Cross-Version Defect Prediction via Hybrid Active Learning with Kernel Principal Component Analysis

Zhou Xu[†‡], Jin Liu[†*], Xiapu Luo[‡] and Tao Zhang[§]
[†]State Key Laboratory of Software Engineering, School of Computer, Wuhan University, China
[‡]Department of Computing, The Hong Kong Polytechnic University, China
[§]College of Computer Science and Technology, Harbin Engineering University, China
[*]Corresponding author email: jinliu@whu.edu.cn

***Abstract*—As defects in software modules may cause product failure and financial loss, it is critical to utilize defect prediction methods to effectively identify the potentially defective modules for a thorough inspection, especially in the early stage of software development lifecycle. For an upcoming version of a software project, it is practical to employ the historical labeled defect data of the prior versions within the same project to conduct defect prediction on the current version, i.e., *C*ross-*V*ersion *D*efect *P*rediction (*CVDP*). However, software development is a dynamic evolution process that may cause the data distribution (such as defect characteristics) to vary across versions. Furthermore, the raw features usually may not well reveal the intrinsic structure information behind the data. Therefore, it is challenging to perform effective CVDP. In this paper, we propose a two-phase CVDP framework that combines *H*ybrid *A*ctive *L*earning and *K*ernel *P*CA (*HALKP*) to address these two issues. In the first stage, HALKP uses a hybrid active learning method to select some informative and representative unlabeled modules from the current version for querying their labels, then merges them into the labeled modules of the prior version to form an enhanced training set. In the second stage, HALKP employs a non-linear mapping method, kernel PCA, to extract representative features by embedding the original data of two versions into a high-dimension space. We evaluate the HALKP framework on 31 versions of 10 projects with three prevalent performance indicators. The experimental results indicate that HALKP achieves encouraging results with average F-measure, g-mean and Balance of 0.480, 0.592 and 0.580, respectively and significantly outperforms nearly all baseline methods.**

I. Introduction

As software projects have increased in both size and complexity, the defects in the software modules (i.e., the methods, files or packages) are inevitable [1], [2]. Timely detecting and repairing the defects before releasing the products are critical for software quality assurance. Since defect prediction aims to effectively identify the defect-prone modules by mining software repositories, it empowers the software practitioner or testers to give priority to the suspicious modules.

Traditional defect prediction builds supervised classification model on the historical labeled modules, then predicts the labels of the new unlabeled modules within the same software project, i.e., **W**ithin-**P**roject **D**efect **P**rediction (**WPDP**). WPDP suffers from scarce labeled data for building an effective and robust defect classifier. To relieve this issue, researchers propose to employ transfer learning [3] by utilizing the labeled data from other software projects to conduct defect prediction on the current project, i.e., **C**ross-**P**roject **D**efect **P**rediction (**CPDP**). But the performance of CPDP is threaten by the risk of data distribution difference across projects.

Previous studies proposed to train the classification model on the labeled modules of the previous version of a project, and then conduct defect prediction on the unlabeled ones of its subsequent version [4], [5], i.e., **C**ross-**V**ersion **D**efect **P**rediction (**CVDP**). The intuition behind CVDP is that, for a software project with multiple versions, the distribution difference between different versions is lower than that between different projects because new version usually carries a large amount of information from the previous versions [6]. It is worth noting that as software is evolving, developers would add new software modules, delete old modules, modify or refactor existing modules during the process of version upgrade. These operations may eliminate existing defects and introduce new defects in the modules, leading to the changes of defect distribution between successive versions [7]. In this case, the classification model built on the labeled modules of the prior version may not be very effective for the unlabeled modules of the current version, because the model does not take the distribution changes into consideration.

To address this issue, a recent work [7] utilized an active learning method that iteratively selects the most valuable samples for querying their labels. In particular, active learning is used to select a small number of unlabeled modules from the current version for labeling. These selected modules are merged into the prior version, expecting that the mixed labeled modules form an enhanced training set by incorporating some additional structure information of the current version. Unfortunately, this approach has several limitations (to be described in the next subsection), which motivate our new framework.

A. Motivation

The key component of active learning is to design appropriate measurement criteria for selecting the candidate samples that are informative and representative [8]. The informativeness of a sample is measured by its prediction uncertainty based on a classification model. The representativeness of a sample is measured by whether it can well represent the space distribution of the data. The uncertainty measure based active learning method used in [7] has the limitation that the selected candidate modules only rely on the classification model built on the labeled modules from the prior version. This process ignores the distribution information contained in the unlabeled modules of the current version. Thus, the candidate modules are merely informative but not representative [8], [9]. As a result, these candidate modules may not be very helpful to form a powerful training set after being merged into the prior version. In this paper, we employ a novel hybrid active learning (HAL) framework to select the most valuable modules for labeling. This framework combines two measurement criteria, namely uncertainty measure and information density measure. The first measure guarantees that the candidate modules are informative, while the second measure insures that the candidate modules are representative by considering the distribution information of the current version.

