Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction

Haonan Tong,Bin Liu,and Shihai Wang

Abstract—Cross-project defect prediction (CPDP) refers to predicting defects in the target project lacking of defect data by using prediction models trained on the historical defect data of other projects (i.e., source data). However, CPDP requires the source and target projects have common metric set (CPDP-CM). Recently, heterogeneous defect prediction (HDP) has drawn the increasing attention, which predicts defects across projects having heterogeneous metric sets. However, building high-performance HDP methods remains a challenge owing to several serious challenges including class imbalance problem, nonlinear, and the distribution differences between source and target datasets. In this paper, we propose a novel kernel spectral embedding transfer ensemble (KSETE) approach for HDP. KSETE first addresses the class-imbalance problem of the source data and then tries to find the latent common feature space for the source and target datasets by combining kernel spectral embedding, transfer learning, and ensemble learning. Experiments are performed on 22 public projects in both HDP and CPDP-CM scenarios in terms of multiple well-known performance measures such as, *AUC*, *G-Measure*, and *MCC*. The experimental results show that (1) KSETE improves the performance over previous HDP methods by at least 22.7%, 138.9%, and 494.4% in terms of *AUC*, *G-Measure*, and *MCC*, respectively. (2) KSETE improves the performance over previous CPDP-CM methods by at least 4.5%, 30.2%, and 17.9% in *AUC*, *G-Measure*, and *MCC*, respectively. It can be concluded that the proposed KSETE is very effective in both the HDP scenario and the CPDP-CM scenario.

Index Terms—heterogeneous defect prediction, cross-project defect prediction, class imbalance learning, spectral embedding, transfer learning, ensemble learning, multiple kernel learning

1 Introduction

Software defect prediction (SDP) is one of the most active research areas in software engineering and has drawn increasing attention of both academic and industrial communities [1], [2], [3], [4], [5], [6], [7], [8], [9]. SDP plays a crucial role in helping to allocate the limited test resources reasonably and improve test efficiency, which aims to predict the defect status of software modules before software testing [10]. Most SDP approaches build the prediction models based on the collected historical defect dataset of a project and apply the trained models to predict the defects of new software modules from the same project, which are called as within-project defect prediction (WPDP) [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. The historical defect dataset consists of the information of software metrics and the defects. Software metrics mainly include static code metrics (such as Halstead metrics [22], McCabe metrics [23], CK metrics [24]) and process metrics ([25], [26], [27]). However, it is difficult to perform WPDP for new projects or projects without sufficient historical defect data [28], [29].

Cross-project defect prediction (CPDP) has been proposed by researchers to address above issue [29], [30], [31], [32], [33], [34], [35], [36]. CPDP methods aim to predict the defects of a project lacking of historical defect data by using the prediction model trained on other similar projects' defect data. Fortunately, there are many publicly available defect datasets in PROMISE repositories [37], which makes it possible to use these public datasets to predict defects of projects lacking historical defect data. These methods can be divided into two categories: instance-based method [29], [30], [31], [32], [35] and feature-based method [33], [34]. Instance-based CPDP method tries to reduce the distribution differences between source and target datasets by selecting similar instances to the target data from the source data or weighting the source instances. Feature-based CPDP method attempts to maximize the distribution similarity between the source and target data by finding the latent common feature space. Existing CPDP methods assume that the source project dataset and the target project dataset use the common software metrics. However, in practice, the source and target datasets may use different software metrics, i.e., different type or the number of metrics. In this scenario, CPDP is specially called as HDP [38].

To address the problem of defect prediction of projects with the heterogeneous metric sets, HDP methods have been proposed by researchers [38], [39], [40], [41]. Jing *et al.* [39] proposed an HDP method CCA+, which uses the canonical correlation analysis (CCA) technique and the unified metric representation (UMR) to find the latent common feature space. Nam and Kim [38] presented an HDP method, which uses feature selection and Kolmogorov–Smirnov test based matching method to reduce the distribution differences between source and target data. Ma *et al.* [40] proposed KCCA+ for HDP, which combines kernel method and CCA technique. Li *et al.* [41] presented a new HDP method called CTKCCA by combining cost-sensitive learning, kernel method, and CCA technique.

However, the building of high-performance HDP models is still a serious challenge owing to some critical problems including the class-imbalance distribution, the com-

- *Haonan Tong, Bin Liu, and Shihai Wang are with Science and Technology on Reliability and Environmental Engineering Laboratory, School of Reliability and Systems Engineering, Beihang University, Beijing, China, 100191. E-mail: {tonghaonan, liubin, wangshihai}@buaa.edu.cn*
- *Shihai Wang is the corresponding author.*
  E-mail: wangshihai@buaa.edu.cn

Manuscript received January 22, 2019; revised May 26, 2019.

plex nonlinear relationship between the source and target datasets, the distribution differences between the source and target datasets and so on. The class-imbalance distribution is a main factor accounting for the unsatisfactory prediction performance [5], [42]. It can cause poor performance on the minority class samples for the prediction models. However, existing HDP methods (such as CCA+, HDP-KS, and KCCA+) except CTKCCA, do not consider the class-imbalance problem. Although CTKCCA uses cost-sensitive method for addressing this problem, an appropriate cost matrix is difficult to determine. According to [40], [41], the nonlinear relationship between the source and target datasets seems to have an impact on the prediction performance. The distribution difference between the source and target datasets, especially in the scenario of HDP, makes the building of high-performance defect prediction models a serious challenge. The class-imbalance distribution and nonlinear correlation further increase the learning difficulty of HDP.

In this paper, we propose a novel kernel spectral embedding transfer ensemble method for HDP. The main idea of our HDP method is to first balance the source dataset, and then find a series of the latent common kernel feature subspace that each subspace not only maximizes the similarity between the projected source and target datasets but preserves the intrinsic characteristic of both source and target datasets. Finally, we built classifiers on each common feature space and combine them for predicting the labels of the target data. The experimental results show that our KSETE significantly performs better than previous state-of-the-art HDP and CPDP-CM methods.

The contributions of this paper are as follows:

(1) Considering the problems including imbalanced data, nonlinear correlation, and distribution differences existing in the heterogeneous defect prediction, a novel HDP method named KSETE based on class-imbalance learning, multi-kernel learning, transfer learning, and ensemble learning is proposed in this paper.

(2) To evaluate the performance of the proposed KSETE, extensive experiments are performed on 22 publicly available datasets from three communities including AEEEM [33], JIRA [43], and PROMISE [44] in both HDP and CPDP-CM scenarios. The experimental results show that our KSETE is very effectiveness compared with the baselines.

(3) A replication package of KSETE is shared on GitHub[1] and Zenodo[2], which is publicly available.

The rest of this paper is organized as follows: Previous works about CPDP-CM and HDP in the SDP field are briefly reviewed in Section 2. We formulate the problem of HDP in Section 3. The details of our proposed KSETE are presented in Section 4. The experimental setup is described in Section 5. Section 6 shows the experimental results. This study is discussed in Section 7. Some potential threats to our study are shown in Section 8. Finally, we make a conclusion about our research in Section 9.

1. https://github.com/THN-BUAA/KSETE-master.git
2. http://doi.org/10.5281/zenodo.3362613

2 Related Works

In this section, we briefly introduce the previous cross-project defect prediction methods using common metric set and the heterogeneous defect prediction methods in the SDP field.

2.1 Cross-Project Defect Prediction Using Common Metrics

Many CPDP methods have been proposed in previous SDP studies [29], [30], [31], [31], [32], [33], [35], [35], [45].

Some CPDP methods attempt to select similar training data with target data from the source data [29], [30], [31], [35], [45]. Turhan *et al.* [29] proposed a CPDP method named Burak filter, which uses the k-nearest neighbor (NN) method to select (aka. filter) training instances from the source data. Specifically, for each instance in target data, they selected some nearest source instances based on Euclidean distance as part of the training data. Ryu *et al.* [30] also developed a CPDP method called hybrid instance selection using the nearest neighbor (HISNN), which is made of two main phases: target data instance selection and source data instance filtering.

Some CPDP methods attempt to reduce the distribution difference between the source and target data by weighting source instances according to their similarity with the target data [31], [32], [35]. Ma *et al.* [31] proposed a CPDP approach named transfer naive Bayes (TNB). For each instance in source data, its similarity weight is calculated based on the distributional characteristics of target data and then constructed a classifier based on naive Bayes. Xia *et al.* [32] developed a CPDP approach named HYDRA, which can adjust the weights of source instances during the training of their proposed extended AdaBoost algorithm. Ryu *et al.* [35] proposed a value-cognitive boosting with support vector machine (VCB-SVM) approach considering class imbalance learning for CPDP. Specifically, the similarity weights of source data instances are used in a mechanism to balance the training data and in the process to train the defect prediction model.

Some CPDP approaches aim to find latent common feature space which can minimize the distribution differences between source and target datasets [33] [36]. Nam *et al.* [33] proposed a new CPDP method called TCA+, which extends transfer component analysis (TCA) by introducing a set of rules for selecting an appropriate normalization method to obtain better CPDP performance. Krishna and Menzies [36] introduced a baseline method named Bellwether for cross-project defect prediction based on existing CPDP methods (such as TCA+ and TNB).

Above CPDP approaches assume that there are common metric sets in both source and target projects. However, the number of common metrics may be very small, even zero. In this scenario, these approaches cannot achieve good-enough prediction performance.

2.2 Heterogenous Defect Prediction

Recently, heterogenous defect prediction methods [39], [40], [41], [46] have been proposed to predict defects across projects which have heterogeneous metric sets.

Jing *et al.* [39] proposed an HDP method named CCA+, which uses the canonical correlation analysis (CCA) technique [47] and the unified metric representation (UMR) to find the latent common feature space between the source and target projects. Specifically, the UMR is made of three kinds of metrics, including the common metrics of the source and target data, source-specific metrics, and target-specific metrics. Based on UMR, the transfer learning method based on CCA is introduced to find common metrics by maximizing the canonical correlation coefficient between source and target data.

Nam and Kim [46] also proposed an HDP method. They firstly used a feature selection method on the source to select a subset of metrics by removing redundant and irrelevant metrics. Then, they matched the source and target metrics based on metric similarity (e.g. Kolmogorov–Smirnov test based matching, KSAnalyzer). With the matched source and target metric sets, they built a defect prediction model for HDP.

Considering the complex nonlinear relationships between software metrics and software defects, Ma *et al.* [40] proposed an HDP method called kernel canonical correlation analysis (KCCA+) by combining kernel method and CCA technique.

Li *et al.* [41] argued that previous HDP methods do not consider the class-imbalance problem, which increases the difficulty of building a good-performance HDP model. Therefore, they proposed a new HDP method named CTKCCA by combining cost-sensitive learning, kernel method, and CCA technique. The experimental results show that CTKCCA performs better than the related CPDP methods and state-of-the-art HDP methods.

However, the performance of existing HDP methods are less than satisfactory owing to several serious challenges including class imbalance problem, complex nonlinear relationship between source and target datasets and between software metrics and defects, and making two distribution similar without losing too much intrinsic characteristic of original datasets.

2.3 Transfer Learning

A major assumption in traditional machine learning approaches is that the training and future data must have the same feature space and the same distribution [48]. Differing from traditional machine learning approaches, transfer learning does not require this assumption and allows transfer domain knowledge between different domains. According to [48], the definition of transfer learning is: *Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and D_T, where $D_S \neq D_T$, or $T_D \neq T_T$.*

Transfer learning techniques have been applied in many fields, such as image classification [49], [50], speech recognition [51], face recognition [52], robots modeling [53], and recommended systems [54]. Actually, CPDP is also a transfer learning problem in the SDP field [31].

3 Problem Formulation

Denote a labeled source project dataset as $\{S = (x_S^{(1)}, \cdots, x_S^{(n_S)})^{\mathrm{T}}, Y = (y_S^{(1)}, \cdots, y_S^{(n_S)})^{\mathrm{T}}\}$ and an unlabeled dataset from the target project as $\{T = (x_T^{(1)}, \cdots, x_T^{(n_T)})^{\mathrm{T}}\}$, where n_S and n_T respectively denote the number of modules (such as functions/classes/files depending on the granularity considered) in the source and target projects, $x_S^{(i)} \in R^{1*d_S}$ represents the d_S metrics' value of i-th module in the source project, $x_T^{(j)} \in R^{1*d_T}$ denotes the value of d_T metrics of j-th module in the target project, and $y_S^{(i)} \in \{0, 1\}$ corresponds the defect information of i-th source project module ($y_S^{(i)}$=0 denotes *non-defective* and $y_S^{(i)}$=1 means *defective*). Note that source and target datasets have heterogeneous metric sets. A module is defective if it has one or more defects. With respect to the unlabeled target project T, the objective of KSETE is to predict their labels by using the labeled source data.

We aim to find a common feature space for the source and target datasets. The optimal projected space is defined as follows:

Definition 1: Given the source dataset S and the target dataset T, let $\Phi(\cdot)$ be a mapping function, $\Phi(S) \in R^{n_S*p}$ and $\Phi(T) \in R^{n_T*q}$ denote respectively the projected source and target datasets, then the optimal further projection of the projected source dataset $B_{\Phi(S)}$, and that of the projected target dataset $B_{\Phi(T)}$ are given by the following optimization objective:

$$\begin{aligned}\underset{B_{\Phi(S)}, B_{\Phi(T)}}{min} = & \; L(B_{\Phi(S)}, \Phi(S)) + L(B_{\Phi(T)}, \Phi(T)) \\ & + \beta \cdot D(B_{\Phi(S)}, B_{\Phi(T)})\end{aligned} \tag{1}$$

where $L(*, *)$ denotes the difference between the projected dataset by using $\Phi(\cdot)$ and its further projected dataset (e.g., $\Phi(S)$ and $B_{\Phi(S)}$), the difference between the two further projected datasets are represented as $D(B_{\Phi(S)}, B_{\Phi(T)})$, β is a hyper-parameter to control how desirable the two datasets are similar. We further define $D(B_{\Phi(S)}, B_{\Phi(T)})$ with $L(*, *)$ as follows:

$$D(B_{\Phi(S)}, B_{\Phi(T)}) = \frac{1}{2}\left(L(B_{\Phi(S)}, \Phi(T)) + L(B_{\Phi(T)}, \Phi(S))\right) \tag{2}$$

which is the average of the difference between the projected source dataset $\Phi(S)$ and the further projected target dataset $B_{\Phi(T)}$, and that between the projected target dataset $\Phi(T)$ and the further projected source dataset $B_{\Phi(S)}$.

4 Proposed Method

Fig 1 presents the overall framework of the proposed KSETE for HDP. In the following paragraphs, we present the details of KSETE from three aspects: (1) preprocessing, (2) kernel spectral embedding, and (3) transfer ensemble learning.

