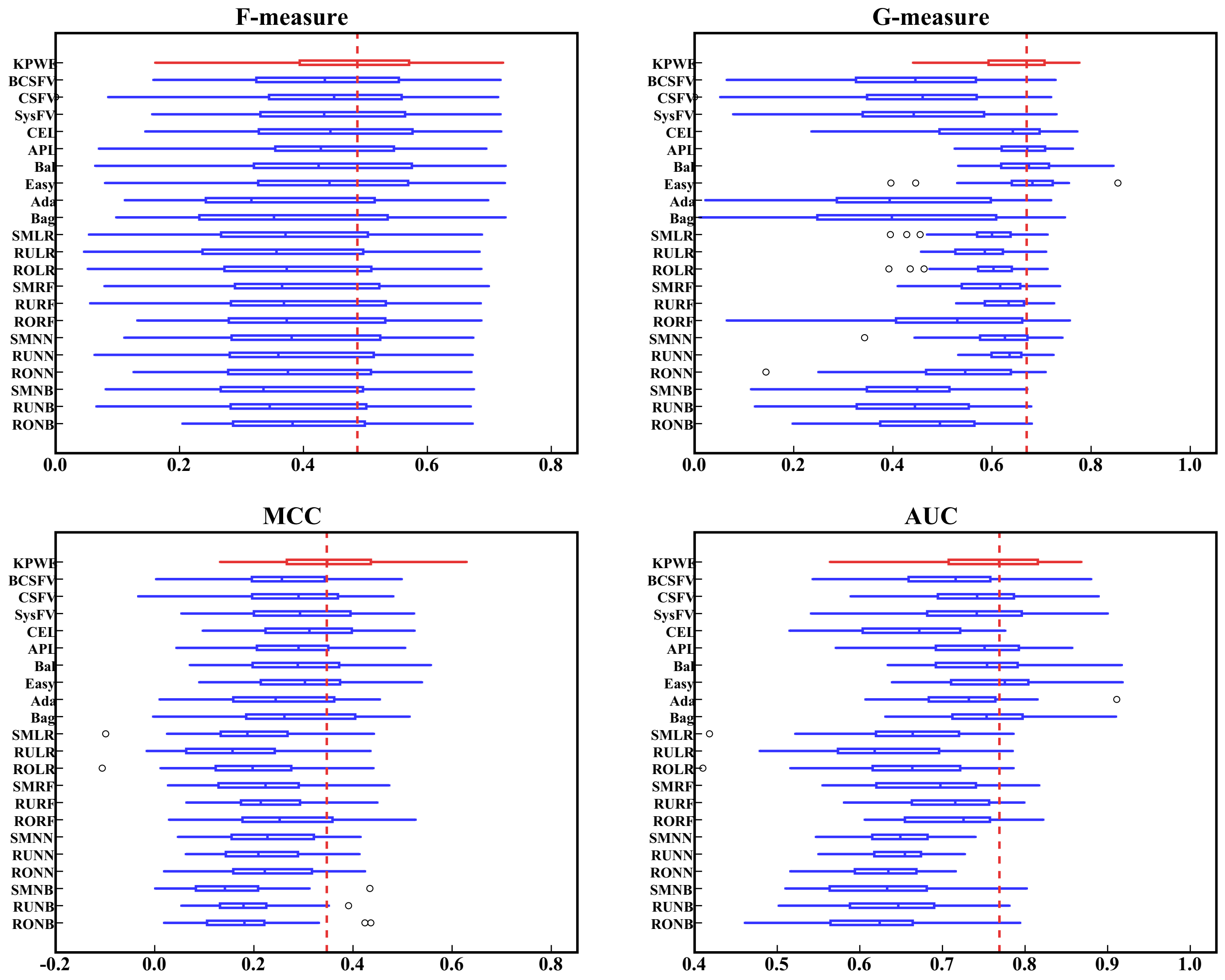


Fig. 10. Box-plots of four indicators for KPWE and 21 class imbalanced learning methods across all 44 projects.