Although the mixed labeled modules of the prior version are more adaptive to the remaining unlabeled modules of the current version, the classifier performance can also be significantly affected by the feature representation of the data of the two versions. Wang et al. [10] pointed out that learning appropriate feature representation for the data is necessary for defect prediction since the raw features may not properly unfold the essential property of the original data. Therefore, some previous studies employed **P**rincipal **C**omponent **A**nalysis (**PCA**) to convert the raw module features to a low-dimension space [11]–[13]. PCA performs well when the data are linearly separable and follow a Gaussian distribution [14]. Unfortunately, the features extracted by PCA may not be beneficial to achieve satisfactory performance for defect prediction [15] since the real data may not meet the above requirements [10], [16]. Lu et al. [7] used a feature extraction method, named **M**ulti-**D**imensionality **S**caling (**MDS**), to reduce the feature dimensionality before applying the active learning method. The main limitation of MDS is that it requires all modules of two versions as a whole input for the method, which limits its scalability to new modules outside of both versions [17]. To tackle this issue, in this paper, we exploit a non-linear feature extraction method, named **K**ernel **PCA** (**KPCA**) [18], to map the data of two versions into a high-dimension feature space, in which the converted data can increase the probability of linear separability and follow an approximate Gaussian distribution [19], [20]. Different from MDS, the data of two versions are converted by KPCA separately using the same mapping rules without merging them together in advance.

To sum up, in this paper, we propose **HALKP**, a novel CVDP framework that leverages the two mentioned-above methods: **H**ybrid **A**ctive **L**earning and **KP**CA. Our framework consists of two major stages: in the first stage, HALKP exploits HAL to select some informative and representative unlabeled modules from the current version of a given project for querying their labels, and then incorporates them with the labeled modules of its prior version to construct a mixed training set; in the second stage, HALKP utilizes KPCA to map the mixed training modules and the remaining unlabeled modules into a high-dimension feature space. Finally, a logistic regression model is built on the mapped training modules to perform CVDP for the mapped unlabeled modules.

We conduct experiments on 31 versions of 10 software projects from the MORPH dataset to evaluate the effectiveness of HALKP for CVDP with three widely-used performance indicators, including F-measure, g-mean, and Balance. Across 31 cross-version pairs, HALKP achieves average F-measure, g-mean and Balance of 0.480, 0.592 and 0.580, respectively. The experimental results demonstrate the superiority of HALKP compared with 11 baseline methods in most cases.

B. Contribution

In summary, we make the following contributions:

(1) We propose a novel framework HALKP to address two important issues in CVDP, including lessening the distribution difference and extracting favorable feature presentation. The former is caused by the evolution of software development whereas the latter is due to the complex structure hidden behind the data.

(2) The proposed framework first uses HAL to carefully choose some informative and representative unlabeled modules from the current version for labeling. These selected modules can supplement part of structure information of current version into the prior version. The framework further leverages KPCA to extract the essential structure of the cross-version data.

(3) We evaluate HALKP on 31 versions of 10 projects with three performance indicators. The extensive experiments manifest that HALKP achieves encouraging results compared with 11 baseline methods.

II. Related Work

A. Software Defect Prediction

Within-Project Defect Prediction (WPDP): For a software project with sufficient historical labeled data, supervised classifiers are built to conduct WPDP for the upcoming modules within the same project. Recent studies have examined the impact of different classification models [21], [22], feature selection methods [15], sampling approaches [23], and model validation techniques [24] on the performance of WPDP.

Cross-Project Defect Prediction (CPDP): For a project that does not have sufficient labeled data, researchers proposed transfer learning methods to conduct CPDP by transferring the knowledge from other projects (a.k.a. target projects) to facilitate the prediction of the defective modules in the current project (a.k.a. source project) [25]. For example, Nam et al. [26] applied an improved transfer component analysis method to make the distributions between source and target project data similar. Xia et al. [27] proposed a two-phase hybrid model reconstruction method by combing genetic algorithm and ensemble learning to create a massive compositional model for CPDP. However, these traditional CPDP methods hold an assumption that the source and target project data share the same feature set.

Heterogeneous CPDP (HCPDP): For cross-project data that have heterogeneous feature sets, researchers proposed heterogeneous learning methods for the HCPDP issue. For example, Jing et al. [28] employed the canonical correlation analysis method to map the source and target project data into a common space by maximizing their similarity. Nam et al. [29] employed similarity calculation methods to select feature pairs that have the most similar distributions from the cross-project data, and then used the matched features to conduct HCPDP.

Usually, CPDP and HCPDP do not perform better than WPDP in most cases [30]–[33], because the performance of traditional CPDP and HCPDP methods is affected by the data distribution difference between the source and target data, especially for HCPDP. In [28], the paper only reported the results of 22 heterogeneous cross-project pairs among total 90 combinations, and thus its universality needs to be further verified. In [29], the proposed method sometimes fails due to no matched features being selected when all the matching scores are lower than a given threshold.

Cross-Version Defect Prediction (CVDP): For a project with multiple versions, researchers proposed to conduct CVDP on the current version by utilizing the labeled modules of its prior versions [7]. CVDP can be treated as a special scenario between WPDP and CPDP. Although the software development is a dynamical process that can lead to the changes of the data structure characteristics during version upgrade, new version usually conserves much information from the old versions. Thus, the distribution difference across versions will be smaller than that of across projects. From this view, CVDP will provide more practical benefits for defect prediction on the current version. Different from some prior work [5], [34]–[36] that directly exploited all the labeled data of the prior version to build prediction model for CVDP, in this paper, we focus on leveraging active learning for CVDP.

B. Active Learning

The core component of active learning is the selection strategies for the candidate samples. Two main strategies aim at selecting the informative and representative samples [8]. To select the informative samples, uncertainty based active learning [37], query-by-committee based active learning [38], expected error reduction based active learning [39] are proposed. To select the representative samples, clustering-based active learning exploits the cluster structure of the unlabeled samples to select such samples [40], [41].

Recently, active learning has been introduced into defect prediction. Luo et al. [42] proposed a two-stage active learning framework combining a clustering technique and support vector machine. Li et al. [43] proposed an active semi-supervised learning method to select the most helpful modules. They first randomly select 10% modules for labeling and then used them to build an ensemble learning model for the rest unlabeled modules. Lu et al. [44] proposed an adaptively defect prediction framework combining supervised learning and active learning. Different from conventional uncertainty measure based active learning that selects the most uncertainty modules, they selected the most certain ones. However, all these methods are applied to WPDP and need a certain amount of labeled modules to initiate the methods.