4.1 Preprocessing

The preprocessing includes following four processes: (1) remove the duplicated instances in S and T; (2) remove the instances having missing value; (3) alleviate the class

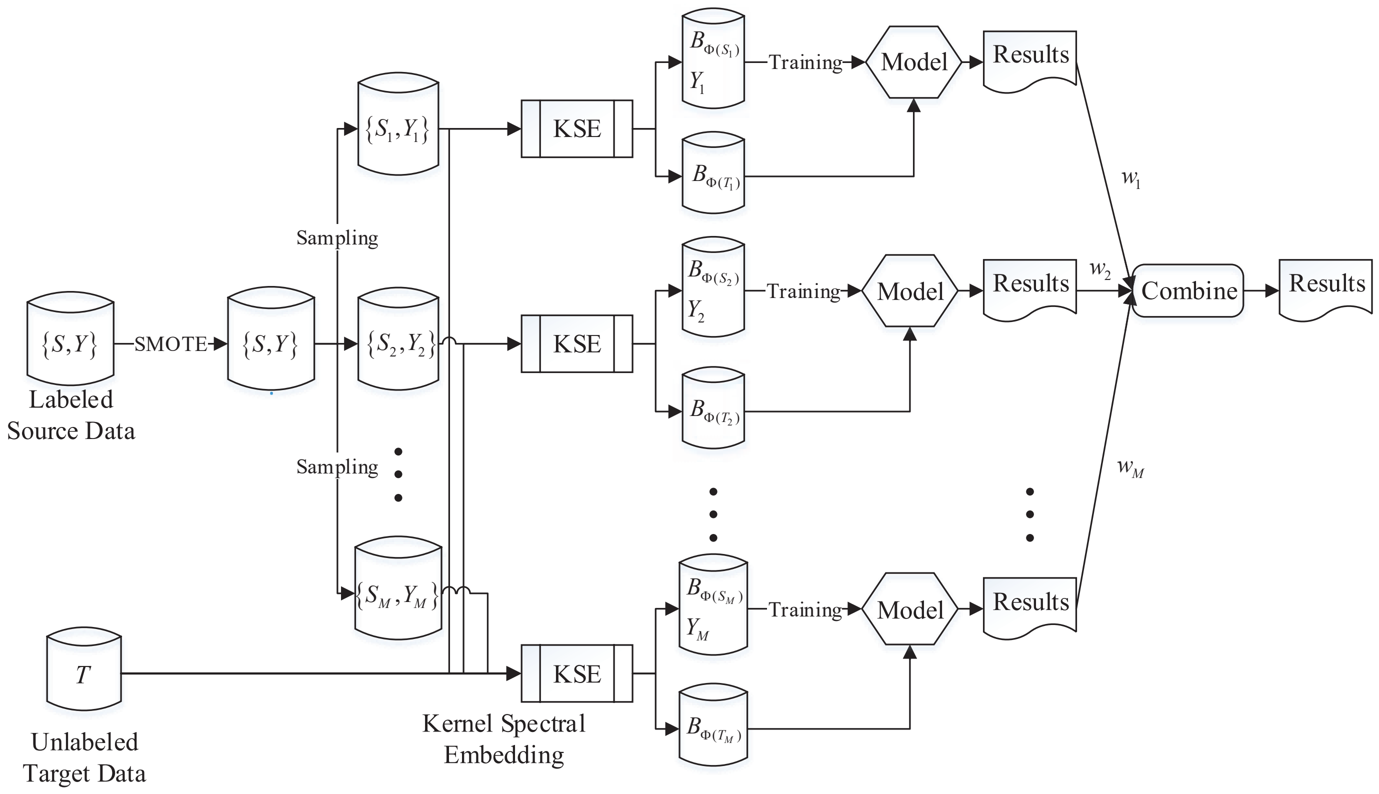


Fig. 1: Framework of our proposed KSETE for heterogeneous defect prediction.

imbalance problem by performing synthetic minority over-sampling technique (SMOTE) [55] on S; (4) normalize S and T with Z-score [56].

SMOTE is one of the most commonly used class-imbalance learning methods in the SDP field. Dealing with the class imbalance problem has been demonstrated to improve the prediction performance [5] [36].

Considering the magnitude differences of different metrics in defect datasets, we use Z-score normalization to scale each metric to have mean 0 and standard deviation 1.

4.2 Kernel Spectral Embedding

For the sake of computation, the source and target datasets must be preprocessed to have the same number of instances (supposing it to be n). In this paper, random sampling is used to increase the number of instances of the smaller dataset. Note that, if the size of target data is increased, the added instances must be removed before predicting the labels of target instances.

Denote the mapping matrices of projected source and target datasets as $P_{\Phi(S)} \in R^{k*p}$ and $P_{\Phi(T)} \in R^{k*q}$, respectively. The differences between the projected dataset and the original dataset (distortion function $L(*,*)$ in Eq.(1)) are defined as follows:

$$\begin{aligned} L(B_{\Phi(S)}, \Phi(S)) &= \left\|B_{\Phi(S)} P_{\Phi(S)} - \Phi(S)\right\|_F^2 \\ L(B_{\Phi(T)}, \Phi(T)) &= \left\|B_{\Phi(T)} P_{\Phi(T)} - \Phi(T)\right\|_F^2 \\ L(B_{\Phi(S)}, \Phi(T)) &= \left\|B_{\Phi(S)} P_{\Phi(T)} - \Phi(T)\right\|_F^2 \\ L(B_{\Phi(T)}, \Phi(S)) &= \left\|B_{\Phi(T)} P_{\Phi(S)} - \Phi(S)\right\|_F^2 \end{aligned} \tag{3}$$

where $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$ represents the Frobenius norm.

With Eq.(1) and (3), the optimization objective can be rewritten as:

$$\begin{aligned} &\underset{B_{\Phi(S)}^{\mathrm{T}} B_{\Phi(S)}=I, B_{\Phi(T)}^{\mathrm{T}} B_{\Phi(T)}=I}{min} G\left(B_{\Phi(S)}, B_{\Phi(T)}, P_{\Phi(S)}, P_{\Phi(T)}\right) \\ &= \underset{B_{\Phi(S)}^{\mathrm{T}} B_{\Phi(S)}=I, B_{\Phi(T)}^{\mathrm{T}} B_{\Phi(T)}=I}{min} \left\|\Phi(S) - B_{\Phi(S)} P_{\Phi(S)}\right\|_F^2 \\ &+ \left\|\Phi(T) - B_{\Phi(T)} P_{\Phi(T)}\right\|_F^2 + \frac{\beta}{2} * \left\|\Phi(S) - B_{\Phi(T)} P_{\Phi(S)}\right\|_F^2 \\ &+ \frac{\beta}{2} * \left\|\Phi(T) - B_{\Phi(S)} P_{\Phi(T)}\right\|_F^2 \end{aligned} \tag{4}$$

where $B_{\Phi(S)} \in R^{n*k}$ and $B_{\Phi(T)} \in R^{n*k}$ are assumed orthogonal, and β is a hyper-parameter to adjust how desirable the projected datasets (i.e., $B_{\Phi(S)}$ and $B_{\Phi(T)}$) are similar.

Lemma 1. The $B_{\Phi(S)}, B_{\Phi(T)}, P_{\Phi(S)}, P_{\Phi(T)}$ in (4) have the following properties:

$$\begin{aligned} P_{\Phi(S)} &= \frac{1}{2+\beta}\left(2B_{\Phi(S)}^{\mathrm{T}}\Phi(S) + \beta B_{\Phi(T)}^{\mathrm{T}}\Phi(S)\right) \\ P_{\Phi(T)} &= \frac{1}{2+\beta}\left(2B_{\Phi(T)}^{\mathrm{T}}\Phi(T) + \beta B_{\Phi(S)}^{\mathrm{T}}\Phi(T)\right) \end{aligned} \tag{5}$$

Proof: Based on the relationship of between Frobenius norm and matrix trace, the properties of orthogonal matrix and cyclic permutation property of trace:

$$\begin{aligned} &\|X\|_F^2 = tr(X^{\mathrm{T}}X), B_{\Phi(S)}^{\mathrm{T}} B_{\Phi(S)} = I, \\ &tr\left(P_{\Phi(S)}^{\mathrm{T}} B_{\Phi(S)}^{\mathrm{T}} \Phi(S)\right) = tr\left(B_{\Phi(S)}^{\mathrm{T}} \Phi(S) P_{\Phi(S)}^{\mathrm{T}}\right) \end{aligned} \tag{6}$$

the following deduction can be obtained:

$$\begin{aligned}&\left\|\Phi(S)-B_{\Phi(S)}P_{\Phi(S)}\right\|_F^2=tr\left(\Phi(S)^{\mathrm{T}}\Phi(S)\right)\\&-2tr\left(B_{\Phi(S)}^{\mathrm{T}}\Phi(S)P_{\Phi(S)}^{\mathrm{T}}\right)+tr\left(P_{\Phi(S)}^{\mathrm{T}}P_{\Phi(S)}\right)\end{aligned}\tag{7}$$

Let $\left\|\Phi(T)-B_{\Phi(T)}P_{\Phi(T)}\right\|_F^2$, $\left\|\Phi(S)-B_{\Phi(T)}P_{\Phi(S)}\right\|_F^2$, and $\left\|\Phi(T)-B_{\Phi(S)}P_{\Phi(T)}\right\|_F^2$ be expressed in the same fashion. The optimization problem in (4) can be rewritten as:

$$\begin{aligned}&\min_{B_{\Phi(S)}^{\mathrm{T}}B_{\Phi(S)}=I,B_{\Phi(T)}^{\mathrm{T}}B_{\Phi(T)}=I} G\left(B_{\Phi(S)},B_{\Phi(T)},P_{\Phi(S)},P_{\Phi(T)}\right)\\&=\min_{B_{\Phi(S)}^{\mathrm{T}}B_{\Phi(S)}=I,B_{\Phi(T)}^{\mathrm{T}}B_{\Phi(T)}=I}\left(1+\frac{\beta}{2}\right)tr(\Phi(S)^{\mathrm{T}}\Phi(S))\\&+\left(1+\frac{\beta}{2}\right)tr(\Phi(T)^{\mathrm{T}}\Phi(T))+\left(1+\frac{\beta}{2}\right)tr(P_{\Phi(S)}^{\mathrm{T}}P_{\Phi(S)})\\&+\left(1+\frac{\beta}{2}\right)tr(P_{\Phi(T)}^{\mathrm{T}}P_{\Phi(T)})-2tr\left(B_{\Phi(S)}^{\mathrm{T}}\Phi(S)P_{\Phi(S)}^{\mathrm{T}}\right)\\&-2tr\left(B_{\Phi(T)}^{\mathrm{T}}\Phi(T)P_{\Phi(T)}^{\mathrm{T}}\right)-\beta\cdot tr\left(B_{\Phi(S)}^{\mathrm{T}}\Phi(T)P_{\Phi(T)}^{\mathrm{T}}\right)\\&-\beta\cdot tr\left(B_{\Phi(T)}^{\mathrm{T}}\Phi(S)P_{\Phi(S)}^{\mathrm{T}}\right)\end{aligned}\tag{8}$$

The derivation of G with respect to $P_{\Phi(S)}$ and $P_{\Phi(T)}$ can be obtained as follows:

$$\begin{aligned}\nabla G\left(P_{\Phi(S)}\right)&=\frac{1}{2+\beta}\left(\beta\cdot B_{\Phi(T)}^{\mathrm{T}}\Phi(S)+2B_{\Phi(S)}^{\mathrm{T}}\Phi(S)\right)\\\nabla G\left(P_{\Phi(T)}\right)&=\frac{1}{2+\beta}\left(\beta\cdot B_{\Phi(S)}^{\mathrm{T}}\Phi(T)+2B_{\Phi(T)}^{\mathrm{T}}\Phi(T)\right)\end{aligned}\tag{9}$$

According to the Karush-Kuhn-Tucker conditions, let $\nabla G\left(P_{\Phi(S)}\right)=0$ and $\nabla G\left(P_{\Phi(T)}\right)=0$, and then we can obtain (5), namely

$$\begin{aligned}P_{\Phi(S)}&=\frac{1}{2+\beta}\left(2B_{\Phi(S)}^{\mathrm{T}}\Phi(S)+\beta B_{\Phi(T)}^{\mathrm{T}}\Phi(S)\right)\\P_{\Phi(T)}&=\frac{1}{2+\beta}\left(2B_{\Phi(T)}^{\mathrm{T}}\Phi(T)+\beta B_{\Phi(S)}^{\mathrm{T}}\Phi(T)\right)\end{aligned}$$

□

Based on (4) and (5), the closed form of $B_{\Phi(T)}$ and $B_{\Phi(S)}$ can be derived according to the following theorem.

Theorem 1. The minimization problem in (4) equals to the following maximization problem:

$$\min_{B_{\Phi(S)}^{\mathrm{T}}B_{\Phi(S)}=I,B_{\Phi(T)}^{\mathrm{T}}B_{\Phi(T)}=I} G=\max_{B^TB=I} tr\left(B^TAB\right)\tag{10}$$

where

$$B=\begin{bmatrix}B_{\Phi(T)}\\B_{\Phi(S)}\end{bmatrix}\quad A=\begin{bmatrix}A_1&A_2\\A_3&A_4\end{bmatrix},\tag{11}$$

and

$$\begin{aligned}&A_1=2\Phi(T)\Phi(T)^{\mathrm{T}}+\frac{\beta^2}{2}\Phi(S)\Phi(S)^{\mathrm{T}}=2K(T,T)\\&+\frac{\beta^2}{2}K(S,S),\\&A_4=2\Phi(S)\Phi(S)^{\mathrm{T}}+\frac{\beta^2}{2}\Phi(T)\Phi(T)^{\mathrm{T}}=2K(S,S)\\&+\frac{\beta^2}{2}K(T,T),\\&A_2=A_3=\beta\left(\Phi(S)\Phi(S)^{\mathrm{T}}+\Phi(T)\Phi(T)^{\mathrm{T}}\right)\\&=\beta\left(K(S,S)+K(T,T)\right)\end{aligned}\tag{12}$$

Proof: Based on (5), the optimization problem in (4) can be rewritten as:

$$\begin{aligned}&\min_{B_{\Phi(S)}^{\mathrm{T}}B_{\Phi(S)}=I,B_{\Phi(T)}^{\mathrm{T}}B_{\Phi(T)}=I} G\left(B_{\Phi(S)},B_{\Phi(T)},P_{\Phi(S)},P_{\Phi(T)}\right)\\&=\left(1+\frac{\beta}{2}\right)\cdot tr\left(\Phi(S)^{\mathrm{T}}\Phi(S)\right)+\left(1+\frac{\beta}{2}\right)\cdot tr\left(\Phi(T)^{\mathrm{T}}\Phi(T)\right)\\&-\frac{1}{2+\beta}B^{\mathrm{T}}AB\end{aligned}\tag{13}$$

Since $tr\left(\Phi(S)^{\mathrm{T}}\Phi(S)\right)$ and $tr\left(\Phi(T)^{\mathrm{T}}\Phi(T)\right)$ are constants, above optimization problem equals to:

$$\min_{B_{\Phi(S)}^{\mathrm{T}}B_{\Phi(S)}=I,B_{\Phi(T)}^{\mathrm{T}}B_{\Phi(T)}=I} G=\max_{B^TB=I} tr\left(B^TAB\right)$$

□

Since $\Phi(S)\Phi(S)^{\mathrm{T}}$ and $\Phi(T)\Phi(T)^{\mathrm{T}}$ are all symmetric, A_1, A_2, A_3, and A_4 in (12) are also symmetric, and then A in (11) is symmetric. Therefore, the optimization problem in (10) has a closed-form solution under the condition $B^{\mathrm{T}}B=I$.

Theorem 2. (Ky-Fan theorem [57]) Denote A as a symmetric matrix with eigenvalues $\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_k$ and eigenvectors $V=[v_1,v_2,\cdots,v_k]$. Then,

$$\sum_{i=1}^{k}\lambda_i=\max_{X^{\mathrm{T}}X=I_k} tr\left(X^{\mathrm{T}}MX\right)$$

and $X=[v_1,v_2,\cdots,v_k]\,Q$ where Q is an arbitrary orthogonal matrix.

According to *Theorem 2*, the optimal B in (10) is the top-k eigenvectors of A. The first half rows of B and the second half rows of B are the best projected datasets $B_{\Phi(T)}$ and $B_{\Phi(S)}$, respectively. The algorithm is presented in Algorithm 1. Here, the Lanczos method [58] is used to calculate the eigenvectors.

4.3 Transfer Ensemble Learning

The objective of transfer ensemble learning is to build a transfer-learning based ensemble classifier for HDP by performing KSE multiple times on the target data and multiple feature subspace of source data. To this end, the framework of random forest (RF) [59] is adopted in this paper. RF is a very famous ensemble learning algorithm, which takes the decision tree as the base learner. RF is characterized by the random sampling of training instances with replacement, the random selection of features of training data, and the combining rule (i.e., voting for classification, mean value for

Algorithm 1 Kernel Spectral Embedding

```
Input: Source data $S$; Target data $T$; Similarity confidence parameter $\beta$; Dimensionality of the projected feature space $k$; Kernel function $K(x,x)$
Output: Projected source data $B_{\Phi(S)}$; Projected target data $B_{\Phi(T)}$
1: Remove the duplicated instances and the instances having missing value for both $S$ and $T$.
2: Perform SMOTE algorithm [55] on $S$, and denote the sampled source data as $S$ again.
3: Increase the size of smaller dataset by random sampling to make the source dataset and the target dataset have the same number of instances. Note that, if the number of instances in target data is increased, these added instances must be removed after the projected target data is obtained.
4: Use z-score to scale $S$ and $T$.
5: Construct matrix A according to Eq.(11) and (12).
6: Calculate top $k$ eigenvalues and the corresponding eigenvectors $V = [v_1, v_2, \cdots, v_k]$.
7: $B_{\Phi(T)}$ and $B_{\Phi(S)}$ are respectively the first half rows and the second half rows of $V$: $B_{\Phi(T)} = \begin{bmatrix} V(1,1) & \cdots & V(1,k) \\ \vdots & \ddots & \vdots \\ V(\frac{l}{2},1) & \cdots & V(\frac{l}{2},k) \end{bmatrix}$, $B_{\Phi(S)} = \begin{bmatrix} V(\frac{l}{2}+1,1) & \cdots & V(\frac{l}{2}+1,1) \\ \vdots & \ddots & \vdots \\ V(l,1) & \cdots & V(l,k) \end{bmatrix}$ where $l$ is the number rows of $V$.
8: Return $B_{\Phi(T)}$ and $B_{\Phi(S)}$.
```

regression). Specifically, given the training and test datasets, the number of base learners M, and the dimension of feature subspace k, RF first constructs a series of training subsets by performing the bootstrap method and random selection of feature subspace on the original training dataset. And then, the base learner is trained on each above training subsets. Finally, using the voting rule to combine the predicted labels of all base learners on the test instances. However, there are some main differences between our transfer ensemble learning and RF: (1) RF is not for transfer learning, which is designed for the traditional supervised machine learning task. (2) To adapt RF for transfer learning, the base learner cannot be trained directly on each training subset as RF. We first feed each training subset and the target dataset into our proposed KSE to obtain the corresponding projected datasets and then train the base learner on the projected source dataset.