Under-sampling (RU), Random Over-sampling (RO) or *SMOTE (SM)* techniques to rebalance the modules of the two classes in the training set, then, four popular classifiers as the same in RQ4 (i.e., NB, NN, LR, and RF) are applied to the rebalanced training set. The method name is the combination of the abbreviation of the sampling strategy and the used classifier. Also, we employ two widely-used ensemble learning methods (i.e., *Bagging (Bag)* and *Adaboost (Ada)* for comparison. Moreover, we use other seven imbalanced learning methods, *Coding-based Ensemble Learning (CEL)* [93], *Systematically developed Forest with cost-sensitive Voting (SysFV)* [94], *Cost-Sensitive decision Forest with cost-sensitive Voting (CSFV)* [95], *Balanced CSFV (BCSFV)* [57], *Asymmetric Partial Least squares classifier (APL)* [96], *EasyEnsemble (Easy)* [97], and *BalanceCascade (Bal)* [97] as the baseline methods. Note that the last three methods have not yet been applied to defect prediction but have been proved to achieve promising performance for imbalanced data in other domains. Among these method, SysFV, CSFV amd BCSFV are cost-sensitive based imbalanced learning methods, while Easy and Bal combine the sampling strategies and ensemble learning methods.

Table 13 presents the average indicator values of KPWE and the 21 class imbalanced baseline methods on PROMISE dataset, NASA dataset, and across all 44 projects of the two datasets. Fig. 10 depicts the box-plots of four indicators for the 22 methods across all 44 projects. We describe the findings from Table 13 and Fig. 10 as follows.

First, from Table 13, the results show that our method KPWE achieves the best average performance in terms of F-measure and MCC on two datasets and across all 44 projects. More specifically, across all 44 projects, the average F-measure value (0.480) by KPWE yields improvements between 7.6% (for CEL) and 34.5% (for RULR) with an average improvement of 19.6%, the average MCC value (0.356) by KPWE gains improvements between 17.9% (for Easy) and 140.5% (for SMNB) with an average improvement of 56.5%. However, Easy, Bal, APL outperform our method KPWE in terms of average G-measure values and Easy outperforms KPWE in terms of the average AUC values across all 44 projects. Overall, KPWE achieves average improvements of 23.4% and 11.2% over the 21 baseline methods in terms of average G-measure and AUC, respectively.

Second, Fig. 10 depicts that KPWE is superior to the 21 baseline methods in terms of the median F-measure and MCC across all 44 projects. In particular, the median MCC by KPWE is higher than the maximum MCC by RONB and SMNB. In addition, the median G-measure by KPWE is similar to that by APL and Bal, whereas the median G-measure and AUC by KPWE are only a little lower than those by Easy.

Third, Fig. 11 visualizes the results of the Friedman test with Nemenyi's post-hoc test for KPWE and the 21 class imbalanced learning methods in terms of the four indicators. As the p values of the Friedman test are all less than 0.05, there exist significant differences among the