Lu et al. [7] were the first to introduce active learning into CVDP to identify the most valuable modules from the current version for labeling by querying the domain experts, and then merged them into prior version to construct a hybrid training set. Due to the limitation of their method as mentioned in Section I-A, in this paper, we employ a hybrid active learning framework to select the candidate modules that are both informative and representative in the current version.

III. Our Method

A. Overview

Figure 1 depicts the overview of our HALKP framework that consists of two stages with multiple steps. In the first stage, for each cycle of active learning, in step ①, a hybrid active learning method is used to select one unlabeled module that is both informative and representative based on labeled modules of the prior version; in step ②, a domain expert (such as software tester and verification engineer) is consulted to thorough inspect the module and assigns its label in step ③; in step ④, the selected module is merged into the prior version to form a mixed labeled module pool. One cycle is completed after the four steps. In the next cycle, the four steps are conducted with the remaining unlabeled modules and the mixed labeled modules to select the next candidate module for labeling. The cycles stop until reaching a predefined threshold, which denotes the number of selected candidate modules. After this stage, we obtain two new module sets, i.e., the remaining unlabeled modules of current version and the mixed labeled modules of prior version. In the second stage, a non-linear feature extraction method KPCA is used to convert the data of two versions into a mapped space in step ⑤. Finally, a standard defect prediction process is performed by training a classifier using the mapped mixed labeled modules of the prior version in step ⑥ and then predicting the labels of the remaining unlabeled modules of the current version in step ⑦.

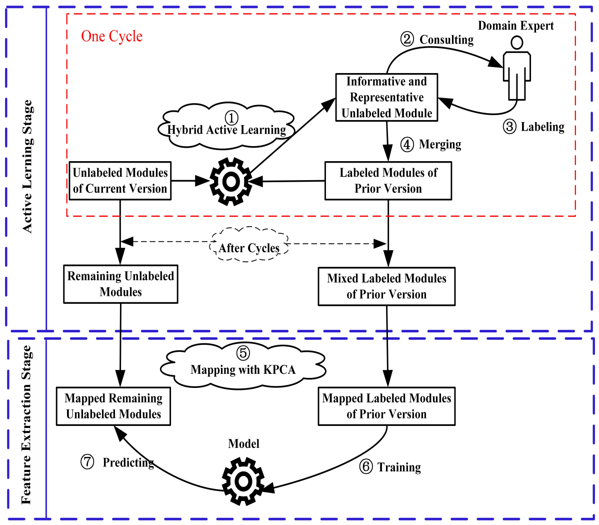
Active Lerning Stage
One Cycle
② Consulting
Domain Expert
Informative and Representative Unlabeled Module
① Hybrid Active Learning
③ Labeling
④ Merging
Unlabeled Modules of Current Version
Labeled Modules of Prior Version
After Cycles
Remaining Unlabeled Modules
Mixed Labeled Modules of Prior Version
Feature Extraction Stage
⑤ Mapping with KPCA
Mapped Remaining Unlabeled Modules
Mapped Labeled Modules of Prior Version
Model
⑦ Predicting
⑥ Training

Fig. 1: Overview of our CVDP framework.

B. Active Learning with Uncertainty Measure

Uncertainty measure based active learning iteratively selects the modules with most uncertainty. It first trains a probabilistic classifier with the pool of the labeled modules of the prior version, and then uses the resulting model to predict the labels of the unlabeled modules of the current version. The uncertainty measure $f(x_i)$ of module x_i is defined as its conditional entropy given the label variable y:

$$f(x_i) = -\sum_{y \in Y} P(y|x_i, \theta_L) \log P(y|x_i, \theta_L), \quad (1)$$

where $Y \in \{0, 1\}$ denotes the label set (i.e., 1 denotes defective module and 0 denotes non-defective module), θ_L denotes the classification model θ trained on the labeled module set L of the prior version, and the conditional probability distribution $P(y|x_i, \theta_L)$ is calculated by this model. Uncertainty measure based active learning selects the unlabeled module that has the largest conditional entropy.

C. Active Learning with Information Density Measure

The shortcoming of uncertainty measure based active learning is that the selected module merely relates to the classifier that built on the labeled modules of the prior version, which may lead to a bias towards the candidate module [9]. Previous work has shown that it is important to select representative samples by considering the distribution information of unlabeled samples [8]. The intuition here is that the module located in the center of data cluster is representative. In this paper, we use an informative density measure to select the representative module. This measure is based on the mutual information between a module and the remaining unlabeled modules since the mutual information is a measurement of interdependence among two variable sets. The information density measure $d(x_i)$ of module x_i is defined as

$$d(x_i) = H(x_i) - H(x_i|U'), \quad (2)$$

where U' denotes the remaining unlabeled module set after removing module x_i from the unlabeled module set U, and $H()$ denotes the entropy function. We use the Gaussian Process framework in [9] to compute this measure.

D. A Hybrid Active Learning Framework

As mentioned above, the uncertainty measure selects the informative modules by relying on the labeled modules of the prior version, whereas information density measure selects the representative modules by considering the distribution information of the unlabeled modules of the current version. Since the two measures pick up candidate modules from different views, in this work, we introduce a hybrid measure framework that combines the advantages of the above two measures following the work [9]. The hybrid measure $h(x_i)$ of module x_i is defined as

$$h(x_i) = f(x_i)^{\beta} d(x_i)^{1-\beta}, \quad (3)$$

where β ($0 \le \beta \le 1$) is a controlling parameter towards the two measures. Higher β indicates that the hybrid measure is more bias to the uncertainty measure, otherwise to the information density measure.