The details of transfer ensemble learning are presented as follows. Before iteration, we first preprocess the source data as Section 4.1 (Step1-2) and then ensure the source data and target data have the same number of instances (Step3). For each iteration, we first make a subset of the source data, which has the same number of instances as the source data and partial or the same features as the source data (Step5-6). And then use z-score to scale the source and target datasets and calculate the projected datasets $B_{\Phi(S)}$ and $B_{\Phi(T)}$ by using KSE algorithm (Step7). If the number of the target data is increased in Step3, then the added instances must be removed from $B_{\Phi(T)}$ (Step8). We next train a classifier using the projected source data and then make the probability prediction on the projected target dataset (Step9-11). And assign a weight to this classifier (Step12). When the stop criterion of iteration is met, we combine the prediction results of all classifiers and output the predicted labels of the target data (Step13-15). Algorithm 2 presents the details of our proposed transfer ensemble learning method.

Algorithm 2 Transfer Ensemble Learning

```
Input: Source data $S$ and the labels $Y$; Target data $T$; Similarity confidence parameter $\beta$; the number of selected features $K$; Dimensionality of the projected feature space $k$; Kernel function $K(x,x)$, base learner $learner$, the number of base learners $M$
Output: Predicted labels of target data $T$.
1: Remove the duplicated instances and the instances having missing value for both $(S,Y)$ and $T$.
2: Perform SMOTE algorithm [55] on $(S,Y)$ and denote the sampled source dataset as $(S,Y)$.
3: Increase the size of smaller dataset by random sampling to make $(S,Y)$ and $T$ have the same number of instances. Note that, if the size of $T$ is increased, the added instances must be removed from the projected target data is obtained.
4: for $i = 1 : M$ do
5:     Take a bootstrap sample $(S_i, Y_i)$ from $(S,Y)$, which has the same size as $(S,Y)$.
6:     Select $k$ features (columns) from $S_i$ to replace $S_i$.
7:     Use z-score to scale $S_i$ abd $T$
8:     Perform KSE (Algorithm 1) on $S_i$ and $T$ and denote the obtained optimal projected datasets as $B_{\Phi(S_i)}$ and $B^i_{\Phi(T)}$. Note that, here there is no need to perform the Steps 1-4 of Algorithm (1).
9:     Remove the added instances in $B^i_{\Phi(T)}$ according to step 3, if any.
10:    Using $(B_{\Phi(S_i)}, Y_i)$ as the training data to train $learner$ and then obtain a hypothesis $h_i(x) : x \to [0,1]$.
11:    Apply $h_i$ on $B^i_{\Phi(T)}$, and denote the probabilistic prediction as $pro_i$.
12:    Assign a weight $w_i$ to $h_i$, by default $w_i = 1/M$.
13: end for
14: The final hypothesis is obtained:
    $h_f(x) = arg \max_{y\in\{0,1\}} \sum_{i=1}^{M} w_i \cdot pro_i\,[h_i(x) = y]$
15: Suppose $B^i_{\Phi(T)}[j]$ is $j$-th instance of $B^i_{\Phi(T)}$, then the label of $j$-th instance of $T$ is predicted as follows:
    $arg \max_{y\in\{0,1\}} \sum_{i=1}^{M} w_i \cdot pro_i \left[h_i(B^i_{\Phi(T)}[j]) = y\right]$
16: Return the predicted labels of the target data.
```

5 Experimental Setup

This section shows the details of our experiments including the research questions, benchmark datasets, performance measures, statistical testing methods, and the experimental settings.

5.1 Research Questions

We investigate two questions in this paper as follows:

RQ1: Does KSETE outperform previous HDP methods?

Motivation. The main objective of this study is to propose a novel HDP method. Therefore, we must first raise and answer the most important research question that whether our method can obtain a better performance than previous HDP methods.

Approach. To answer RQ1, we compare KSETE with three state-of-the-art HDP approaches including CCA+ [39], HDP-KS [46], and CTKCCA [41] by using hypothesis testing (see Section 5.4 for details). All the baselines use their default settings according to corresponding original studies. Based on aforementioned statistical testing methods, we propose the following hypotheses:

- Null hypothesis: KSETE has no statistically significant difference from other HDP methods.
- Alternative hypothesis: KSETE outperforms other HDP methods with statistical significance.

RQ2: Does KSETE outperform previous CPDP-CM methods?

Motivation. Although CPDP-CM methods cannot be used in the HDP scenario, HDP methods can be used in the CPDP-CM scenario. We then want to investigate whether our method can outperform existing CPDP-CM methods or not.

Approach. To answer RQ2, we compare KSETE with five well-known CPDP-CM methods including Burak filter [29], TNB [31], TCA+ [33], HISNN [30], and Bellwether (use TCA+ as the transfer learner) [36] by hypothesis testing (see Section 5.4 for details). All the baselines use their default settings according to corresponding original studies. And then we propose the following hypotheses:

- Null hypothesis: Null hypothesis: KSETE has no statistically significant difference from other CPDP-CM methods.
- Alternative hypothesis: KSETE outperforms other CPDP-CM methods with statistical significance.

5.2 Benchmark Datasets

A total of 22 publicly available datasets from three different communities including AEEEM [33], JIRA [43], and PROMISE [44] are used in our experiments. Datasets in the same community have the same metric sets, but datasets in different communities have heterogeneous metric sets. The NASA datasets used in previous studies [41] [46] are not utilized in this paper owing to the data quality problems [60]. Since we do not consider the inter-release defect prediction [61], for the multiple-version projects, only one version is used. Table 1 shows the details of these datasets. A brief description of each community is presented as follows:

AEEEM was collected by D'Ambros *et al.* [62]. Each dataset in AEEEM consists of 61 software metrics: 17 source code metrics (i.e., CK metrics [24] and other 11 object-oriented metrics), 17 entropy-of-source-code metrics, 17 churn-of-source-code metrics, 5 entropy-of-change metrics, and some other metrics.

PROMISE was prepared by Jureczko and Madeyski [44]. It contains 20 class-level metrics, such as CK metrics [24] and QMOOD metrics [63].

JIRA is a new repository of highly-curated defect datasets collected by Yatish *et al.* [43]. JIRA contains 65 software metrics: 54 code metrics, 5 process metrics, and 6 ownership metrics. Each module in JIRA refers to a java file.

TABLE 1: An Overview of 22 Studied Projects

Community	Project	Module Level	# of Metrics	# of Instances	
				All	Defective(%)
AEEEM [33]	EQ	Class	61	324	129 (39.81%)
	JDT	Class	61	997	206 (20.66%)
	Lucene	Class	61	691	64 (9.26%)
	Mylyn	Class	61	1862	245 (13.16%)
	PDE	Class	61	1497	209 (13.96%)
JIRA [43]	activemq-5.0.0	File	65	1884	293 (15.55%)
	derby-10.5.1.1	File	65	2705	383 (14.16%)
	groovy-1_6_BETA_1	File	65	821	70 (8.53%)
	hbase-0.94.0	File	65	1059	218 (20.59%)
	hive-0.9.0	File	65	1416	283 (19.99%)
	jruby-1.1	File	65	731	87 (11.9%)
	wicket-1.3.0-beta2	File	65	1763	130 (7.37%)
PROMISE [44]	ant-1.7	Class	20	745	166 (22.28%)
	camel-1.4	Class	20	872	145 (16.6%)
	ivy-2.0	Class	20	352	40 (11.36%)
	jedit-4.0	Class	20	306	75 (24.51%)
	log4j-1.0	Class	20	135	34 (25.19%)
	poi-2.0	Class	20	314	37 (11.78%)
	tomcat	Class	20	858	77 (8.97%)
	velocity-1.6	Class	20	229	78 (34.06%)
	xalan-2.4	Class	20	723	110 (15.21%)
	xerces-1.3	Class	20	453	69 (15.23%)

5.3 Performance Evaluation Measures

Seven well-known measures are applied to evaluate the prediction performance, including *PD, PF, AUC, G-Measure, MCC, Popt(20%)*, and *IFA*. Other measures such as *Accuracy, Precision*, and *F-Measure* are not used, because they are poor performance indicator for class-imbalanced datasets according to previous SDP studies [64] [65]. *PD* and *PF* have been widely used in previous SDP studies [29], [35], [41], [45]. As Menzies *et al.* [10] stated, an ideal prediction model should have a high *PD* but a low *PF*. *AUC, G-measure*, and *MCC* are three overall performance measures, which take both *PD* and *PF* into consideration and have been widely used in previous SDP studies [45], [46], [65], [66], [67], [68], [69], especially in the case of imbalanced defect datasets. *Popt* and *IFA* comes from effort-aware just-in-time (JIT) defect prediction [70] [71] [72] [73]. Apart from *PF* and *IFA*, for other measures, the larger the value is, the better the prediction performance is. Most of these measures can be defined by confusion matrix [74] as shown in Table 2. By convention, the defective modules are regarded as positive class samples and the non-defective ones as negative class ones [75].

- *PD* (aka. *recall, true positive rate*, or *sensitivity*) and *PF* (aka. *false positive rate*) are defined as follows:

$$PD = \frac{TP}{TP+FN}; PF = \frac{FP}{FP+TN} \tag{14}$$

- *AUC* [76] is the area under of receiver operating characteristic (ROC) curve. This curve is plotted in

TABLE 2: Confusion Matrix

		Predicted	
		Positive	Negative
Actual	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

a 2-D space with PF as the x-axis and PD as the y-axis. The AUC is a widely used measure because it is rarely affected by class imbalance [77]. Its range is [0,1], and AUC=0.5 denotes the performance of a random prediction model.

- *G-Measure* [65] is the harmonic mean of *PD* and (1-*PF*), which has been widely used in previous SDP studies [45], [65], [67], [68]. *G-Measure* is a good indicator of performance for imbalanced datasets. The bigger the value of *G-measure* is, the better the prediction performance is. *G-Measure* is defined as:

$$G - Measure = \frac{(2 * PD * (1 - PF))}{PD + (1 - PF)} \tag{15}$$

- *MCC* [78] is an overall performance measure by taking TP, TN, FP, and FN into consideration. *MCC* has been widely used in previous SDP studies [39], [45], [68], [69] since it can be utilized even the data is unbalanced [74]. The definition of *MCC* is:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{16}$$

- *Popt(20%)* refers to the percentage of detected defective changes to all defective changes when using 20% of effort (i.e., the number of changed code lines) [70]. It is based on the concept of the code-churn-based Alberg diagram.
- *IFA* [73] is the number of initial false alarm encountered until the first true defect is found. According to [73], first sort the predicted defective changes in ascending order based on their *change* size (i.e., the sum of lines of code added and deleted by the current change), if k changes have been inspected when the first actual defective change is found, then *IFA=k*. Obviously, *IFA* does not care how many defective changes can be correctly predicted for a prediction model.

Note that, since the benchmark datasets used in this study have no metrics related to change size, we just use LOC (i.e., lines of code in modules) instead of change size to compute *Popt(20%)* and *IFA*. Furthermore, when all positive instances are misclassified (TP=0) and all negative instances are correctly classified (FP=0), the value of MCC is NAN because of TP+FP=0. To avoid this situation, we just let FP=1, TN=TN-1, and remain TP and FN unchanged. Thus, the value of MCC is a very small negative number for the imbalanced datasets. From Table 1, we can see that the benchmark datasets are highly imbalanced. Therefore, we can directly let MCC=0 when MCC is NAN.

5.4 Statistical Testing

Three statistical testing methods including Wilcoxon signed-rank test (α=0.05) [79], Cliff's δ effect size test (α=0.05) [80], and Scott-Knott effect size difference (ESD) test (α=0.05) [81] are adopted in this study. We use Wilcoxon signed-rank test to clarify whether our KSETE is statistically outperform each baseline on each dataset. Cliff's δ is used to quantify the magnitude of difference between KSETE and each baseline. Scott-Knott ESD test is used to demonstrate that whether our KSETE statistically performs better than the baselines on all datasets.

Specifically, the Wilcoxon signed-rank test at the significance level 0.05 is used to detect whether there is a significant difference in the predictive performance between KSETE and each baseline on each dataset. Wilcoxon signed-rank test is a non-parametric alternative to the paired t-test, but it does not make the assumption that the data must be normally distributed.

Cliff's δ [80] is non-parametric effect size measure with the suggestion of Romano *et al.* [82]: negligible ($|\delta| < 0.147$), small ($0.147 \leq |\delta| < 0.330$), medium ($0.330 \leq |\delta| < 0.474$), and large ($|\delta| \geq 0.474$). Cliff's δ has been widely used in previous SDP studies [27], [68], [83]. In this paper, Cliff's delta is computed according to the method proposed by Macbeth et al. [84].

Scott-Knott ESD test [81] uses hierarchical cluster analysis to partition different treatment means into statistically distinct groups, which has no the overlap problem of other multiple comparison test methods, e.g., Nemenyi's post-hoc test [85] suggested by Demšar [86]. Scott-Knott ESD test is a variant of Scott-Knott test [87] with two corrections: (1) it does not assume that the data is normally distributed; (2) it merges any two statistically different groups that have a negligible effect size into one group according to the Cohen's delta [88]. Both Scott-Knott test and Scott-Knott ESD test have been used in previous SDP studies [8] [89] [90].

5.5 Experimental Settings

5.5.1 Validation Method

A total of 22 datasets from three communities including AEEEM (5 projects), JIRA (7 projects), and PROMISE (10 projects) are used as the benchmark. For WPDP, *k*-fold cross-validation [91] is one of the most commonly used model validation techniques. However, it is unsuitable for CPDP scenario because the source data (training data) and target data (test data) come from different projects. Model validation technique used in this study is presented as follows:

For HDP scenario, if one dataset is selected as the source data from a community, the target data must be selected from the reminding communities. In this way, there are a total of 310 (5*17+7*15+10*12) possible combinations for these 22 projects from three communities. Given a combination, e.g., EQ⇒ant-1.7, the simplest way to validate a given model is that train this model on EQ and then test on ant-1.7 for only a single run. However, this is very possible to make an excessively pessimistic or optimistic evaluation of the performance of the given model on ant-1.7. To alleviate this potential problem and to overcome the inherent randomness in our KSETE (e.g., SMOTE), the settings of our validation method are: (1) as done in previous HDP study [41], we first randomly select 90% instances of the source data as training data to train the model and then test on the target data, since 90% instances of the source dataset can not only ensure the training data is not always the same, but also 90% instances can represent the source data well to some extent; (2) we repeat this procedure 30 times (30 is the minimum needed samples to satisfy the central limit theorem [36]) and report the average value on each target dataset.

For CPDP-CM scenario, if one dataset is selected as the source data from a community, then the target data must be selected from the same community. In this way, we can obtain 152 (5*4+7*6+10*9) possible combinations. And then we take the same validation procedure as done in HDP scenario to report the average results for each model on each target data.