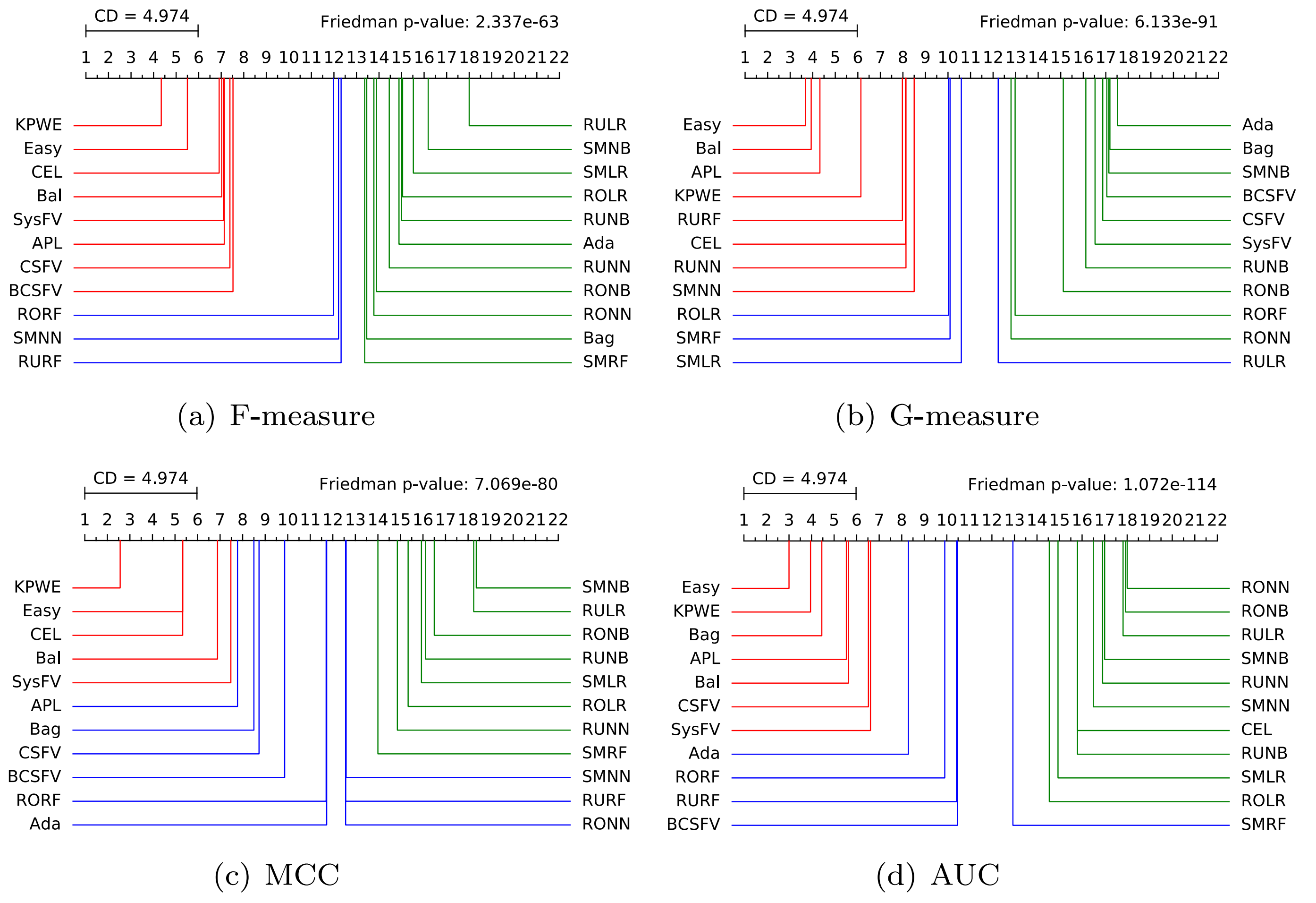


Fig. 11. Comparison of KPWE against the 21 class imbalanced learning methods with Friedman test and Nemenyi's post-hoc test in terms of four indicators.

22 methods in terms of all four indicators. The results of the post-hoc test illustrate that KPWE also belongs to the top rank group in terms of all indicators. However, in terms of F-measure, G-measure MCC and AUC, KPWE does not perform significantly well compared with seven, seven, four and six baseline methods respectively in which the common methods are Easy and Bal. These observations manifest that KPWE, Easy and Bal belong to the top rank group and perform no statistically significant differences with each other in terms of all four indicators. Since this is the first work to investigate the performance of method Easy and methods Bal on software defect data, the experimental results indicate that they are also potentially effective methods for defect prediction as our method KPWE is.

Discussion: The under-sampling methods may neglects the potentially useful information contained in the ignored non-defective modules, and the over-sampling methods may cause the model over-fitting by adding some redundancy defective modules. In addition, data sampling based imbalanced learning methods usually change the data distribution of the defect data. From this point, the cost-sensitive learning methods (such as our KPWE method) which does not change the data distribution are better choices for imbalanced defect data. Considering the main drawback of under-sampling methods, Easy and Bal sample multiple subsets from the majority class and then use each of these subsets to train an ensemble. Finally, they combine all weak classifiers of these ensembles into a final output [97]. The two methods can wisely explore these ignored modules, which enable them to perform well on the imbalanced data.

Summary: KPWE performs better than the 21 baseline methods especially in terms of F-measure and MCC. On average, compared with the baseline methods, KPWE achieves 19.1%, 23.9%, 57.7%, 11.3% performance improvement in terms of the four indicators respectively over PROMISE dataset, 21.0%, 23.2%, 53.2%, 11.4% performance improvement in terms of the four indicators respectively over NASA dataset, and 19.6%, 23.4%, 56.5%, 11.2% performance improvement in terms of the four indicators respectively across all 44 projects. In addition, KPWE performs no statistically significant differences compared with Easy and Bal across all 44 projects in terms of all four indicators.