The module $\widehat{x_i}$ with the maximum hybrid measure value is selected as the candidate module, that is

$$\widehat{x_i} = \arg\max_{x_i \in U} h(x_i) \quad (4)$$

The label of module $\widehat{x_i}$ is determined by the domain experts by thoroughly inspecting the module and judge whether it contains defects with their rich experience. After the active learning, we obtain two new datasets, i.e., the mixture labeled module set L' and the remaining unlabeled module set U'. Algorithm 1 presents the process of the hybrid framework.

Algorithm 1 Framework of Hybrid Active Learning.

```
Input: The labeled module set of prior version, $L$; The unlabeled module set of current version, $U$; The probabilistic classifier, $\theta$.
Output: The mixed labeled module set of prior version, $L'$; The rest of unlabeled module set of current version, $U'$.
1:  repeat
2:      for $x_i \in U$ do
3:          Calculate $f(x_i)$ according to Eq (1).
4:          Calculate $d(x_i)$ according to Eq (2).
5:          Calculate $h(x_i)$ according to Eq (3).
6:      end for
7:      Select $\widehat{x_i}$ that has the largest hybrid measure value according to Eq (4).
8:      Remove $\widehat{x_i}$ from $U$ ($U' = U - \widehat{x_i}$), $U = U'$.
9:      Query the label $y_{\widehat{x_i}}$ of $\widehat{x_i}$.
10:     Merge $(\widehat{x_i}, y_{\widehat{x_i}})$ into $L$ ($L' = L + (\widehat{x_i}, y_{\widehat{x_i}})$), $L = L'$.
11: until Meeting the stop criterion.
12: return $L'$ and $U'$.
```

E. Feature Extraction Based on KPCA

Following the active learning, we further employ the non-linear feature extraction method KPCA to map set L' and U' into a high-dimension feature space F in which the standard PCA is performed. Assume one of the module set (such as the set U') as $x'_i \in U', i = 1, 2, \ldots, n$, where $x_i = [x'_{i1}, x'_{i2}, \ldots, x'_{im}]^{\mathrm{T}} \in \Re^m$ denotes the feature set and n denotes the number of modules in U'.

First, we map each data point (i.e., a module) x'_i into a new point $\varphi(x_i')$ with a non-linear function φ and centralize the mapped data points. The covariance matrix $\mathbf{C}$ of the mapped data is defined as

$$\mathbf{C} = \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \varphi(x'_i)\varphi(x'_i)^{\mathrm{T}} \quad (5)$$

To perform the linear PCA in F, we diagonalize the above covariance matrix $\mathbf{C}$ by solving the eigenvalue problem

$$\mathbf{CV} = \lambda \mathbf{V}, \quad (6)$$

where λ and $\mathbf{V}$ denote the eigenvalues and eigenvectors of $\mathbf{C}$, respectively.

Since all solutions $\mathbf{V}$ lie in the span of the mapped data, we multiply both sides of Eq (6) by $\varphi(x'_l)^{\mathrm{T}}$ as

$$\varphi(x'_l)^{\mathrm{T}}\mathbf{CV} = \lambda\varphi(x'_l)^{\mathrm{T}}\mathbf{V}, \forall l = 1, 2, \ldots, n \quad (7)$$

Meanwhile, the eigenvectors $\mathbf{V}$ can be linearly expressed via $\varphi(x'_1), \varphi(x'_2), \ldots, \varphi(x'_n)$ with coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$:

$$\mathbf{V} = \textstyle\sum_{j=1}^{n} \alpha_j \varphi(x'_j) \quad (8)$$

By substituting Eq (5) and Eq (8) into Eq (7), the equation is rewritten as following formula:

$$\begin{aligned} \tfrac{1}{n}\varphi(x'_l)^{\mathrm{T}} \textstyle\sum_{i=1}^{n} \varphi(x'_i)\varphi(x'_i)^{\mathrm{T}} \sum_{j=1}^{n} \alpha_j \varphi(x'_j) \\ = \lambda\varphi(x'_l)^{\mathrm{T}} \textstyle\sum_{j=1}^{n} \alpha_j \varphi(x'_j) \end{aligned} \quad (9)$$

Define the kernel function $\kappa(x'_i, x'_j)$ as $\kappa(x'_i, x'_j) = \varphi(x'_i)^{\mathrm{T}}\varphi(x'_j)$. Then Eq (9) is rewritten as

$$\begin{aligned}\frac{1}{n}\textstyle\sum_{l=1,i=1}^{n}\kappa(x'_l, x'_i)\sum_{i=1,j=1}^{n}\alpha_j\kappa(x'_i, x'_j) \\ = \lambda\textstyle\sum_{l=1,j=1}^{n}\alpha_j\kappa(x'_l, x'_j)\end{aligned} \tag{10}$$

Define the kernel matrix $\mathbf{K}$ with size $n \times n$ as $\mathbf{K}_{i,j} = \kappa(x'_i, x'_j)$. Then Eq (10) is rewritten as

$$\mathbf{K}^2\alpha = n\lambda\mathbf{K}\alpha, \tag{11}$$

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

The α in Eq (11) can be obtained by solving the following eigenvalue problem

$$\mathbf{K}\alpha = n\lambda\alpha \tag{12}$$

for nonzero eigenvalues λ and corresponding eigenvectors α. After obtaining α, the mapped data is calculated as $\mathbf{K}\alpha$.