5.5.2 Parameter Settings

In experiments, for the parameters of Algorithms 1 and 2, we set k=1, β=1, M=5, and K=size(source, 2)-1 where size(source, 2) denotes the number of metrics in the original source dataset, respectively. SMOTE is used on the source data to alleviate the class-imbalance problem. For the processed source data, the ratio of the defective modules to the non-defective ones is about 0.85. Moreover, in Eq (12), we need to calculate some kernel matrices. To take advantage of multiple kernel learning [92], [93], we construct a hybrid kernel function based on multiple Gaussian kernel functions, i.e., $K(x_i, x_j) = exp\left(-\frac{\|x_i-x_j\|^2}{2\sigma^2}\right)$. Given a set of positive number $w = \{w_i\}_{i=1,\cdots,m}$ and a dataset $X \in R^{n*d}$ where m is the number of Gaussian kernel functions, n denotes the number of instances, and d represents the number of features, then the kernel matrix of X, i.e. $K(X, X)$, is calculated as follows:

$$\begin{aligned} K(X,X) &= \frac{1}{m}\sum_{i=1}^{m} K_i(X,X) \\ &= \frac{1}{m}\sum_{i=1}^{m} exp\left(\frac{-dist(X^{\mathrm{T}})}{w(i)\cdot sum(var(X))}\right) \end{aligned} \tag{17}$$

where $dist(X^{\mathrm{T}})$ returns an n-by-n of matrix A (A_{ij} denotes the Euclidean distance between i-th instance and j-th instance of X), $var(X)$ returns a 1-by-d vector in which j-th element is the variance of j-th column (feature) of X, and $sum(\cdot)$ is a sum function. In this paper, we let m=3 and w=$\{0.5, 1, 1.5\}$.

Logistic regression (LR) [94] is chosen as the base learner in this paper because it has been widely used in previous SDP studies [1], [38], [83], [95], [96]. Furthermore, LR is not only very simple, but it performs well compared with the complex modeling techniques for SDP [5]. Given an unlabeled instance, a trained LR can give its probability of being positive class. In this study, we set the threshold as 0.5, i.e., if the probability of being positive class is not smaller than the threshold, we label this instance as the positive class.

6 Experimental Results

6.1 RQ1: Does KSETE outperform previous HDP methods?

Table 3 shows the mean results of *PD* and *PF* for each method on each project. From this table, we can see that the *PD* of our KSETE varies from 0.366 to 0.70, the *PF* varies from 0.183 to 0.431. On average, our KSETE obtains the *PD* and *PF* as 0.540 and 0.286, respectively. Although CTKCCA achieves the best average *PD* (0.838), it makes this at a cost of extremely high *PF* (0.844). Compared with HDP-KS and CCA+, KSETE greatly improves *PD* as the cost of an acceptable average PF (0.286). Figure 2 further graphically visualize the average results of *PD* and *PF* for our method and all baseline methods on overall 22 projects.

Tables 4 and 5 respectively present the comparison results of three overall measures (i.e., *AUC*, *G-Measure*, and *MCC*) and two effort-aware measures (i.e., *Popt(20%)* and *IFA*) for our KSETE and all the baselines (three existing HDP methods) on each project. The best result is in boldface. The results of a method which has no significant difference (by Wilcoxon signed-rank test at significance level 0.05) compared with the best method having the best result is marked by ✓. The row 'Average' reports the average results of each method across overall 22 projects, and the row 'Median' the median results. The row 'Sig. Win/Tie/Lose' reports the number of projects for which our method achieves a better, equal, and worse performance than the corresponding baseline method with statistical significance according to the Wilcoxon signed-rank test. In particular, a table cell in deep gray background denotes that our method makes a *large* significant improvement compared with the corresponding baseline method and a table cell in the light gray background indicates a *moderate* significant improvement in terms of Cliff's δ as described in Section 5.4.

From Tables 4 and 5, we can see that our KSETE achieves high performance on each project in terms of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA*. The *AUC* of KSETE varies from 0.511 to 0.780, the G-Measure varies from 0.416 to 0.640, *MCC* varies from 0.062 to 0.364, *Popt(20%)* ranges from 0.143 to 0.438, and *IFA* ranges from 1.2 to 57.8. Especially, KSETE nearly always significantly outperforms all the baselines in terms of *AUC*, *G-Measure*, and *MCC*. Meanwhile, in most cases, KSETE performs significantly better than all the baselines in term of *Popt(20%)* and *IFA* except that KSETE ties with CTKCCA in terms of *Popt(20%)*.

On average, KSETE obtains the *AUC* as 0.691, the *G-Measure* as 0.554, the *MCC* as 0.215, the *Popt(20%)* as 0.279, and the *IFA* as 23. The improvement of KSETE over the baselines on average is *at least* by 22.7%, 138.9%, 494.4%, and 19.7% in terms of *AUC*, *G-Measure*, *MCC*, and *IFA*, respectively. In terms of average *Popt(20%)*, CTKCCA (0.33) closely followed by KSETE (0.279) obtains the best performance.

On the median, KSETE obtains *AUC* as 0.698, the *G-Measure* as 0.562, the *MCC* as 0.208, the *Popt(20%)* as 0.274, and the *IFA* as 22.5. For *AUC*, *G-Measure*, and *MCC*, the improvement of KSETE over the baselines on the median is *at least* by 23.2%, 128.3%, and 414.8%, respectively.

In terms of Cliff's δ, KSETE nearly always largely improves the performance over the baselines in terms of *AUC*, *G-Measure*, and *MCC*. For most cases, KSETE makes a large improvement with statistical significance over HDP-KS and CCA+. KSETE loses on 14 projects compared with CTKCCA and wins on 5 projects with statistical significance in *Popt(20%)*. In most cases, KSETE largely improve the performance over the baselines in terms of *IFA*.

Figure 3 further shows the boxplots of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* for KSETE and all baselines across overall 22 projects. From the figure, we can see that KSETE obviously outperforms or is comparable with the baselines in terms of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA*.

Figure 4(a)-4(e) present the results of Scott-Knott ESD test for the proposed KSETE and previous HDP methods on all the datasets in *AUC*, *G-Meassure*, *MCC*, *Popt(20%)*, and *IFA*, respectively. The *x-axis* shows the HDP methods and the *y-axis* represents the ranking value. Each method corresponds to a vertical line which denotes the range of ranking of this method on all datasets. The dot in the vertical line is the average ranking. Different colors denotes distinct groups with statistical significance. The smaller ranking, the better performance except *IFA*. From these figures, we can see that (1) For *AUC* (Figure 4(a)), *G-Measure* (Figure 4(b)), and *MCC* (Figure 4(c)), KSETE has the smallest average ranking and KSETE is divided into a different group without including any baseline. It means that KSETE significantly outperforms the baselines. (2) For *Popt(20%)* (Figure 4(d)), four methods are divided into four different groups and CTKCCA obtains the smallest ranking followed by our KSETE. (3) For *IFA* (Figure 4(e)), KSETE obtains the biggest average ranking and is divided into a group which has no any baseline.

Based on the above results, we can draw the following conclusion:

The proposed KSETE significantly outperforms the previous heterogeneous defect prediction methods and improves the performance over the baselines on average by at least 22.7%, 138.9%, 494.4%, and 19.7% in terms of *AUC*, *G-Measure*, *MCC*, and *IFA*, respectively. KSETE significantly outperforms the baselines except CTKCCA in terms of *Popt(20%)*.

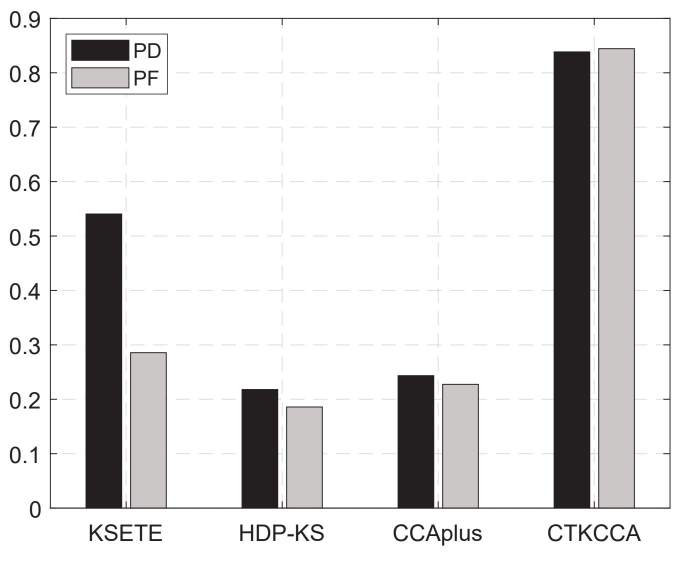


Fig. 2: Barplot of *PD* and *PF* on overall 22 projects

TABLE 3: Comparison results in *PD* and *PF* for the proposed KSETE and existing HDP methods

Target	KSETE		HDP-KS		CCA+		CTKCCA	
	PD	PF	PD	PF	PD	PF	PD	PF
EQ	45.3	18.3	6.1	5.0	28.9	26.5	70.2	72.9
JDT	56.5	19.7	6.0	5.9	18.8	16.8	92.0	91.6
Lucene	57.1	23.2	11.8	11.8	22.7	18.0	89.6	88.8
Mylyn	43.4	20.2	10.7	7.2	16.4	16.5	95.2	95.3
PDE	49.6	26.2	0.4	0.1	18.0	17.7	95.3	94.8
activemq-5.0.0	60.2	23.9	11.3	11.0	23.6	24.0	96.6	95.5
derby-10.5.1.1	50.2	21.4	23.7	17.0	27.0	26.7	97.4	97.0
groovy-1_6_B...	46.0	21.4	12.5	8.5	31.7	30.5	90.8	90.7
hbase-0.94.0	36.6	22.5	10.1	4.8	33.1	33.0	92.6	93.4
hive-0.9.0	40.7	21.3	12.6	8.2	23.8	23.7	94.6	93.9
jruby-1.1	61.3	20.5	9.2	3.7	28.8	26.7	88.2	89.7
wicket-1.3.0-b...	61.1	30.0	24.4	22.1	38.2	38.2	96.3	95.6
ant-1.7	58.7	34.9	33.9	30.2	17.3	14.1	88.0	87.7
camel-1.4	57.1	40.4	27.1	26.2	19.4	17.4	89.4	87.1
ivy-2.0	59.3	34.9	29.3	20.6	23.6	21.0	78.1	75.9
jedit-4.0	53.6	30.7	22.1	19.4	21.3	17.0	74.4	74.8
log4j-1.0	65.7	43.1	14.6	13.0	32.4	30.8	42.0	48.3
poi-2.0	54.1	34.5	54.4	51.4	20.8	20.1	63.2	69.2
tomcat	70.0	37.7	49.3	43.7	16.6	12.7	90.4	87.5
velocity-1.6	38.0	32.5	49.2	44.3	28.8	29.0	60.2	65.9
xalan-2.4	61.7	39.2	28.7	26.9	22.9	20.4	88.1	86.0
xerces-1.3	62.8	31.9	32.0	27.8	21.4	19.6	71.9	75.9
Average	54.0	28.6	21.8	18.6	24.3	22.7	83.8	84.4

All numbers are in **percentage** (%).

6.2 RQ2: Does KSETE outperform previous CPDP-CM methods?

Table 6 shows the comparison results of *PD* and *PF* for each method on each project. Form the table, we notice that the *PD* of KSETE varies from 0.26.4 to 0.691 and the *PF* ranges from 0.135 to 0.395. On average, KSETE achieves the *PD* as 0.50 and the *PF* as 0.239. KSETE largely improves the *PD* compared with the baselines at a cost of tolerable performance loss on *PF*. Figure 5 further graphically visualizes the average results of *PD* and *PF* for all methods by using bar-plot.

Tables 8 and 7 respectively present the mean results of three overall measures (i.e., *AUC*, *G-Measure*, and *MCC*) and two effort-aware measures (i.e., *Popt(20%)* and *IFA*) for our KSETE and all baselines (i.e., five existing CPDP-CM methods) on each project. The best result is in boldface. The row 'Average' reports the average results of overall 22 projects, and the row 'Median' the median results. The row 'W/T/L' reports the number of projects for which our method achieves a better, equal, and worse performance than the corresponding baseline method with statistical significance ($p\text{-}value<0.05$ by the Wilcoxon signed-rank test).

From Tables 8 and 7, we can see that KSETE achieves high performance in terms of *AUC*, *G-Measure*, *MCC*, and *Popt(20%)*. The *AUC* of our KSETE varies from 0.513 to 0.807, the *G-Measure* varies from 0.372 to 0.699, the *MCC* varies from 0.102 to 0.368, the *Popt(20%)* varies from 0.123 to 0.410, and the *IFA* varies from 1.0 to 56.3.

On average, KSETE obtains the *AUC* as 0.691, the *G-Measure* as 0.543, the *MCC* as 0.224, the *Popt(20%)* as 0.259, and the *IFA* as 19.4. KSETE improves the *AUC*, *G-Measure*, and *MCC* over the baselines by at least 4.5%, 30.2%, and 17.9%, respectively. Except for TNB, KSETE improves the *Popt(20%)* over other baselines by at least 32.1%. KSETE improves the *IFA* over the baselines by at least 22.1% except

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2939303, IEEE Transactions on Software Engineering

TABLE 4: Comparison results in *AUC*, *G-Measure*, and *MCC* for the proposed KSETE and existing HDP methods. The best value is in boldface.

Target	AUC				G-Measure				MCC			
	KSETE	HDP-KS	CCA+	CTKCCA	KSETE	HDP-KS	CCA+	CTKCCA	KSETE	HDP-KS	CCA+	CTKCCA
EQ	**72.3**	51.4	51.2	53.8	**52.2**	3.4	24.9	39.1	**30.9**	1.8	3.7	-3.0
JDT	**74.7**	56.6	51.0	57.8	**61.7**	0.2	28.6	15.4	**36.4**	0.3	2.7	0.5
Lucene	**73.5**	68.1	52.3	52.4	**59.3**	0.0	31.9	20.0	**24.1**	0.0	4.1	0.8
Mylyn	**67.2**	57.1	49.9	51.7	**51.5**	7.6	26.1	8.9	**20.1**	5.0	-0.1	-0.2
PDE	**67.5**	59.0	50.1	52.7	**53.1**	0.6	27.5	9.8	**19.2**	0.3	0.4	0.8
activemq-5.0.0	**78.0**	47.0	49.8	52.3	**62.0**	0.8	24.3	8.6	**31.6**	0.3	-0.5	2.0
derby-10.5.1.1	**72.4**	61.7	50.1	51.2	**54.1**	13.0	23.4	5.8	**24.9**	5.5	0.2	0.9
groovy-1_6_BETA_1	**71.7**	68.9✓	50.6	49.7	**51.5**	6.5	15.3	16.8	**17.8**	4.8	0.8	0.0
hbase-0.94.0	**62.9**	58.4	50.1	50.3	**41.6**	11.6	20.9	12.4	**14.3**	5.5	0.3	-1.3
hive-0.9.0	**70.9**	59.0	50.1	51.7	**43.7**	9.8	25.4	11.5	**21.1**	7.9	0.3	1.2
jruby-1.1	**75.8**	49.8	51.0	51.0	**63.2**	7.9	17.0	18.5	**33.7**	5.4	2.4	-1.7
wicket-1.3.0-beta2	**72.1**	58.5	50.0	51.1	**54.7**	4.0	13.3	8.4	**20.6**	1.9	-0.2	0.9
ant-1.7	**66.5**	57.2	51.6	62.9	**58.1**	8.5	26.0	21.6	**21.8**	5.2	3.9	0.3
camel-1.4	**62.5**	50.8	51.0	58.7	**55.1**	18.6	26.4	22.6	**13.6**	0.7	2.4	2.6
ivy-2.0	**67.2**	55.5	51.3	58.4	**59.8**	16.5	25.4	36.8	**17.2**	7.9	2.8	1.6
jedit-4.0	**66.9**	56.7	52.2	59.1	**57.0**	6.2	24.2	37.6	**21.5**	4.0	5.7	-0.5
log4j-1.0	**68.7**	55.5	50.8	50.1	**56.4**	3.2	18.2	46.1	**21.3**	3.3	2.0	-5.6
poi-2.0	**68.2**	52.3	50.3	54.9	**56.7**	13.1	22.8	41.4	**13.9**	3.3	1.3	-4.3
tomcat	**71.8**	57.5	51.9	64.4	**64.0**	9.9	25.9	22.0	**20.1**	6.3	3.2	2.6
velocity-1.6	51.1✓	**53.6**	49.9	50.9✓	**45.8**	14.6	17.4	43.5	**6.2**	4.8✓	0.0	-5.8
xalan-2.4	**67.4**	50.3	51.2	61.1	**55.9**	18.7	25.2	24.2	**17.5**	1.4	2.6	2.2
xerces-1.3	**71.4**	54.0	50.9	55.4	**60.5**	16.7	19.7	36.1	**25.9**	4.1	2.3	-3.6
Average	69.1	56.3	50.8	54.6	55.4	8.7	23.2	23.1	21.5	3.6	1.8	-0.4
Improvement of Average		22.7%	36.1%	26.6%	—	536.3%	138.9%	140.2%	—	494.4%	1075.4%	5034.4%
Median	69.8	56.6	50.8	52.6	56.2	8.2	24.6	20.8	20.8	4.1	2.2	0.4
Improvement of Median		23.2%	37.3%	32.8%	—	584.8%	128.3%	170%	—	414.8%	869.8%	5112.5%
Sig. Win/Tie/Lose		20/2/0	22/0/0	21/1/0	—	22/0/0	22/0/0	22/0/0		21/1/0	22/0/0	22/0/0

All numbers are in **percentage** (%) except row 'Sig. W/T/L'. Checkmark ✓ denotes that the corresponding method has no significant difference compared with the method having the best value. A table cell in deep gray background ■ indicates KSETE makes a large ($|\delta| \geq 0.474$) significant ($p\text{-}value$<0.05) improvement over the corresponding baseline and the light gray background □ indicates a moderate ($0.330 \leq |\delta| < 0.474$) significant improvement.