6. Threats to validity

6.1. External validity

External validity focuses on whether our experimental conclusions will vary on different projects. We conduct experiments on total 44 projects of two defect datasets to reduce the threat for this kind of validity. In addition, since the features of our benchmark dataset are all static product metrics and the modules are abstracted at class level (for PROMISE dataset) and component level (for NASA dataset), we cannot claim that our experimental conclusions can be generalized to the defect datasets with process metrics and the modules extracted at file level.

6.2. Internal validity

We implement most baseline methods using the function library of machine learning and toolbox in MATLAB to reduce the potential influence of the incorrect implementations on our experimental results. In addition, we tune the optimal parameter values, such as the width of

Table 13
Average indicator values of KPWE and 21 class imbalanced learning methods with WELM on two datasets and across all projects.

Dataset	Indicator	RONB	RUNB	SMNB	RONN	RUNN	SMNN	RORF	RURF	SMRF	ROLR	RULR	SMLR	Bag	Ada	Easy	Bal	APL	CEL	SysFV	CSFV	BCSFV	KPWE
PROMISE	F-measure	0.416	0.400	0.385	0.414	0.407	0.415	0.431	0.421	0.408	0.395	0.375	0.393	0.409	0.397	0.458	0.450	0.447	0.467	0.455	0.456	0.455	**0.500**
	G-measure	0.476	0.416	0.422	0.546	0.633	0.613	0.534	0.639	0.598	0.598	0.586	0.595	0.429	0.443	0.664	**0.665**	0.660	0.610	0.455	0.459	0.454	0.660
	MCC	0.187	0.181	0.152	0.247	0.228	0.242	0.276	0.256	0.228	0.203	0.171	0.200	0.298	0.260	0.311	0.298	0.286	0.324	0.294	0.291	0.272	**0.374**
	AUC	0.622	0.634	0.632	0.634	0.647	0.647	0.721	0.715	0.693	0.668	0.636	0.664	0.755	0.724	**0.765**	0.742	0.737	0.668	0.730	0.740	0.707	0.764
NASA	F-measure	0.333	0.329	0.313	0.347	0.331	0.354	0.308	0.327	0.331	0.326	0.295	0.327	0.281	0.296	0.393	0.389	0.379	0.374	0.389	0.357	0.380	**0.410**
	G-measure	0.486	0.502	0.463	0.525	0.623	0.620	0.410	0.597	0.583	0.599	0.550	0.593	0.303	0.355	0.689	0.669	**0.677**	0.561	0.401	0.384	0.388	0.611
	MCC	0.144	0.174	0.134	0.213	0.187	0.213	0.176	0.180	0.172	0.172	0.117	0.170	0.217	0.215	0.275	0.264	0.255	0.254	0.266	0.247	0.240	**0.296**
	AUC	0.603	0.674	0.629	0.622	0.641	0.645	0.682	0.682	0.656	0.643	0.601	0.651	0.748	0.734	**0.759**	0.748	0.757	0.644	0.736	0.734	0.710	0.754
ALL	F-measure	0.397	0.383	0.369	0.399	0.389	0.401	0.403	0.400	0.391	0.379	0.357	0.378	0.380	0.375	0.443	0.436	0.432	0.446	0.440	0.434	0.438	**0.480**
	G-measure	0.479	0.435	0.431	0.541	0.630	0.615	0.506	0.629	0.594	0.598	0.578	0.595	0.400	0.423	**0.669**	0.666	0.664	0.599	0.443	0.442	0.439	0.649
	MCC	0.177	0.180	0.148	0.239	0.218	0.235	0.253	0.239	0.215	0.196	0.159	0.193	0.279	0.250	0.302	0.290	0.279	0.308	0.288	0.281	0.265	**0.356**
	AUC	0.617	0.643	0.632	0.631	0.646	0.646	0.712	0.708	0.685	0.662	0.628	0.661	0.754	0.726	**0.764**	0.744	0.741	0.662	0.732	0.738	0.708	0.761

kernel parameter in KPCA and the number of hidden nodes in WELM, from a relatively wide range of tested options. Nevertheless, a more carefully controlled experiment for the parameter selection should be considered.