The module $\varphi(x'_L)$ in another module set (such as set L') is mapped as

$$\begin{aligned}\mathbf{V}\cdot\varphi(x'_L) &= \textstyle\sum_{i=1}^{n}\alpha\varphi(x'_i)^{\mathrm{T}}\varphi(x'_L) \\ &= \textstyle\sum_{i=1}^{n}\alpha\kappa(x'_i, x'_L)\end{aligned} \tag{13}$$

To eliminate the noise in the data, when performing PCA in the space F, we maintain the most important principal components that capture at least 95% of total variances of the data according to their cumulative contribution rates [45].

After feature extraction, the data of two versions are transformed into set L_{map} and U_{map}, respectively.

IV. Implementation

In CVDP framework, there are some parameters needed to be tuned. In the active learning phase, although querying the labels comes at a cost, if the process can make the defect prediction more effective and improve the software quality, the cost is acceptable as long as we control the querying number (i.e., the threshold) at a small amount, usually less than 20% of the total number of unlabeled modules [7]. In this work, we select four thresholds, i.e., 5%, 10%, 15%, 20%. In addition, since β is a version-specific control parameter that varies between different cross-version pairs, we could not assign a constant value as the optimal parameter for all cases. In this work, we set a relatively wide range as β= 0, 0.1, ..., 0.9, 1, and use the F-measure as the primary indicator to determine the optimal β value for different cross-version pairs. $\beta = 1$ denotes the basic uncertainty measure based active learning while $\beta = 0$ stands for the information density measure based active learning. How to adaptively set the parameter values remains an open issue in our future work. In practice, the labels of the candidate unlabeled modules are assigned by the domain experts. In this work, we simulate the process by endowing the ground-truth labels to them following prior work [7], [43].

In the feature extraction phase, we choose the Gaussian RBF kernel as the non-linear function for KPCA since it usually exhibits promising performance in many applications [46]–[48]. The RBF kernel is defined as

$$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{14}$$

where $\|\cdot\|$ denotes the l_2 norm and $2\sigma^2 = \omega$ denotes the width of the RBF function. In terms of the parameter σ, we set $\sigma = 10$ through extensive experiences since it can achieve satisfactory results on most cross-version pairs.

To imitate CVDP, we design two different cross-version schemes. The first scheme is to conduct CVDP between two successive versions. More specifically, for the ant project in benchmark dataset (as mentioned in Section V-A), the modules in version 1.3 are used to train a prediction model with some selected modules from version 1.4. This model is then tested on the remaining modules in version 1.4. The second scheme is to conduct CVDP between the last version and all its prior versions. More specifically, the modules in version 1.3, 1.4 and 1.5 are used to train a prediction model with some selected modules from version 1.6. This model is then used to predict the labels of the remaining unlabeled modules in version 1.6.

In terms of the classifier for uncertainty measure based active learning and the cross-version prediction process, we select the logistic regression classifier as the base learner. This classifier is extensively used in defect prediction studies [21], [49]–[54] and active learning applications [40], [55], [56].

V. Evaluation

A. Benchmark Dataset

We conduct extensive experiments on 31 versions of 10 software projects taken from MORPH dataset [57], which has been widely used in many previous defect prediction studies [22], [58]–[60]. Each module in the project consists of 20 object-oriented features and a dependent variable, i.e., defect number in each module. These features are collected at method level by Jureczko, Madeyski and Spinellis with ckjm tool [61], [62]. More detailed descriptions about these features can be found in [61]. In this work, we label the module as 1 if it contains one or more defects. Otherwise, we label it as 0.

The statistic of the projects is shown in Table I, including the number of the modules (# M), the newly added modules (+ M) and the removed modules compared with the prior version (- M), the defective modules (# D), the new defective modules (+ D) and the eliminated defective modules (- D) compared with the prior version, the percentage of defective modules (% D). Note that the new defective modules include the newly introduced defective modules and the modules without defect in prior version but with defect in current version. The eliminated defective modules include the removed defective modules from prior version and the modules with defect in prior version but without defect in current version. From the table, we observe that, for most projects, the current versions inherit many modules from the prior version, however, the added (removed) modules and defective modules will lead to the distribution difference across the versions.

B. Performance Indicators

We measure the performance of HALKP with three indicators, namely F-measure, g-mean and Balance, which are widely used in defect prediction [10], [26], [28], [33], [43], [63]–[68]. The three indicators can be derived from some basic indicators listed in Table II and defined as

$$\text{F-measure} = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}} \tag{15}$$

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

$$\text{Balance} = 1 - \sqrt{\frac{(0 - \text{pf})^2 + (1 - \text{pd})^2}{2}} \tag{17}$$

For all research questions, we use a non-parametric test, the Wilcoxon signed-rank test, to check whether the differences between HALKP and the baseline methods are statistically significant at a confidence level of 95%. This test is suitable for tested objects that have unknown distribution [69], [70]. The difference is significant if the p value of the test is lower than 0.05. Further, we use the effect size, Cliff's Delta [71], to qualify the amount of the difference. The difference is substantial if the d value of the Cliff's Delta is not lower than 0.146 [27], [49], [72].