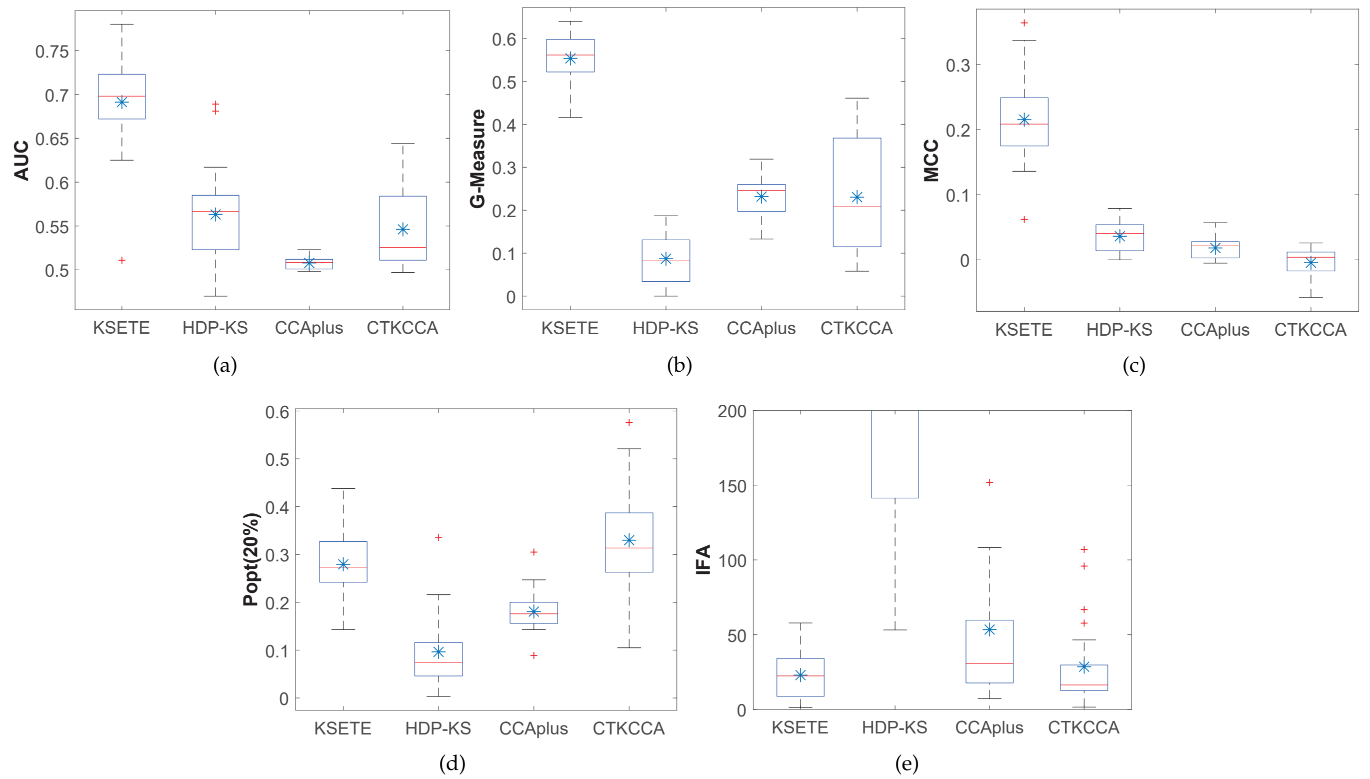


Fig. 3: Boxplots of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* across all datasets for KSETE and other HDP methods. The horizontal line in the box denotes the median, while the asterisk indicates the mean.

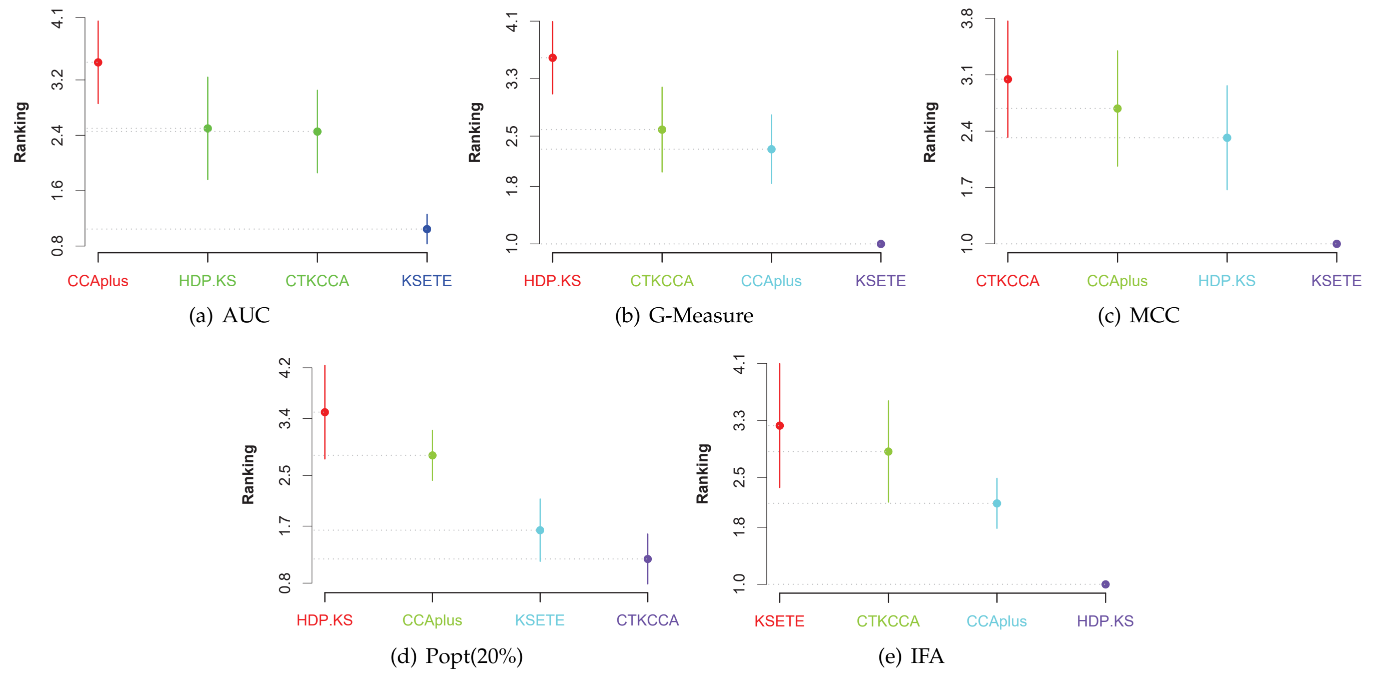


Fig. 4: The results of Scott-Knott ESD test in *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* across all datasets for KSETE and other HDP methods. The smaller ranking, the better performance except *IFA*.

TABLE 5: Comparison results in *Popt(20%)* and *IFA* for the proposed KSETE and existing HDP methods. The best value is in boldface.

Target	Popt				IFA			
	KSETE	HDP-KS	CCA+	CTKCCA	KSETE	HDP-KS	CCA+	CTKCCA
EQ	38.0	5.6	30.5	**57.6**	**1.2**	94.6	16.2	6.7
JDT	33.7	2.4	20.7	**37.1**	**2.4**	707.0	8.5	15.1
Lucene	36.9	4.6	18.9	**37.2**	26.9	535.4	**24.9**	26.5
Mylyn	24.3	5.0	15.3	**32.9**	20.6	976.6	7.2	**3.4**
PDE	**25.6**	0.3	15.6	25.1	**1.7**	1254.2	16.9	4.2
activemq-5...	**28.1**	3.1	16.2	26.3	**10.2**	1298.6	17.7	14.3
derby-10...	24.5	8.2	18.0	**29.9**	57.8	1624.5	30.3	**18.6**
groovy-1...	17.1	5.8	15.0	**30.2**	**32.9**	531.7	151.8	95.9
hbase-0.94.0	22.7	6.9	20.8	**32.5**	34.1✓	637.9	59.7	**29.7**
hive-0.9.0	26.6	8.0	21.0	**39.5**	37.3	535.5	15.8✓	**9.7**
jruby-1.1	21.1	4.1	16.4	**26.9**	**54.8**	525.0	108.2	107.0
wicket-1...	**18.5**	3.6	8.9	10.5	**32.9**	1148.8	286.6	66.8
ant-1.7	**28.6**	11.6	19.3	27.0	**34.6**	265.0	39.3	46.5
camel-1.4	31.8	14.3	16.7	**40.5**	27.8	382.9	**23.1**	29.7
ivy-2.0	14.3	8.2	15.7	**27.2**	**16.6**	183.0	44.5	22.6
jedit-4.0	32.7	10.2	20.0	**38.7**	**11.7**	141.4	30.3✓	13.7
log4j-1.0	**35.5**	6.9	18.5	21.1	**8.8**	77.9	27.1	15.1
poi-2.0	31.9✓	20.9	15.3	**32.5**	19.5	96.5	31.2✓	**13.4**
tomcat	24.2✓	15.6	14.3	**24.6**	**24.3**	303.4	88.9	57.7
velocity-1.6	29.9	33.6	24.7	**52.1**	**8.3**	53.2	33.9	12.7
xalan-2.4	25.0✓	11.6	17.2	**26.1**	38.2	336.9	52.7✓	**17.7**
xerces-1.3	43.8	21.6	18.4	**50.2**	2.4	139.3	60.8	**1.6**
Average	27.9	9.6	18.1	33	23	538.6	53.4	28.6
Improvement of AVG		189.9%	54.7%	-15.3%	—	95.7%	57%	19.7%
Median	27.4	7.5	17.6	31.4	22.5	454	30.8	16.4
Improvement of MED		267.1%	55.4%	-12.8%	—	95.1%	27%	-36.9%
Sig. Win/Tie/Lose		21/1/0	18/4/0	5/3/14	—	21/1/0	14/3/5	13/2/7

All numbers are in **percentage** (%) except row 'Sig. W/T/L' and the results in *IFA*. Checkmark ✓denotes that there is no significant difference compared with the model having best value.

Burak Filter.

On median, KSETE achieves the *AUC* as 0.686, the *G-Measure* as 0.532, the *MCC* as 0.211, the *Popt(20%)* as 0.25, and the *IFA* as 13.3. KSETE improves the *AUC*, *G-Measure*, and *MCC* over the baselines by at least 1.9%, 13.2%, and 12.3%, respectively. In terms of *Popt(20%)*, except for TNB, KSETE outperforms other baselines by at least 29.6%. In terms of *IFA*, KSETE improves the baselines by at least 28.1% except Burak Filter and Bellwether.

Figure 3 further shows the boxplots of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* for our method and all the baselines across overall 22 projects. From this figure, we can see that KSETE outperforms all the baselines in terms of *AUC*, *G-Measure*, and *MCC* on average or median. In terms of *Popt(20%)*, KSETE outperforms all baselines except TNB on average or median. In terms of *IFA*, KSETE performs worse than Burak Filter but outperforms other baselines on average.

Figure 7(a)-7(e) present the results of Scott-Knott ESD test for the proposed KSETE and previous HDP methods on all the datasets in *AUC*, *G-Meassure*, *MCC*, *Popt(20%)*, and *IFA*, respectively. The *x-axis* shows the HDP methods and the *y-axis* represents the ranking value. Each method corresponds to a vertical line which denotes the range of ranking of this method on all datasets. The dot in the vertical line is the average ranking. Different colors denotes distinct groups with statistical significance. The smaller ranking, the better performance except *IFA*. From these figures, we can see that (1) For *AUC* (Figure 7(a)), *G-Measure* (Figure 7(b)), and *MCC* (Figure 7(c)) KSETE obtains the smallest average ranking and is categorized into a group without including any baseline. (2) For *Popt(20%)* (Figure 7(d)), KSETE is categorized into a group without including any baseline and obtains the second smallest average ranking. (3) For *IFA* (Figure 7(e)), Bellwether obtains the biggest average ranking closely followed by KSETE which achieves the second largest average ranking.

Based on the above results, we can draw the following conclusion:

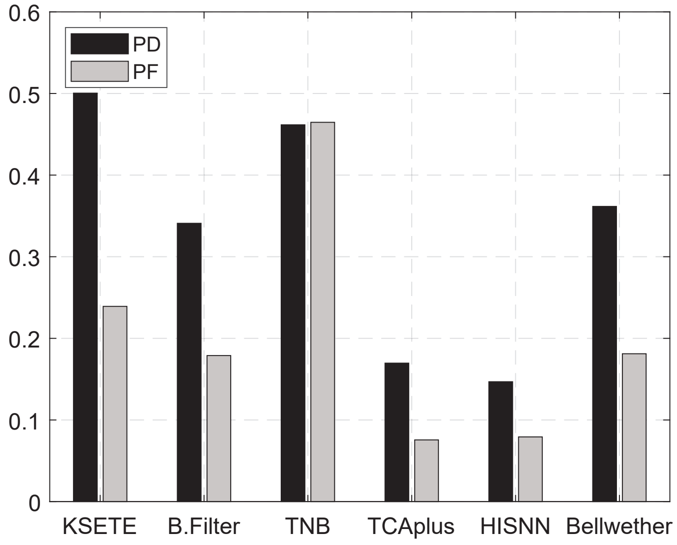


Fig. 5: Barplots of average *PD* and *PF* on overall 22 projects

The proposed KSETE is also suitable for cross-project defect prediction using common metrics (CPDP-CM) scenario. KSETE significantly outperforms the previous CPDP-CM methods and improves the performance over the baselines on average by at least 4.5%, 30.2%, and 17.9%, in terms of *AUC*, *G-Measure*, and *MCC* across all the 22 projects. In terms of *Popt(20%)* and *IFA*, KSETE outperforms the baselines except TNB and Bellwether, respectively.

7 Discussion

7.1 Why KSETE works?

This section answers this question from two perspectives: architecture and empirical analysis.

Architecture perspective. The good performance of our proposed KSETE can be attributed to three main aspects: (1) class imbalance learning, (2) (multiple) kernel spectral embedding, (3) transfer ensemble learning.

Software defect dataset is usually imbalanced, where the number of non-defective (majority class) instances is much more than that of defective (minority class) ones [97]. Class imbalance is a major reason accounting for the poor predicting performance on the minority class instances (defective modules) [75] [98]. The class-imbalance problem has been widely studied in the WPDP scenario [45], [99], [99], [100]. Class imbalance learning methods can be divided into two categories: data-level (such as random oversampling, random under-sampling, and SMOTE) and algorithm-level methods (such as cost-sensitive learning and ensemble learning). Actually, it is also a key problem needed to be considered in the scenario of HDP [41]. However, most previous HDP methods [38], [39], [40] did not take it into consideration, except for CTKCCA [41] which uses cost-sensitive learning for addressing the class-imbalance problem. Differing from CTKCCA, we use SMOTE to preprocess the imbalanced source data. We find that if we perform SMOTE on the projected source data, the prediction performance of KSETE obviously decreases compared with the case that performing SMOTE directly on the source data.