6.3. *Construct validity*

Although we employ four extensively-used indicators to evaluate the performances of KPWE and the baseline methods for defect prediction, these indicators do not take the effort of inspecting cost into consideration. We will use the effect-aware indicators to evaluate the effectiveness of our method in future work.

6.4. *Conclusion validity*

We use a state-of-the-art double Scott-Knott ESD method to check whether the differences between KPWE and the baseline methods are significant. With this statistic test, the assessment towards the superiority of KPWE is more rigorous.

7. Conclusion

In this work, we propose a new defect prediction framework KPWE that comprises feature extraction stage and model construction stage. In the first stage, to handle the complex structures in defect data, we learn the representative features by mapping the original data into a latent feature space with a nonlinear feature extraction method KPCA. The mapped features in the new space can better represent the raw data. In the second stage, we construct a class imbalanced classifier on the extracted features by introducing a state-of-the-art learning algorithm WELM. Besides the advantages of fine generalization ability and less prone to local optimum, WELM strengthens the impact of defective modules by assigning them higher weights. We have carefully evaluated KPWE on 34 projects from PROMISE dataset and 10 projects from NASA dataset with four indicators. The experimental results show that KPWE exhibits superiority over 41 baselines methods, especially in terms of F-measure, MCC and AUC.

In future work, we will provide guidelines on deciding the optimal number of hidden nodes and kernel parameter values for KPWE, as they vary for different projects. In addition, we plan to explore the impact of the different kernel functions in KPCA and the different activation functions in WELM on the performance of KPWE.

Acknowledgments

The authors would like to acknowledge the support provided by the grands of the National Natural Science Foundation of China (61572374, U1636220, 61472423, 61602258), Open Fund of Key Laboratory of Network Assessment Technology from CAS, the Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012), the National Science Foundation (DGE-1522883), Hong Kong RGC Project (CityU C1008-16G), Hong Kong General Research Fund (PolyU 152279/16E, 152223/17E), and the China Postdoctoral Science Foundation (2017M621247).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2018.10.004.

References

[1] J. Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, John Wiley & Sons, 2005.
[2] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, John Wiley & Sons, 2011.
[3] M. Shepperd, D. Bowes, T. Hall, Researcher bias: the use of machine learning in software defect prediction, IEEE Trans. Softw. Eng. (TSE) 40 (6) (2014) 603–616.