TABLE I: Statistic of the Benchmark Dataset

Project	Version	# M	+ M	- M	# D	+ D	- D	% D
ant	1.3	125			20			16.00%
	1.4	178	53	0	40	34	14	22.47%
	1.5	293	125	10	32	25	33	10.92%
	1.6	351	59	1	92	87	15	26.21%
camel	1.2	608			216			35.53%
	1.4	872	303	39	145	97	119	16.63%
	1.6	965	108	15	188	154	63	19.48%
ivy	1.1	111			63			56.76%
	1.4	241	132	2	16	9	55	6.64%
	2.0	352	352	241	40	40	16	11.36%
jedit	3.2	272			90			33.09%
	4.0	306	41	7	75	54	43	24.51%
	4.1	312	21	15	79	61	26	25.32%
log4j	1.0	135			34			25.19%
	1.1	109	11	37	37	27	12	33.94%
	1.2	205	102	6	189	187	4	92.20%
poi	1.5	237			141			59.49%
	2.0	314	90	13	37	19	122	11.78%
	2.5	385	71	0	248	240	10	64.42%
synapse	1.0	157			16			10.19%
	1.1	222	70	5	60	58	7	27.03%
	1.2	256	37	3	86	71	30	33.59%
velocity	1.4	196			147			75.00%
	1.5	214	59	41	142	115	68	66.36%
	1.6	229	20	5	78	85	52	34.06%
xalan	2.4	723			110			15.21%
	2.5	803	114	34	387	351	39	48.19%
	2.6	885	119	37	411	287	116	46.44%
xerces	1.2	440			71			16.14%
	1.3	453	20	7	69	65	54	15.23%
	1.4	588	260	125	437	428	34	74.32%

C. Experimental Results

RQ1: How effective is HALKP compared with some variations of its downgraded versions?

Method: As mentioned above, our CVDP framework HALKP consists of two stages: active learning stage for picking up informative and representative unlabeled modules from the current version to form an enhanced training data and feature

TABLE II: Indicators for Defect Prediction

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

extraction stage for better presenting the data of two versions. This question investigates whether the two-phase framework is better than five baseline methods of its downgraded variations, including the method that only uses the **Ori**ginal data of two versions for CVDP (**Ori**), the method that **O**nly applies **KPCA** to the original data of two versions without active learning stage (**O-KPCA**), the method that **O**nly applies **HAL** to the original data of two versions without feature extraction stage (**O-HAL**). In addition, considering that KPCA is a non-linear variation of PCA, we also explore whether the features extracted by non-linear method KPCA are more effective than that by linear method PCA for CVDP. We compare HALKP with two extra variations: one method that **O**nly performs **PCA** on the original data of two versions (**O-PCA**) and one method that combines **HAL** with **PCA** (**HALPCA**).

Results: Due to the space limit, we only report the detailed results of the three indicators for each cross-version pair under threshold 5% as shown in Table III. The best indicator values are in bold. For the results of the three indicators under other three thresholds, we report the corresponding box-plots as depicted in Figures 2 to 4. All detailed results under each thresholds can be found in our additional materials [73].

Table III shows that HALKP achieves the best average performance across 31 cross-version pairs in terms of all three indicators. More concretely, in terms of F-measure, HALKP is better than the baseline methods on 23 out of 31 cross-version pairs and the average F-measure by HALKP (0.480) gains improvements between 15.8% and 53.9% over the baseline methods; in terms of g-mean, HALKP is superior to the baseline methods on 26 out of 31 cross-version pairs and the average g-mean by HALKP (0.592) gains improvements between 9.5% and 40.2% over the baseline methods; in terms of Balance, HALKP outperforms the baseline methods on 25 out of 31 cross-version pairs and the average Balance by HALKP (0.580) gains improvements between 8.6% and 30.1% over the baseline methods. The p and d values indicate that the performance differences between HALKP and the variations are statistically significant and substantial.

Figures 2 to 4 depict the box-plots of the six methods on three indicators across all cross-version pairs under threshold 10%, 15%, 20%, respectively. All figures show that the median and minimum values of all three indicators by HALKP are superior to that by the baseline methods. In addition, the maximum values of g-mean and Balance by HALKP are higher than that by all baseline methods.

In terms of the first three methods, they are the conventional CVDP methods due to building classification model with all modules of prior version and predict all modules of current

TABLE III: The Detailed Results for HALKP and Its Five Variations on Each Cross-Version Pair under Threshold 5%