We take the kernel method into consideration for considering the complex nonlinear relationship of the metrics between the source and target datasets and the complex nonlinear relationship of metrics and defects. To take advantage of multiple kernel learning, a hybrid kernel function based on Gaussian kernel function is constructed and used in this paper. To find the optimal projected feature space, we maximize the distribution similarity between the kernel projected source and target datasets and simultaneously preserve the intrinsic characteristics of the source and target datasets maximally. However, previous HDP methods either do not apply kernel method such as [38], [39] or just use the simple kernel function such as [40], [41]. Moreover, previous HDP methods just directly try to maximize the distribution similarity without considering preserving the intrinsic characteristics of the original datasets. If too much useful information of original datasets is lost during finding the common feature space, then the final common feature space cannot perform very well.

Previous HDP methods [38], [39], [40], [41] are all the single transfer learning method, which try to find the latent common feature space by utilizing one time of feature space transformation on the original datasets. However, the proposed KSETE is an ensemble-based transfer learning method, which attempts to construct a series of latent common feature space on multiple kernel subspace of original datasets, and then combines these latent common feature space. It is very challenging to make two different distribution sufficiently similar. It is more possible to find the optimal common feature space from multiple feature space compared with the case that from only one feature space. The results of Section 7.3 can help to illustrate this viewpoint intuitively.

Empirical perspective. KSETE includes three key components: SMOTE, (multiple) kernel learning, and transfer ensemble learning (TE). Now we plan to investigate what is the effect of each component on the performance of KSETE. To this end, we need first to determine the benchmark and the baselines. We just select three typical datasets (i.e., *Lucene*, *hbase-0.94.0*, and *poi-2.0*) as benchmark, since (1) they come from three different communities; (2) their size N (the number of modules) include big ($N \geq 1000$ for *hbase-0.94.0*), medium ($N \geq 500$ for *Lucene*), and small ($N < 500$ for *poi-2.0*); (3) their defective rate (DR) vary from high ($DR \geq 0.2$ for *hbase-0.94.0*) to medium ($0.1 \leq DR < 0.2$ for *poi-2.0*) and small ($DR < 0.1$ for *Lucene*). Thus we can obtain six combinations, e.g., Lucene⇒poi-2.0, according to Section 5.5.1. We then construct six baselines as shown in Table 9. For each combination, we also select 90% instances from the source data as training data to train the model and then use it on the target data. In this way, we run each method 30 times and take the average results as the performance in this combination. Finally, we report the average results of KSETE and six baselines on the above six combinations in terms of *AUC*, *G-Measure*, *MCC*, and *Popt(20%)*.

Figure 8 presents the boxplots of *AUC*, *G-Measure*, *MCC*, and *Popt(20%)* for KSETE and all the baselines. From the figure, we can see that (1) KSE-TE has similar performance to

TABLE 6: Comparison results in *PD* and *PF* for the proposed KSETE and existing CPDP-CM methods

Target	KSETE		Burak Filter		TNB		TCA+		HISNN		Bellwether	
	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF
EQ	50.4	18.9	19.8	14.0	66.3	66.3	14.2	4.1	9.6	2.9	38.1	20.6
JDT	66.4	24.7	32.7	19.0	60.2	61.7	44.8	30.4	29.4	22.4	49.5	49.9
Lucene	69.1	31.6	30.6	17.1	29.2	31.6	17.4	10.4	21.6	10.7	21.9	10.8
Mylyn	53.5	24.6	46.4	37.3	52.9	51.6	16.6	9.2	30.0	24.6	30.5	7.5
PDE	61.1	30.9	28.1	19.5	61.2	60.4	17.5	11.2	24.9	17.0	42.0	12.2
activemq-5.0.0	64.3	20.6	31.3	13.9	39.9	40.8	0.2	0.0	10.0	4.9	67.7	66.7
derby-10.5.1.1	49.0	15.7	25.4	19.0	58.5	58.7	10.2	3.0	14.1	6.1	3.6	0.8
groovy-1_6_B...	43.6	16.1	35.7	12.6	42.3	42.4	6.4	1.1	16.3	6.0	24.6	3.1
hbase-0.94.0	35.8	18.1	45.3	17.8	52.1	51.3	7.4	2.4	23.7	14.9	49.6	41.3
hive-0.9.0	37.1	14.3	19.3	6.2	35.8	37.0	6.3	0.7	12.0	9.0	6.1	1.0
jruby-1.1	58.2	13.5	54.5	24.8	31.7	33.9	11.3	1.1	27.2	13.0	43.7	3.6
wicket-1.3.0-beta2	58.6	19.4	50.4	25.7	17.3	17.8	9.5	2.7	19.6	6.2	0.0	0.0
ant-1.7	47.7	27.2	39.5	18.6	52.9	52.7	21.1	7.9	9.4	3.6	57.8	11.5
camel-1.4	40.6	26.2	22.7	16.3	47.4	46.2	9.9	5.0	5.2	4.0	9.1	3.5
ivy-2.0	47.8	26.5	47.4	19.3	48.1	49.8	31.1	9.6	12.8	4.7	67.2	40.0
jedit-4.0	39.7	25.6	41.5	20.9	53.6	50.3	23.2	9.1	11.2	3.3	48.7	27.2
log4j-1.0	58.8	39.5	16.4	5.0	40.6	40.5	9.2	2.3	6.4	2.4	14.7	3.0
poi-2.0	42.0	27.5	39.5	22.2	46.3	45.7	23.3	10.8	8.3	3.1	36.1	12.9
tomcat	55.2	29.2	31.1	11.5	45.7	46.4	35.0	13.7	10.1	3.8	62.3	23.3
velocity-1.6	26.4	18.5	20.8	12.6	42.4	43.2	10.5	6.4	5.1	3.1	34.9	24.2
xalan-2.4	51.4	33.2	41.6	21.6	54.0	53.9	27.7	12.5	10.5	5.7	52.7	18.6
xerces-1.3	44.2	24.4	30.1	18.7	37.2	40.1	20.4	12.7	5.7	3.0	34.8	16.8
Average	50.0	23.9	34.1	17.9	46.2	46.5	17	7.6	14.7	7.9	36.2	18.1

All numbers are in **percentage** (%).

TABLE 7: Comparison results in *AUC*, *G-Measure*, and *MCC* for the proposed KSETE and existing CPDP-CM methods. The best value is in boldface.

Target	AUC						G-Measure						MCC					
	KSETE	B.Filter	TNB	TCA+	HISNN	Bellwether	KSETE	B.Filter	TNB	TCA+	HISNN	Bellwether	KSETE	B.Filter	TNB	TCA+	HISNN	Bellwether
EQ	**72.3**	54.2	50.7	64.4	52.6	64.7	**61.9**	29.5	36.2	24.1	17.0	51.5	**33.1**	6.7	0.6	19.3	14.1	19.2
JDT	**77.6**	63.6	49.3	58.4	52.0	42.7	**69.9**	31.5	43.6	49.4	31.9	49.8	**36.8**	16.6	-1.3	15.6	8.9	-0.3
Lucene	**76.1**	58.0	49.9	64.6	54.9	63.6	**68.4**	36.7	40.1	22.8	28.0	35.1	**23.5**	11.5	-1.5	9.0	10.7	10.1
Mylyn	68.6	57.7	50.5	67.4	53.2	**72.0**	**62.4**	38.4	44.9	22.0	30.6	45.9	21.8	8.0	1.1	12.8	6.9	**25.5**
PDE	70.6	63.0	51.3	67.4✓	53.7	**72.6**	**64.1**	33.4	38.1	23.0	24.2	56.8	22.0	10.2	1.0	8.8	12.0	**27.8**
activemq-5.0.0	**80.7**	66.0	49.9	56.0	51.8	50.5	**69.8**	41.7	36.5	0.3	16.6	2.2	**36.7**	19.4	-0.8	1.2	9.1	4.6
derby-10.5.1.1	**74.2**	60.0	49.2	67.1	52.6	71.5	**60.3**	21.9	40.0	14.5	23.1	6.9	**29.3**	9.3	-0.1	10.7	11.4	8.7
groovy-1...	71.2	71.7	49.3	61.6	52.0	**77.9**	**56.9**	47.3	39.1	9.6	25.7	39.3	19.5	18.8	-0.1	7.0	11.2	**27.3**
hbase-0.94.0	65.0	**70.1**	50.3	66.0✓	51.3	54.1	49.7✓	**53.8**	46.1	10.1	31.6	19.1	17.4	**26.6**	0.8	7.1	9.9	7.3
hive-0.9.0	**71.3**	60.6	48.9	69.9✓	47.9	71.1✓	**51.1**	29.8	42.2	10.0	17.9	11.5	**23.6**	22.4✓	-1.0	12.3	10.2	14.3
jruby-1.1	**79.9**	70.2	48.6	58.8	53.2	78.0	**66.6**	57.4	41.3	16.4	37.7	60.1	37.2	26.0	-1.6	19.2	17.1	**47.2**
wicket-1.3...	**75.9**	70.2	51.3	62.0	56.3	50.4	**67.4**	50.7	27.2	11.3	29.4	0.0	**25.2**	16.5	-0.5	4.4	15.1	0.0
ant-1.7	65.2	73.2	49.4	72.0	51.6	**79.3**	45.7	44.2	45.4	29.6	16.0	**69.9**	23.2	24.9	0.4	22.2	12.2	**46.9**
camel-1.4	61.0	62.2✓	49.3	59.3	51.5	**63.0**	40.2✓	28.4	**43.1**	15.7	8.5	16.6	**14.4**	7.3	0.9	9.0	4.1	10.3
ivy-2.0	63.2	**74.9**	49.6	71.1	51.2	63.6	52.4✓	55.0✓	46.4✓	42.4	21.4	**56.8**	17.2	24.6✓	-1.2	**24.9**	12.5	18.3✓
jedit-4.0	64.1	**70.5**	51.9	67.1✓	53.8	60.7	38.5	46.8✓	47.7✓	32.9	19.0	**48.1**	15.8	22.9✓	2.9	18.2	15.1	**24.5**
log4j-1.0	63.5	72.8	51.0	71.5	56.7	**78.8**	**47.9**	24.7	40.1✓	14.8	11.0	25.5	18.5✓	17.9✓	-0.2	13.6	10.7	**21.4**
poi-2.0	**65.3**	63.1	50.6	60.2	54.0	62.6	42.2	42.2	45.8	33.3	14.4	**51.1**	12.8	17.6	0.1	16.2	8.9	**21.2**
tomcat	68.6	71.0	49.5	64.8	51.8	**76.7**	54.0	37.1	45.0	47.4	17.2	**68.8**	18.0	16.7	-0.4	19.3	10.2	**26.0**
velocity-1.6	51.3	**59.4**	48.2	58.4✓	50.5	55.4	37.2✓	27.3	41.4✓	16.1	9.4	**41.7**	10.2✓	**11.3**	-0.8	8.1	5.7	11.2✓
xalan-2.4	65.7	70.7	50.5	68.6	51.4	**74.5**	45.4	46.0	44.2	37.5	17.3	**64.0**	16.6	19.3	0.0	19.1	9.4	**29.2**
xerces-1.3	68.1✓	65.4	49.0	59.3	52.5	**70.0**	43.1✓	36.8	43.8	31.2	10.5	**49.0**	**20.3**	13.0	-2.3	10.3	6.6	17.6
Average	69.1	65.8	49.9	64.4	52.6	66.1	54.3	39.1	41.7	23.4	20.8	39.5	22.4	16.7	-0.2	13.1	10.5	19
Improvement of Average		4.9%	38.4%	7.3%	31.4%	4.5%	—	38.9%	30.2%	132.3%	160.7%	37.4%	—	34.2%	12427.5%	71%	112.5%	17.9%
Median	68.6	65.7	49.8	64.7	52.3	67.4	53.2	37.8	42.7	22.4	18.5	47	21.1	17.2	-0.2	12.6	10.5	18.8
Improvement of Median		4.4%	37.9%	6%	31.3%	1.9%	—	40.9%	24.7%	137.5%	188.3%	13.2%	—	22.7%	14133.3%	67.7%	101.4%	12.3%
Sig. Win/Tie/Lose		11/4/7	22/0/0	11/7/4	21/1/0	10/3/9	—	14/7/1	14/7/1	20/2/0	22/0/0	14/3/5	—	11/7/4	22/0/0	14/7/1	21/1/0	10/3/9

All numbers are in **percentage** (%) except for row 'Sig. W/T/L'. Checkmark ✓denotes that the corresponding method has no significant difference compared with the method having the best value. A table cell in deep gray background ▨ indicates KSETE makes a large ($|\delta| \geq 0.474$) significant ($p\text{-}value$<0.05) improvement over the corresponding baseline and the light gray background ░ indicates a moderate ($0.330 \leq |\delta| < 0.474$) significant improvement.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2939303, IEEE Transactions on Software Engineering

TABLE 8: Comparison results in Popt(20%) and IFA for the proposed KSETE and existing CPDP-CM methods. The best value is in boldface.

Target	Popt(20%)						IFA					
	KSETE	B. Filter	TNB	TCA+	HISNN	Bellwether	KSETE	B. Filter	TNB	TCA+	HISNN	Bellwether
EQ	41.0	19.9	**57.5**	12.2	12.7	18.1	**1.0**	4.5	4.7	1.9	1.2✓	6.0
JDT	34.5	19.5	31.2	28.6	24.3	**36.6**	1.3	5.9	12.3	5.6	7.5	**1.0**
Lucene	**40.4**	19.0	22.0	9.7	16.7	7.2	**2.7**	9.0	17.5	6.3	63.8	7.0
Mylyn	26.9✓	20.9	**27.7**	8.1	17.9	19.5	**1.3**	13.7	4.1	5.5	8.3	**1.3**✓
PDE	**26.6**	16.9	22.5	9.7	13.5	21.1	**1.6**	5.2	4.6	20.6	4.3	2.0✓
activemq-5.0.0	**29.0**	18.8	20.8	0.3	11.3	19.0	11.7	16.4	20.5	1228.5	**7.5**	12.7
derby-10.5.1.1	25.7✓	12.7	**26.1**	6.4	12.6	4.7	20.8	9.8✓	23.2✓	389.7✓	18.9	**7.7**
groovy-1_6_BETA_1	16.6	19.7✓	**21.3**	3.6	13.0	14.4	31.8	16.1	57.2	368.8	86.2	**4.0**
hbase-0.94.0	23.1	**30.4**	30.0✓	4.4	20.8	21.3	39.9	**12.2**	30.7	281.2✓	67.4✓	13.5✓
hive-0.9.0	26.8	20.0	**29.4**	5.7	13.0	8.4	47.5	5.0	9.7	350.2✓	6.5	**2.0**
jruby-1.1	19.8	26.8✓	24.2	8.8	23.2	**28.6**	56.3	62.3	50.9	103.0✓	14.2	**2.0**
wicket-1.3.0-beta2	**20.9**	17.9	12.1	3.6	13.7	0.0	40.8	90.1	79.3	1318.5	**18.1**	1581.0
ant-1.7	25.1	22.5	27.8✓	15.0	12.3	**28.2**	23.9	9.0	32.4	5.8	5.7	**2.0**
camel-1.4	24.3	11.5	**30.3**	7.8	4.2	5.9	14.0	10.4✓	25.9	21.1	129.8✓	**5.0**
ivy-2.0	12.3	18.4	**23.0**	15.8	11.5	22.2✓	26.6	12.9✓	20.6	**8.1**	16.4✓	16.1
jedit-4.0	23.7	27.3	**37.6**	17.5	13.7	24.2	11.3	5.2	15.7	28.2✓	**3.5**	4.2✓
log4j-1.0	**34.0**	18.0	25.5	8.5	8.9	10.4	9.0	18.4✓	12.1	34.6✓	34.5✓	**3.0**
poi-2.0	22.2	19.6	**27.3**	11.2	8.4	15.5	15.8	5.7	13.1	11.9	38.5	**2.8**
tomcat	**22.2**	14.7	20.5	17.1	10.0	14.1	42.0	53.2✓	61.9	**11.1**	47.8✓	26.0
velocity-1.6	24.8	16.9	**41.2**	9.4	6.3	22.9	5.2	2.9✓	10.1	16.8✓	19.8✓	**2.4**
xalan-2.4	21.8	21.5	**26.3**	15.8	10.4	21.4	12.6	8.5✓	29.6	9.9✓	23.1✓	**7.0**
xerces-1.3	28.5✓	18.6	**32.4**	11.7	5.5	16.8	10.3✓	8.3✓	12.3✓	**8.0**	18.9✓	10.0✓
Average	25.9	19.6	28	10.5	12.9	17.3	19.4	17.5	24.9	192.5	29.2	78.1
Improvement of Average		32.1%	-7.5%	146.9%	100.8%	49.9%	—	-11.1%	22.1%	89.9%	33.4%	75.1%
Median	25	19.3	26.8	9.6	12.6	18.5	13.3	9.4	19	18.7	18.5	4.6
Improvement of Median		29.6%	-6.9%	161.3%	97.2%	34.5%	—	-41.5%	30%	28.9%	28.1%	-189.1%
Sig. Win/Tie/Lose		15/3/4	7/3/12	21/0/1	19/3/0	15/3/4	—	5/7/10	15/4/3	10/8/4	9/6/7	3/4/15

All numbers are in **percentage** (%) except row 'Sig. W/T/L' and the results in *IFA*. Checkmark ✓denotes that there is no significant difference compared with the model having best value. A table cell in deep gray background ▨ indicates KSETE makes a large ($|\delta| \geq 0.474$) significant ($p\text{-}value$<0.05) improvement over the corresponding baseline and the light gray background ▨ indicates a moderate ($0.330 \leq |\delta| < 0.474$) significant improvement.