[4] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Trans. Softw. Eng. (TSE) 37 (3) (2011) 356–370.
[5] X. Yang, K. Tang, X. Yao, A learning-to-rank approach to software defect prediction, IEEE Trans. Reliab. 64 (1) (2015) 234–246.
[6] P. Knab, M. Pinzger, A. Bernstein, Predicting defect densities in source code files with decision tree learners, in: Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR), ACM, 2006, pp. 119–125.
[7] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. (TSE) 33 (1) (2007) 2–13.
[8] L. Guo, Y. Ma, B. Cukic, H. Singh, Robust prediction of fault-proneness by random forests, in: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2004, pp. 417–428.
[9] C. Macho, S. McIntosh, M. Pinzger, Predicting build co-changes with source code change and commit categories, in: Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 1, IEEE, 2016, pp. 541–551.
[10] X. Jing, F. Wu, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning, in: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE), ACM, 2015, pp. 496–507.
[11] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, J. Syst. Softw. (JSS) 81 (5) (2008) 649–660.
[12] Z. Yan, X. Chen, P. Guo, Software defect prediction using fuzzy support vector regression, Adv. Neural Netw. (2010) 17–24.
[13] M.M.T. Thwin, T.-S. Quah, Application of neural networks for software quality prediction using object-oriented metrics, J. Syst. Softw. (JSS) 76 (2) (2005) 147–156.
[14] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, S.J. Aud, Application of neural networks to software quality modeling of a very large telecommunications system, IEEE Trans. Neural Netw. (TNN) 8 (4) (1997) 902–909.
[15] D.E. Neumann, An enhanced neural network technique for software risk analysis, IEEE Trans. Soft. Eng. (TSE) 28 (9) (2002) 904–912.
[16] A. Panichella, R. Oliveto, A. De Lucia, Cross-project defect prediction models: L'union fait la force, in: Proceedings of the 21st Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSM-R-WCRE), IEEE, 2014, pp. 164–173.
[17] A. Shanthini, Effect of ensemble methods for software fault prediction at various metrics level, Int. J. Appl. Inf. Syst. (2014).
[18] X. Xia, D. Lo, S. McIntosh, E. Shihab, A.E. Hassan, Cross-project build co-change prediction, in: Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2015, pp. 311–320.
[19] Z. Xu, J. Liu, Z. Yang, G. An, X. Jia, The impact of feature selection on defect prediction performance: an empirical comparison, in: Proceedings of the 27th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2016, pp. 309–320.
[20] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemom. Intell. Lab. Syst. 2 (1–3) (1987) 37–52.
[21] Q. Song, J. Ni, G. Wang, A fast clustering-based feature subset selection algorithm for high-dimensional data, IEEE Trans. Knowl. Data Eng. (TKDE) 25 (1) (2013) 1–14.
[22] T. Wang, Z. Zhang, X. Jing, L. Zhang, Multiple kernel ensemble learning for software defect prediction, Autom. Softw. Eng. (ASE) 23 (4) (2016) 569–590.
[23] F. Liu, X. Gao, B. Zhou, J. Deng, Software defect prediction model based on PCA-isvm, Comput. Simulat. (2014).
[24] H. Cao, Z. Qin, T. Feng, A novel PCA-bp fuzzy neural network model for software defect prediction, Adv. Sci. Lett. 9 (1) (2012) 423–428.
[25] C. Zhong, Software quality prediction method with hybrid applying principal components analysis and wavelet neural network and genetic algorithm, Int. J. Digital Content Technol. Appl. 5 (3) (2011).
[26] T.M. Khoshgoftaar, R. Shan, E.B. Allen, Improving tree-based models of software quality with principal components analysis, in: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2000, pp. 198–209.
[27] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: some comments on the nasa software defect datasets, IEEE Trans. Softw. Eng. (TSE) 39 (9) (2013) 1208–1215.
[28] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, The misuse of the nasa metrics data program data sets for automated software defect prediction, in: Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE), IET, 2011, pp. 96–103.
[29] T. Menzies, K. Ammar, A. Nikora, J. DiStefano, How simple is software defect detection, Submitt. Emprical Softw. Eng. J. (2003).
[30] B. Schölkopf, A. Smola, K.-R. Müller, Kernel principal component analysis, in: Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN), Springer, 1997, pp. 583–588.
[31] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Comput. 10 (5) (1998) 1299–1319.
[32] K.I. Kim, M.O. Franz, B. Scholkopf, Iterative kernel principal component analysis for image modeling, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 27 (9) (2005) 1351–1366.
[33] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. (TSE) 34 (4) (2008) 485–496.
[34] W. Zong, G. Huang, Y. Chen, Weighted extreme learning machine for imbalance learning, Neurocomputing 101 (2013) 229–242.
[35] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.