Cross-Version Pair			F-measure						g-mean						Balance					
Project	Prior	Current	Ori	O-PCA	O-KPCA	O-HAL	HALPCA	HALKP	Ori	O-PCA	O-KPCA	O-HAL	HALPCA	HALKP	Ori	O-PCA	O-KPCA	O-HAL	HALPCA	HALKP
ant	1.3	1.4	0.215	0.151	0.354	0.237	0.240	**0.359**	0.390	0.306	0.535	0.410	0.392	**0.538**	0.409	0.362	0.523	0.421	0.406	**0.527**
	1.4	1.5	0.273	0.338	0.252	0.414	**0.364**	0.316	0.504	0.561	0.590	0.621	0.568	**0.674**	0.487	0.532	0.587	0.583	0.535	**0.671**
	1.5	1.6	0.286	0.293	0.336	0.336	0.333	**0.383**	0.414	0.424	0.478	0.456	0.456	**0.530**	0.416	0.423	0.466	0.443	0.442	**0.510**
	1.3+1.4+1.5	1.6	0.298	0.234	0.347	0.330	0.263	**0.422**	0.426	0.372	0.493	0.453	0.397	**0.565**	0.423	0.393	0.480	0.441	0.406	**0.542**
camel	1.2	1.4	0.322	0.332	0.388	0.353	0.333	**0.401**	0.518	0.500	**0.647**	0.544	0.507	0.645	0.501	0.481	**0.644**	0.521	0.485	0.638
	1.4	1.6	0.169	0.158	0.346	0.197	0.194	**0.406**	0.314	0.299	0.548	0.343	0.335	**0.581**	0.364	0.357	0.534	0.378	0.373	**0.557**
	1.2+1.4	1.6	0.233	0.220	0.373	0.256	0.250	**0.429**	0.394	0.372	0.546	0.405	0.398	**0.583**	0.408	0.394	0.523	0.413	0.408	**0.552**
ivy	1.1	1.4	0.122	0.170	0.163	0.173	**0.220**	0.165	0.492	0.604	0.596	0.603	**0.687**	0.583	0.492	0.588	0.596	0.600	**0.675**	0.583
	1.4	2.0	0.145	0.042	0.299	0.095	0.182	**0.349**	0.311	0.156	0.578	0.229	0.327	**0.586**	0.363	0.310	**0.560**	0.330	0.369	0.558
	1.1+1.4	2.0	0.338	0.314	0.260	**0.343**	0.319	0.300	0.531	0.508	0.525	0.537	0.514	**0.583**	0.503	0.486	0.511	0.508	0.490	**0.564**
jedit	3.2	4.0	0.541	0.540	0.527	0.567	0.577	**0.634**	0.687	0.689	0.691	0.697	0.706	**0.772**	0.673	0.678	0.690	0.676	0.686	**0.769**
	4.0	4.1	0.548	0.528	0.533	**0.600**	0.574	0.591	0.640	0.628	0.676	0.682	0.671	**0.732**	0.596	0.586	0.662	0.635	0.629	**0.724**
	3.2+4.0	4.1	0.566	0.531	0.546	**0.614**	0.571	0.564	0.681	0.657	0.697	0.703	0.687	**0.708**	0.652	0.631	0.693	0.663	0.662	**0.699**
log4j	1.0	1.1	0.567	0.623	0.590	0.655	**0.667**	0.632	0.649	0.691	0.668	0.720	**0.728**	0.712	0.613	0.653	0.632	0.684	**0.693**	0.683
	1.1	1.2	0.452	0.381	0.587	0.605	0.524	**0.725**	0.491	0.456	0.518	**0.577**	0.516	0.401	0.485	0.454	0.517	**0.569**	0.512	0.410
	1.0+1.1	1.2	0.397	0.376	0.486	0.448	0.409	**0.613**	0.483	0.467	0.550	0.506	0.492	**0.622**	0.467	0.456	0.519	0.492	0.474	**0.598**
poi	1.5	2.0	0.183	0.173	0.170	0.207	0.187	**0.259**	0.452	0.417	0.452	0.494	0.445	**0.585**	0.454	0.423	0.452	0.494	0.449	**0.584**
	2.0	2.5	0.077	0.055	0.297	0.103	0.073	**0.604**	0.200	0.167	0.401	0.232	0.194	**0.616**	0.321	0.313	0.417	0.332	0.320	**0.606**
	1.5+2.0	2.5	0.132	0.112	0.594	0.196	0.171	**0.675**	0.262	0.242	0.618	0.325	0.303	**0.683**	0.343	0.335	0.601	0.371	0.360	**0.667**
synapse	1.0	1.1	0.062	0.219	0.384	0.286	0.254	**0.426**	0.181	0.359	0.547	0.426	0.391	**0.574**	0.316	0.387	0.540	0.428	0.404	**0.559**
	1.1	1.2	0.406	0.413	0.444	0.417	0.435	**0.479**	0.523	0.523	0.564	0.533	0.542	**0.591**	0.502	0.497	0.551	0.510	0.512	**0.584**
	1.0+1.1	1.2	0.389	0.372	0.397	0.396	0.404	**0.458**	0.498	0.485	0.523	0.504	0.510	**0.568**	0.473	0.465	0.510	0.478	0.482	**0.544**
velocity	1.4	1.5	0.718	0.731	0.698	0.739	0.743	**0.751**	0.239	0.322	0.311	0.252	0.351	**0.493**	0.330	0.370	0.361	0.337	0.386	**0.488**
	1.5	1.6	0.552	0.541	0.541	**0.602**	0.577	0.596	0.560	0.535	0.562	0.631	0.592	**0.657**	0.542	0.519	0.548	0.610	0.574	**0.647**
	1.4+1.5	1.6	0.534	0.535	0.524	**0.563**	0.561	0.527	0.383	0.368	**0.504**	0.402	0.393	0.494	0.400	0.391	**0.494**	0.410	0.405	0.485
xalan	2.4	2.5	0.153	0.112	0.298	0.168	0.134	**0.345**	0.288	0.244	0.417	0.303	0.268	**0.454**	0.353	0.336	0.426	0.359	0.345	**0.452**
	2.5	2.6	0.512	0.539	0.595	0.528	0.559	**0.639**	0.559	0.584	0.612	0.572	0.610	**0.671**	0.557	0.582	0.612	0.570	0.603	**0.668**
	2.4+2.5	2.6	0.263	0.263	0.505	0.292	0.289	**0.610**	0.390	0.390	0.560	0.414	0.411	**0.636**	0.404	0.403	0.557	0.418	0.415	**0.636**
xerces	1.2	1.3	0.154	0.110	0.204	0.184	0.119	**0.294**	0.294	0.241	0.418	0.322	0.254	**0.469**	0.354	0.334	0.428	0.367	0.339	**0.457**
	1.3	1.4	0.215	0.172	0.407	0.248	0.264	**0.536**	0.346	0.306	0.465	0.372	0.390	**0.584**	0.379	0.359	0.466	0.394	0.400	**0.555**
	1.2+1.3	1.4	0.096	0.092	0.396	0.119	0.120	**0.402**	0.224	0.219	0.461	0.251	0.253	**0.468**	0.328	0.327	0.462	0.338	0.338	**0.466**
	AVG		0.320	0.312	0.414	0.364	0.352	**0.480**	0.430	0.422	0.540	0.468	0.461	**0.592**	0.449	0.446	0.534	0.477	0.470	**0.580**
	$p(10^{-4})$		0.023	0.020	0.457	0.149	0.148		0.019	0.017	0.783	0.162	0.130		0.012	0.012	0.050	0.286	0.130	
	d		0.522	0.530	0.283	0.397	0.442		0.667	0.634	0.340	0.506	0.538		0.672	0.657	0.308	0.518	0.568	