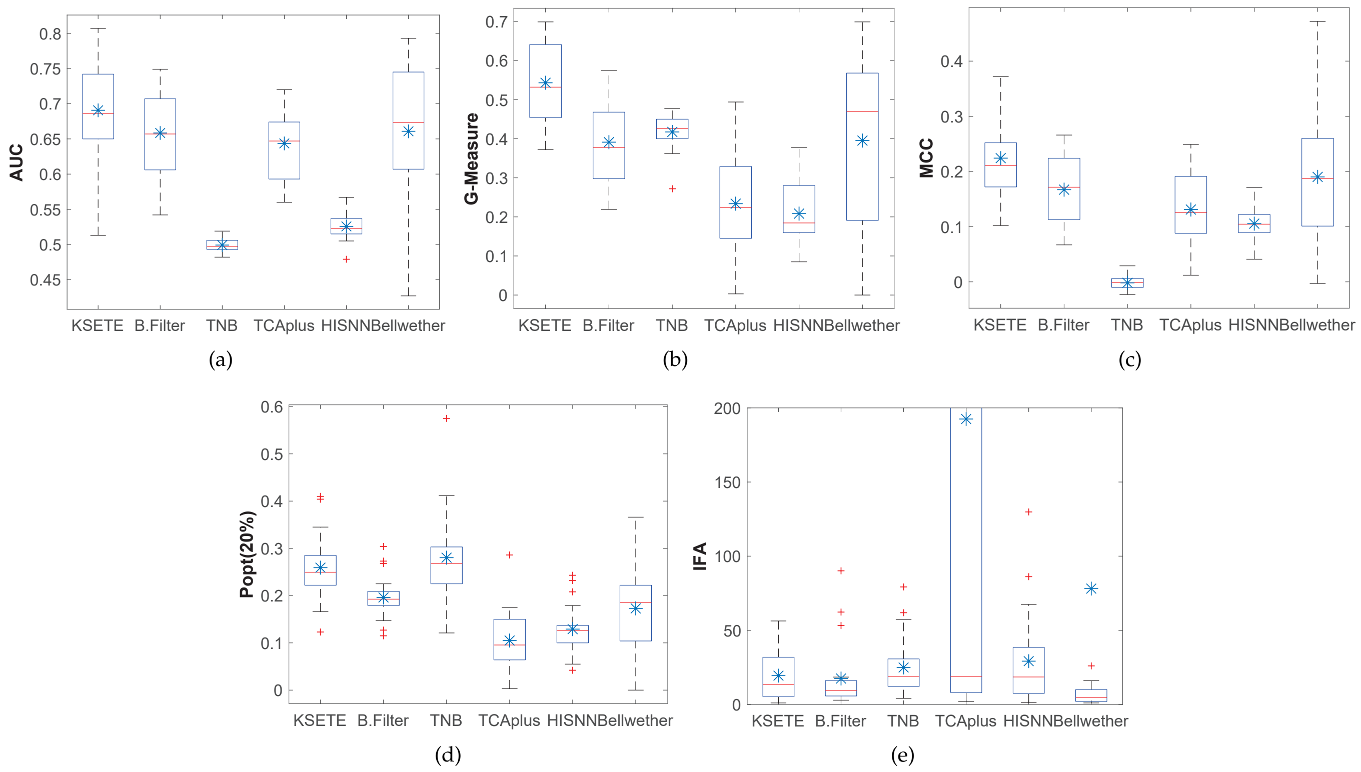
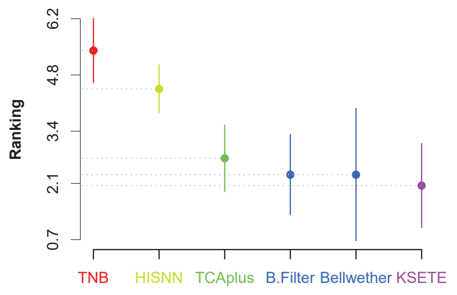
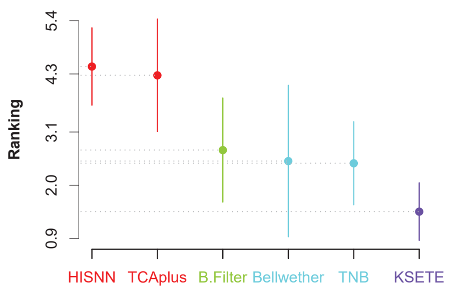


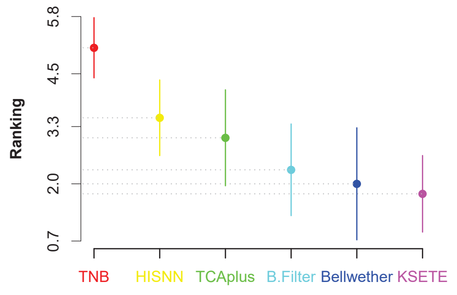
Fig. 6: Boxplots of *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* across all datasets for KSETE and other CPDP-CM methods. The horizontal line in the box denotes the median, while the asterisk indicates the mean.



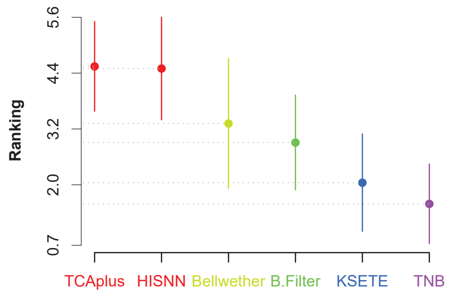
(a) AUC



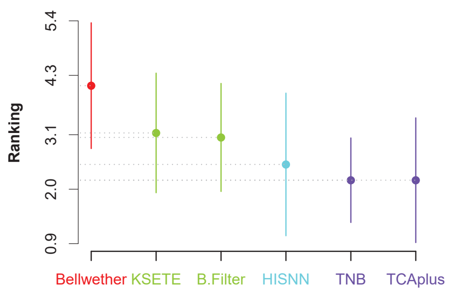
(b) G-Measure



(c) MCC



(d) Popt(20%)



(e) IFA

Fig. 7: The results of Scott-Knott ESD test in *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* across all datasets for KSETE and other CPDP-CM methods. The smaller ranking, the better performance except *IFA*.

KSETE in terms of AUC and MCC, but performs worse than KSETE in terms of G-measure and Popt(20%). This indicates that SMOTE has a positive effect on the performance of KSETE. (2) KSETE outperforms KSE-SMOTE-TE in terms of G-measure and Popt(20%), has similar performance to it in terms of AUC and MCC. This indicates using SMOTE before feature space transformation is better than using SMOTE after that. (3) Methods using kernel function always obviously outperforms the methods without using that. (4) SE almost always performs worst, especially in terms of G-measure and Popt(20%). (5) KSETE outperforms Single_KSETE in terms of AUC, G-Measure, and Popt(20%), and has similar performance in MCC. This indicates that multiple kernels method has a positive effect on the performance. (6) KETSE seems to always outperform KSE in terms of *AUC*, *G-Measure*, *MCC*, and *Popt(20%)*. This indicates the TE has a positive effect on the performance of KSETE.

TABLE 9: Baselines used to investigate why KSETE works

Baseline	Description
KSE-SMOTE-TE	Use SMOTE algorithm after KSE, which is the only difference compared with KSETE. The reminding settings is completely same to KSETE.
KSE-TE	Just without using SMOTE compared with KSETE.
KSE	Just without using transfer ensemble algorithm compared with KSETE.
Single-KSETE	Just use a single kernel compared with multiple kernel in KSETE.
SMOTE-SE	Without using both kernel and transfer ensemble learning compared with KSETE.
SE	Without using SMOTE, kernel, and transfer ensemble learning compared with KSETE. SE refers to the spectral embedding method.

7.2 The effect of β on the performance of KSETE?

By default, β (i.e, the similarity confidence parameter) is set to be 1. To study the effect of different values of β on the performance of the proposed KSETE, we take one prediction combination *Lucene⇒ant-1.7* (i.e., *Lucene* is the source data, *ant-1.7* is the target data) as an example. Specifically, we let β take different values 0.5, 1, 1.5, 2, 2.5, and 3. For each k value, we run KSETE 30 times. For each time, 90% instances of the source dataset are randomly selected as the training data, then the prediction performance of each HDP method is evaluated on the target data. Finally, we report the average PD, PF, G-measure, MCC of KSETE for different β values.

Figure 9 shows the error-bars of PD, PF, G-measure, and MCC of KSETE with different σ values. The x-axis represents the value of β. The y-axis denotes the value of each performance. The dot in the middle of the vertical line denotes the average performance of KSETE across 30 runs on the prediction combination *Lucene⇒ant-1.7*.

From the figure, we notice that (1) if we pay less attention to the distribution similarity between the projected source and target datasets ($D(B_\Phi(S), B_\Phi(T))$ in Eq.(1)), i.e., $\beta = 0.5$, PF,G-measure, and MCC are very poor. (2) if we pay less attention to the loss of intrinsic characteristic of the source and target datasets ($L(B_{\Phi(S)}, \Phi(S))$ and $L(B_{\Phi(T)}, \Phi(T))$ in Eq.(1)). e.g., the case that $\beta = 2.5$, PD, G-measure, and MCC are also poor, especially PD and G-measure. Therefore, in this paper, β is set to 1 for naturally balancing the two objectives.

7.3 The effect of different values of M on the performance of KSETE?

By default, we let the number of base learners in our transfer ensemble algorithm be M=5. To study the effect

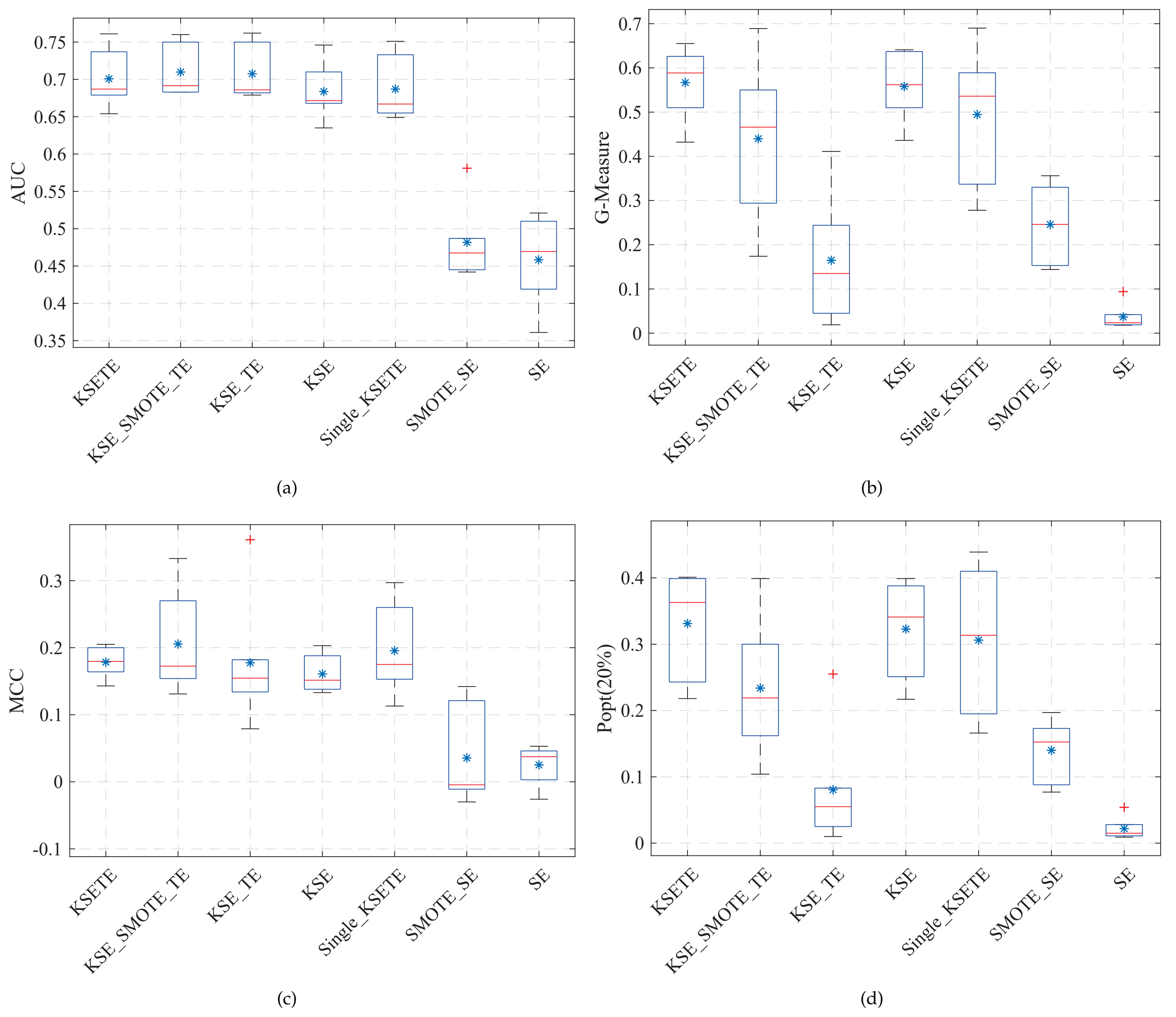


Fig. 8: Boxplots of *AUC*, *G-Measure*, *MCC*, and *Popt(20%)* values for our KSETE and all the baselines (see Table 9). The horizontal line in the box denotes the median, while the asterisk indicates the mean.

of different M values on the performance of the proposed KSETE, we take one prediction combination *Lucene⇒ant-1.7* (i.e., *Lucene* is the source data, *ant-1.7* is the target data) as an example. Specifically, let M take different values 1, 3, 5, 7, 9, and 11. Then, we take the same experiments as that when we study the effect of different β values on the performance of KSETE.

Figure 10 shows the error-bars of PD, PF, G-measure, and MCC of KSETE with different M values. The x-axis represents the value of M. The y-axis denotes the value of each performance. The dot in the middle of the vertical line denotes the average performance of KSETE across 30 runs on the prediction combination *Lucene⇒ant-1.7*.

From the figure, we notice that (1) if we do not utilize the ensemble strategy (i.e., when M=1), PF, G-measure, and MCC obviously are the worst compared with other cases utilizing the ensemble strategy. (2) to some extent (M ranges from 1 to 9), PF, G-measure, and MCC generally increase along with the increasing of M, although PD is relatively stable. (3) When M increases to a certain extent, the performance of KSETE might not further improve even will decrease. We also notice that when M ranges from 5 to 9, KSETE is not very sensitive to M, especially in terms of PD and G-measure. Therefore, M=5 can be utilized as the default setting.

8 Threats to Validity

In this section, some potential threats to the validity of our research are presented.