[36] S. Shivaji, E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve code change-based bug prediction, IEEE Trans. Softw. Eng. (TSE) 39 (4) (2013) 552–569.
[37] S. Shivaji, J.E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve bug prediction, in: Proceedings of the 24th International Conference on Automated Software Engineering (ASE), IEEE Computer Society, 2009, pp. 600–604.
[38] K. Gao, T.M. Khoshgoftaar, H. Wang, N. Seliya, Choosing software metrics for defect prediction: an investigation on feature selection techniques, Softw. Pract. Exp. (SPE) 41 (5) (2011) 579–606.
[39] X. Chen, Y. Shen, Z. Cui, X. Ju, Applying feature selection to software defect prediction using multi-objective optimization, in: Proceedings of the 41st Annual Computer Software and Applications Conference (COMPSAC), 2, IEEE, 2017, pp. 54–59.
[40] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Inf. Sci. (Ny) 179 (8) (2009) 1040–1058.
[41] B. Ghotra, S. McIntosh, A.E. Hassan, A large-scale study of the impact of feature selection techniques on defect classification models, in: Proceedings of the 14th International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 146–157.
[42] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Appl. Softw. Comput. 27 (2015) 504–518.
[43] R. Malhotra, An empirical framework for defect prediction using machine learning techniques with android software, Appl. Softw. Comput. 49 (2016) 1034–1050.
[44] B. Ghotra, S. McIntosh, A.E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: Proceedings of the 37th International Conference on Software Engineering (ICSE), IEEE Press, 2015, pp. 789–800.
[45] R. Malhotra, R. Raje, An empirical comparison of machine learning techniques for software defect prediction, in: Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 320–327.
[46] J. Ren, K. Qin, Y. Ma, G. Luo, On software defect prediction using machine learning, J. Appl. Math. 2014 (2014).
[47] G. Luo, H. Chen, Kernel based asymmetric learning for software defect prediction, IEICE Trans. Inf. Syst. 95 (1) (2012) 267–270.
[48] G. Luo, Y. Ma, K. Qin, Asymmetric learning based on kernel partial least squares for software defect prediction, IEICE Trans. Inf. Syst. 95 (7) (2012) 2006–2008.
[49] D.P. Mesquita, L.S. Rocha, J.P.P. Gomes, A.R.R. Neto, Classification with reject option for software defect prediction, Appl. Softw. Comput. 49 (2016) 1085–1093.
[50] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K.-i. Matsumoto, The effects of over and under sampling on fault-prone module detection, in: Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2007, pp. 196–204.
[51] K.E. Bennin, J. Keung, A. Monden, P. Phannachitta, S. Mensah, The significant effects of data sampling approaches on software defect prioritization and classification, in: Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE Press, 2017, pp. 364–373.
[52] K.E. Bennin, J. Keung, A. Monden, Impact of the distribution parameter of data sampling approaches on software defect prediction models, in: Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2017, pp. 630–635.
[53] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, arXiv:1801.10269 (2018).
[54] T.M. Khoshgoftaar, E. Geleyn, L. Nguyen, L. Bullard, Cost-sensitive boosting in software quality modeling, in: Proceedings of the 7th International Symposium on High Assurance Systems Engineering, IEEE, 2002, pp. 51–60.
[55] J. Zheng, Cost-sensitive boosting neural networks for software defect prediction, Expert Syst. Appl. 37 (6) (2010) 4537–4543.
[56] M. Liu, L. Miao, D. Zhang, Two-stage cost-sensitive learning for software defect prediction, IEEE Trans. Reliab. 63 (2) (2014) 676–686.
[57] M.J. Siers, M.Z. Islam, Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem, Inf. Syst. 51 (2015) 62–71.
[58] J. Peng, D.R. Heisterkamp, Kernel indexing for relevance feedback image retrieval, in: Proceedings of the 10th International Conference on Image Processing (ICIP), 1, IEEE, 2003, pp. I–733.
[59] J. Li, S. Chu, J.-S. Pan, Kernel principal component analysis (Kpca)-based face recognition, in: Kernel Learning Algorithms for Face Recognition, Springer, 2014, pp. 71–99.
[60] H. Abdi, L.J. Williams, Principal component analysis, Wiley Interdiscip. Rev. Comput. Stat. 2 (4) (2010) 433–459.
[61] G. Huang, G. Huang, S. Song, K. You, Trends in extreme learning machines: a review, Neural Netw. 61 (2015) 32–48.
[62] S. Ding, H. Zhao, Y. Zhang, X. Xu, R. Nie, Extreme learning machine: algorithm, theory and applications, Artif. Intell. Rev. 44 (1) (2015) 103–115.
[63] G. Huang, L. Chen, C.K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. (TNN) 17 (4) (2006) 879–892.
[64] C.R. Rao, S.K. Mitra, Generalized inverse of matrices and its applications, John Wiley & Sons, New York, 1971.
[65] C.R. Johnson, Matrix Theory and Applications, 40, American Mathematical Soc., 1990.
[66] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, 2013.
[67] A. Asuncion, D. Newman, Uci machine learning repository, 2007. https://www.archive.ics.uci.edu/ml/index.php.