8.1 Internal Validity

Threats to the internal validity of our study mainly relate to the re-implementation of baselines, the lack of causal

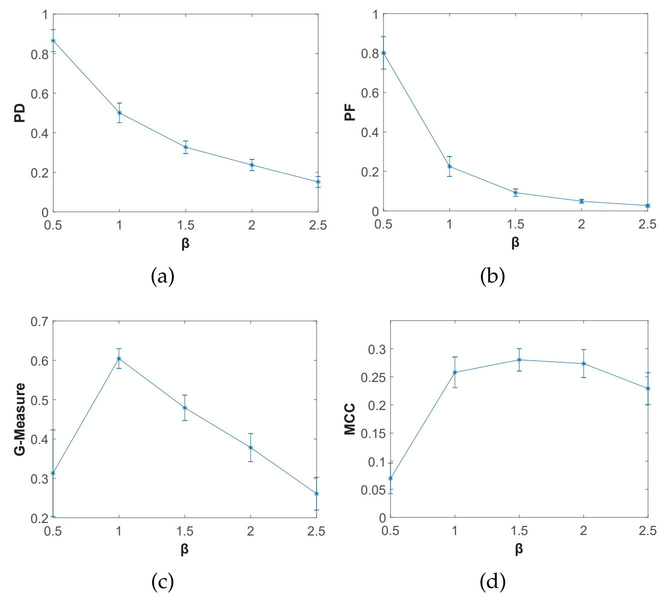


Fig. 9: The error-bars of performance of KSETE with different values of M on the prediction combination *Lucene⇒ant-1.7*. The error-bar represents one standard deviation above and below the mean value.

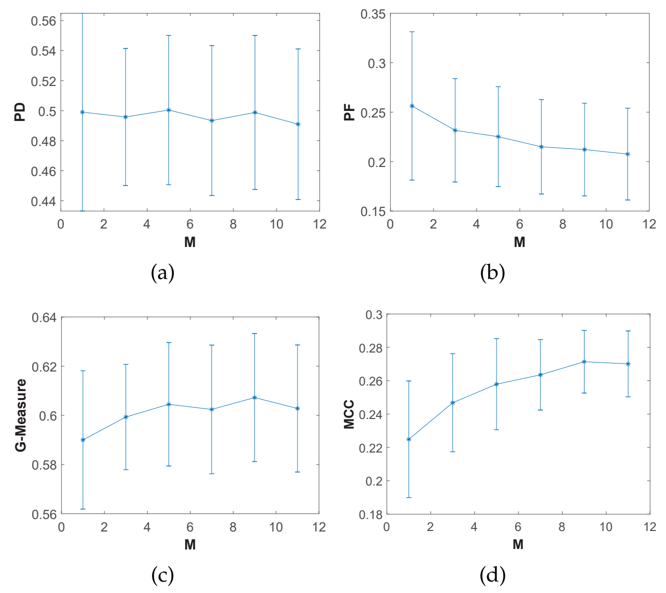


Fig. 10: The error-bars of KSETE with different values of M on the prediction combination *Lucene⇒ant-1.7*. The error-bar represents one standard deviation above and below the mean value.

effect analysis, and the model interpretability bias. Since the source code of all the baselines except for CTKCCA [41] and Bellwether [36] is not publicly opened, we implement the reminding baselines carefully according to the description in corresponding studies. Although we have checked our source code carefully, there still may be errors that we did not notice. Since the causal effect between independent and dependent variables is not the objective of this study, we do not analyze it and related works can be seen in [1], [101]. KSETE achieves good prediction performance in both HDP and CPDP-CM scenarios by feature space transformation. However, it is difficult to illustrate the meanings of new features. All the methods which use feature space transformation such as CCA+ [39], and CTKCCA [41], TCA+ [33], and the deep-learning based methods [83] [34], suffer to this threat. Building prediction models having both high performance and good interpretability is our future work.

8.2 External Validity

External validity indicates the degree to generalize the research results to other situations [32], [102]. To evaluate the performance of KSETE, experiments are performed on 22 public projects from three communities including AEEEM [33], JIRA [43], and PROMISE [44]. We compared KSETE with the five CPDP-CM methods and three HDP methods in terms of seven performance measures (see Section 5.3). The Wilcoxon signed-rank test, Cliff's delta effect size, and Scott-Knott ESD test are also used. However, we still cannot claim that the findings will be completely suitable for other defect datasets. More defect datasets and the baseline methods should be applied to reduce this threat.

8.3 Construct Validity

Threats to construct validity for our study mainly refer to the biases of performance evaluation measures. In this study, seven well-known performance measures including *PD*, *PF*, *AUC*, *G-Measure*, *MCC*, *Popt(20%)*, and *IFA* are used where *Popt(20%)* and *IFA* are two effort-aware measures widely used in JIT defect prediction [26], [70]. However, some other measures, such as *Balance* [10] and *H-measure* [103], are not applied owing to the space limitation, which also have been utilized for the imbalanced defect datasets in previous studies [75] [14]. Meanwhile, since our method is not for JIT defect prediction and the benchmark does not include the change size metrics, we just use LOC instead of change size when computing *Popt(20%)* and *IFA*. Therefore, this study has these threats.

9 Conclusion

Differing from CPDP-CM, HDP aims to predict defects across projects with heterogeneous metric sets.

In this paper, we propose a novel KSETE method for HDP. KSETE first uses SMOTE to alleviate the class-imbalance problem of the source data. And then KSETE tries to find a latent common feature space by our proposed multiple kernel spectral embedding transfer method, which can maximize the distribution similarity between the source and target datasets and can preserve the intrinsic properties of them. Finally, we build an ensemble classifier based on multiple common feature space by our transfer ensemble algorithm and use it to predict the label of unlabeled target data.

Experiments are performed on 22 projects from three community including AEEEM [33], JIRA [43], and PROMISE [44]. The performance of KSETE is evaluated in terms of seven well-known measures including *PD, PF, AUC, G-Measure, MCC, Popt(20%)*, and *IFA*. We compare KSETE with the existing HDP methods and CPDP-CM methods by using the Wilcoxon signed-rank test, Cliff's δ, and Scott-Knott ESD test in HDP scenario and CPDP-CM scenario, respectively. The experimental results show that KSETE is effective in the both HDP and CPDP-CM scenarios.

As future work, we plan to improve our method by using deep learning owing to its powerful feature learning capability.

Acknowledgments

This work is funded by #61400020404. We thank the editors and anonymous reviewers for their constructive comments and suggestions for improving this paper.

References

[1] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, Oct. 2005.

[2] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, May 2009.

[3] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7 – 15, 2009.

[4] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," *Automated Software Engg.*, vol. 17, no. 4, pp. 375–407, Dec. 2010.

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

[6] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, Sept 2013.

[7] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626 – 4636, 2011.

[8] B. Ghotra, S. McIntosh, and 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 - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 789–800.

[9] S. S. Rathore and S. Kumar, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems," *Knowledge-Based Systems*, vol. 119, no. Supplement C, pp. 232 – 256, 2017.

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

[11] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 483–492, May 2007.

[12] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649 – 660, 2008.

[13] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Softw. Pract. Exper.*, vol. 41, no. 5, pp. 579–606, Apr. 2011.

[14] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using bayesian network classifiers," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237–257, 2013.

[15] G. Czibula, Z. Marian, and I. G. Czibula, "Software defect prediction using relational association rule mining," *Information Sciences*, vol. 264, pp. 260 – 278, 2014.

[16] X. Yang, K. Tang, and X. Yao, "A Learning-to-Rank Approach to Software Defect Prediction," *IEEE TRANSACTIONS ON RELIABILITY*, vol. 64, no. 1, pp. 234–246, MAR 2015.

[17] W. Li, Z. Huang, and Q. Li, "Three-way decisions based software defect prediction," *Knowledge-Based Systems*, vol. 91, no. Supplement C, pp. 263 – 274, 2016.

[18] P. Singh, N. R. Pal, S. Verma, and O. P. Vyas, "Fuzzy rule-based approach for software fault prediction," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 5, pp. 826–837, May 2017.

[19] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, and T. Zhang, "Software defect prediction based on kernel pca and weighted extreme learning machine," *Information and Software Technology*, 2018.

[20] G. Abaei, A. Selamat, and H. Fujita, "An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction," *Knowledge-Based Systems*, vol. 74, pp. 28 – 39, 2015.

[21] Z. A. Rana, M. A. Mian, and S. Shamail, "Improving recall of software defect prediction models using association mining," *Knowledge-Based Systems*, vol. 90, pp. 1 – 13, 2015.

[22] M. H. Halstead, *Elements of software science*, ser. Elsevier computer science library : operational programming systems series. New York, NY: North-Holland, 1977.

[23] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec 1976.

[24] C. Kemerer and S. Chidamber, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, 06 1994.

[25] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? an empirical study," *Software Quality Journal*, pp. 1–30, 2014.

[26] S. Kim, E. J. W. Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, March 2008.

[27] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Developer micro interaction metrics for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 11, pp. 1015–1035, Nov 2016.

[28] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 91–100.

[29] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, Oct 2009.

[30] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *Journal of Computer Science and Technology*, vol. 30, 2015.

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

[32] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[33] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 382–391.

[34] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 297–308.

[35] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Softw. Engg.*, vol. 21, no. 1, pp. 43–71, Feb. 2016.

[36] R. Krishna and T. Menzies, "Bellwethers: A baseline method for transfer learning," *IEEE Transactions on Software Engineering*, 2018.

[37] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," 2012.

[38] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 508–519.

[39] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 496–507.

[40] Y. MA, S. ZHU, Y. CHEN, and J. LI, "Kernel cca based transfer learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 100, no. 8, pp. 1903–1906, 2017.

[41] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, and S. Ying, "Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction," *Automated Software Engineering*, vol. 25, no. 2, pp. 201–245, Jun 2018.

[42] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2 – 17, 2010, sI: Top Scholars.

[43] S. Yatish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn, "Mining software defects: Should we consider affected releases?" in *The International Conference on Software Engineering (ICSE)*, 2019.

[44] M. Jureczko and 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*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10.

[45] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, pp. 67 – 77, 2015.

[46] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, Sep. 2018.

[47] D. R. Hardoon, S. R. Szedmak, and J. R. Shawe-taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural Comput.*, vol. 16, no. 12, pp. 2639–2664, Dec. 2004.

[48] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[49] A. Quattoni, M. Collins, and T. Darrell, "Transfer learning for image classification with sparse prototype representations," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[50] X. Shi, Q. Liu, W. Fan, and P. S. Yu, "Transfer across completely different feature spaces via spectral embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 906–918, April 2013.

[51] D. Wang and T. F. Zheng, "Transfer learning for speech and language processing," in *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Dec 2015, pp. 1225–1237.

[52] X. Cao, D. Wipf, F. Wen, G. Duan, and J. Sun, "A practical transfer learning algorithm for face verification," in *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ser. ICCV '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 3208–3215.

[53] T. T. Um, M. S. Park, and J. Park, "Independent joint learning: A novel task-to-task transfer learning scheme for robot models," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 5679–5684.

[54] E. Grolman, A. Bar, B. Shapira, L. Rokach, and A. Dayan, "Utilizing transfer learning for in-domain collaborative filtering," *Know.-Based Syst.*, vol. 107, no. C, pp. 70–82, Sep. 2016.

[55] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.

[56] J. Huang, Y.-F. Li, and M. Xie, "An empirical analysis of data preprocessing for machine learning-based software cost estimation," *Information and Software Technology*, vol. 67, pp. 108–127, 2015.

[57] R. Bhatia, *Matrix analysis*, ser. Graduate texts in mathematics. New York, NY: Springer, 1997.

[58] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," in *Proceedings of the Symposium on Spectral Theory and Differential Problems*. Oklahoma Agricultural and Mechanical College, Stillwater, Okla., 1951, pp. 301–316.

[59] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[60] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the nasa software defect data sets," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '16. New York, NY, USA: ACM, 2016, pp. 13:1–13:5.

[61] M. Wen, R. Wu, and S. C. Cheung, "How well do change sequences predict defects? sequence learning from software changes," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[62] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Softw. Engg.*, vol. 17, no. 4-5, pp. 531–577, Aug. 2012.

[63] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 28, pp. 4–17, 2002.

[64] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 33, no. 9, pp. 637–640, SEP 2007.

[65] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1054–1068, Aug. 2013.

[66] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 309–320.

[67] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct 2013, pp. 45–54.

[68] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, Oct 2016.

[69] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, "Mutation-aware fault prediction," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: ACM, 2016, pp. 330–341.

[70] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[71] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 157–168.

[72] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, 2017, pp. 72–83.

[73] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 159–170.

[74] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, June 2014.

[75] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, June 2013.

[76] A. P. Bradley, "The use of the area under the roc curve in

the evaluation of machine learning algorithms," *Pattern Recogn.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[77] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, jan 2018.

[78] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica Et Biophysica Acta*, vol. 405, no. 2, p. 442, 1975.

[79] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[80] N. Cliff, *Ordinal methods for behavioral data analysis*. Lawrence Erlbaum Associates, 1996.

[81] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, Jan 2017.

[82] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *Proceedings of Annual Meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[83] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, Aug 2015, pp. 17–26.

[84] G. Macbeth, E. Razumiejczyk, and R. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[85] P. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton University, 1963.

[86] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

[87] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, vol. 30, no. 3, pp. 507–512, 1974.

[88] K. Muller, "Statistical power analysis for the behavioral sciences," *Technometrics*, vol. 31, no. 4, pp. 499–500, 1989.

[89] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Oct 2016, pp. 309–320.

[90] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, July 2019.

[91] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction, Springer Series in Statistics*.

[92] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 6–.

[93] M. Gonen and E. Alpaydin, "Multiple Kernel Learning Algorithms," *JOURNAL OF MACHINE LEARNING RESEARCH*, vol. 12, pp. 2211–2268, JUL 2011.

[94] D. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society. Series B*, vol. 20, 07 1958.

[95] Y. Zhao, Y. Yang, H. Lu, Y. Zhou, Q. Song, and B. Xu, "An empirical analysis of package-modularization metrics: Implications for software fault-proneness," *Information and Software Technology*, vol. 57, no. Supplement C, pp. 186 – 203, 2015.

[96] W. Ma, L. Chen, Y. Yang, Y. Zhou, and B. Xu, "Empirical analysis of network measures for effort-aware fault-proneness prediction," *Information and Software Technology*, vol. 69, no. Supplement C, pp. 50–70, 2016.

[97] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, Oct 1988.

[98] Z. Mahmood, D. Bowes, P. C. R. Lane, and T. Hall, "What is the impact of imbalance on software defect prediction performance?" in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:4.

[99] H. Aman, S. Amasaki, T. Sasaki, and M. Kawahara, "Lines of comments as a noteworthy metric for analyzing fault-proneness in methods," *IEICE Transactions on Information and Systems*, vol. E98D, no. 12, pp. 2218 – 2228, 2015.

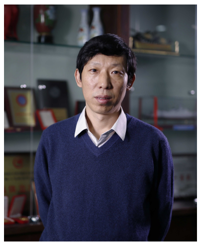
[100] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, no. Supplement C, pp. 388 – 402, 2015.

[101] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, June 2007.

[102] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *Empirical Software Engineering*, vol. 22, no. 6, pp. 2806–2851, Dec 2017.

[103] D. J. Hand, "Measuring classifier performance: a coherent alternative to the area under the roc curve," *Machine Learning*, vol. 77, no. 1, pp. 103–123, 2009.

Haonan Tong is a Ph.D candidate at the School of Reliability and Systems Engineering, Beihang University, China. He received his M.S. degree from Beihang University. His research interests are machine learning, software reliability prediction, and mining software repositories. E-mail: tonghaonan@buaa.edu.cn.



Bin Liu is a Professor at the School of Reliability and Systems Engineering, Beihang University, China. He received the Ph.D degree and M.S. degree in Software Reliability and Software Testing from Beihang University, the B.S. degree in Aviation Science and Engineering from Beihang University. His research interests are software testing, software architecture, software fault diagnosis and location, and software reliability. E-mail: liubin@buaa.edu.cn.

Shihai Wang is a Lecturer at the School of Reliability and Systems Engineering, Beihang University, China. He received the Ph.D degree in Computer Science from University of Manchester in 2010, M.S. degree in Compute Science from University of Sheffield, and B.S. degree in Compute Science from Harbin Institute of Technology. His research interests include machine learning, software testing, pattern recognition and application in software engineering. E-mail: wangshihai@buaa.edu.cn.