[68] Y. Zhang, D. Lo, X. Xia, J. Sun, An empirical study of classifier combination for cross-project defect prediction, in: Proceedings of the 39th Computer Software and Applications Conference (COMPSAC), 2, IEEE, 2015, pp. 264–269.
[69] L. Chen, B. Fang, Z. Shang, Y. Tang, Negative samples reduction in cross-company software defects prediction, Inf. Softw. Technol. (IST) 62 (2015) 67–77.
[70] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, 2010, p. 9.
[71] M. Jureczko, D. Spinellis, Using object-oriented design metrics to predict software defects, Models Methods Syst. Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej (2010) 69–81.
[72] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inf. Softw. Technol. (IST) 54 (3) (2012) 248–256.
[73] Z. Zhang, X. Jing, T. Wang, Label propagation based semi-supervised learning for software defect prediction, Autom. Softw. Eng. (ASE) 24 (1) (2017) 47–69.
[74] A.J. Smola, B. Schölkopf, Learning with Kernels, GMD-Forschungszentrum Informationstechnik, 1998.
[75] W. Yu, F. Zhuang, Q. He, Z. Shi, Learning deep representations via extreme learning machines, Neurocomputing 149 (2015) 308–315.
[76] K. Herzig, S. Just, A. Rau, A. Zeller, Predicting defects using change genealogies, in: Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2013, pp. 118–127.
[77] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, J. Liu, Dictionary learning based software defect prediction, in: Proceedings of the 36th International Conference on Software Engineering (ICSE), ACM, 2014, pp. 414–423.
[78] J. Hryszko, L. Madeyski, M. Dabrowska, P. Konopka, Defect prediction with bad smells in code, arXiv:1703.06300 (2017).
[79] D. Ryu, O. Choi, J. Baik, Value-cognitive boosting with a support vector machine for cross-project defect prediction, Empir. Softw. Eng. 21 (1) (2016) 43–71.
[80] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[81] T. Mende, R. Koschke, Effort-aware defect prediction models, in: Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, 2010, pp. 107–116.
[82] J.H. Zar, et al., Biostatistical Analysis, Pearson Education India, 1999.
[83] Y. Jiang, B. Cukic, Y. Ma, Techniques for evaluating fault prediction models, Empir. Softw. Eng. (ESE) 13 (5) (2008) 561–595.
[84] M. DAmbros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empir. Softw. Eng. (ESE) 17 (4–5) (2012) 531–577.
[85] J. Nam, S. Kim, Clami: defect prediction on unlabeled datasets, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 452–463.
[86] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, S. Ying, On the multiple sources and privacy preservation issues for heterogeneous defect prediction, IEEE Trans. Softw. Eng. (TSE) (2017).
[87] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, S. Ying, Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction, Autom. Softw. Eng. (ASE) 25 (2) (2018) 201–245.
[88] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, in: Proceedings of the 40th International Conference on Software Engineering (ICSE), ACM, 2018, p. 1063.
[89] B. Pizzileo, K. Li, G.W. Irwin, W. Zhao, Improved structure optimization for fuzzy-neural networks, IEEE Trans. Fuzzy Syst. (TFS) 20 (6) (2012) 1076–1089.
[90] C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines, IEEE Trans. Neural Netw. (TNN) 13 (2) (2002) 415–425.
[91] S.D. Thepade, M.M. Kalbhor, Novel data mining based image classification with Bayes, tree, rule, lazy and function classifiers using fractional row mean of cosine, sine and walsh column transformed images, in: Proceedings of the International Conference on Communication, Information and Computing Technology (ICCICT), IEEE, 2015, pp. 1–6.
[92] S. Shivaji, Efficient bug prediction and fix suggestions, University of California, Santa Cruz, 2013 Ph.D. thesis.
[93] Z. Sun, Q. Song, X. Zhu, Using coding-based ensemble learning to improve software defect prediction, IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) 42 (6) (2012) 1806–1817.
[94] Z. Islam, H. Giggins, Knowledge discovery through SysFor: a systematically developed forest of multiple decision trees, in: Proceedings of the Ninth Australasian Data Mining Conference-Volume 121, Australian Computer Society, Inc., 2011, pp. 195–204.
[95] M.J. Siers, M.Z. Islam, Cost sensitive decision forest and voting for software defect prediction, in: Proceedings of the Pacific Rim International Conference on Artificial Intelligence, Springer, 2014, pp. 929–936.
[96] H. Qu, G. Li, W. Xu, An asymmetric classifier based on partial least squares, Pattern Recognit. 43 (10) (2010) 3448–3457.
[97] X. Liu, J. Wu, Z. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern. Part B (Cybern.) 39 (2) (2009) 539–550.