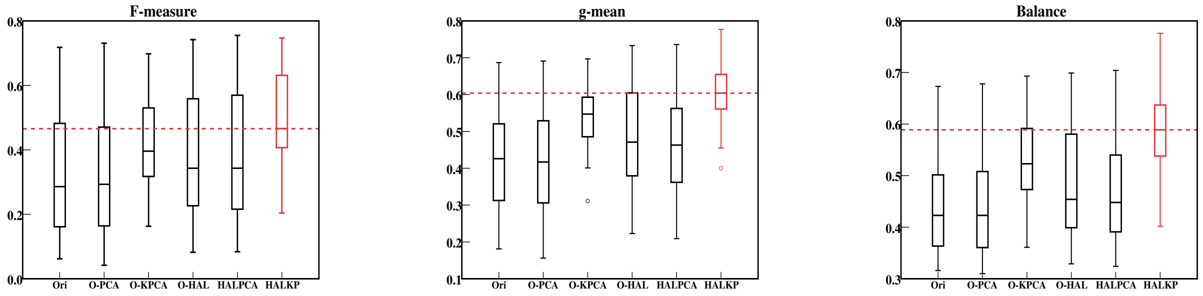

Fig. 2: Box-plots of HALKP and its five variations on three performance indicators under threshold 10%.

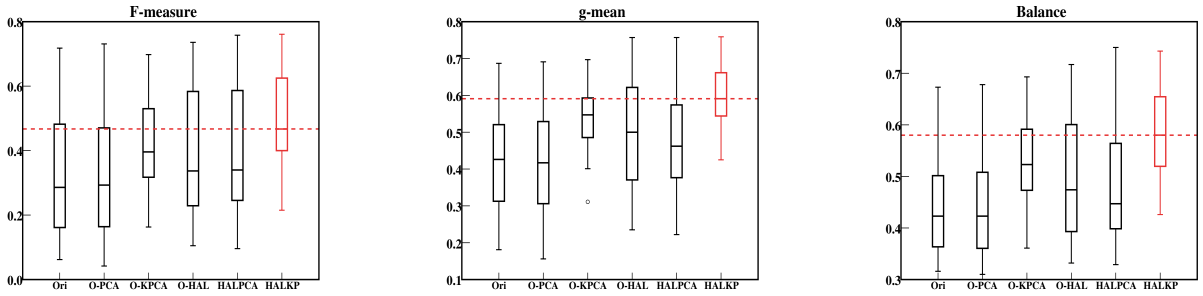

Fig. 3: Box-plots of HALKP and its five variations on three performance indicators under threshold 15%.

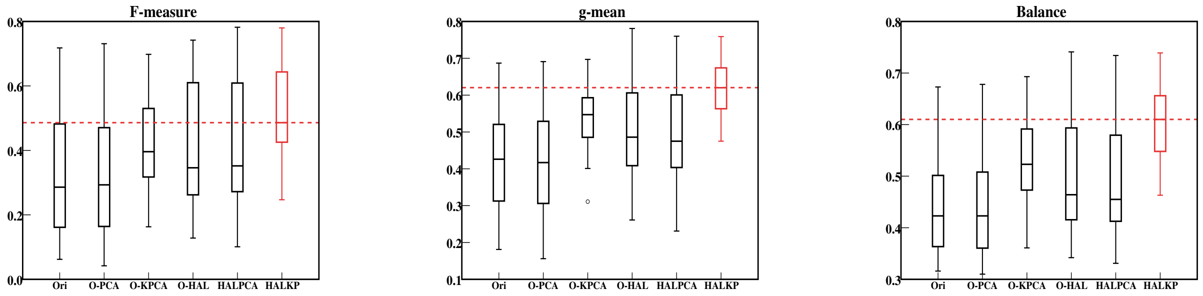

Fig. 4: Box-plots of HALKP and its five variations on three performance indicators under threshold 20%.

TABLE IV: Optimal β for Three HAL Based Methods for Each Cross-Version Pair under Four Thresholds

Threshold	Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
5%	O-HAL	0.0	0.4	0.1	0.1	0.7	0.3	0.1	0.6	0.4	0.4	0.0	0.3	0.0	1.0	1.0	0.0	0.4	0.2	0.7	0.6	0.3	0.7	0.1	0.1	0.9	0.0	0.4	0.1	0.4	0.3	0.2
	HALPCA	0.0	0.4	0.1	0.3	1.0	0.6	0.0	0.6	0.5	0.0	0.0	0.3	0.3	0.4	0.9	0.0	0.8	1.0	0.4	0.3	0.0	0.9	1.0	0.7	0.8	0.6	0.8	0.1	1.0	0.6	0.0
	HALKP	0.2	0.3	0.1	0.2	0.3	0.4	0.8	0.0	0.8	0.4	0.0	0.1	0.4	0.0	0.5	0.6	0.6	0.0	0.1	0.9	0.0	0.2	1.0	0.0	0.8	0.0	0.3	0.7	0.6	0.2	0.0
10%	O-HAL	0.0	0.4	0.1	0.1	0.1	0.0	0.2	1.0	0.4	0.5	0.0	0.0	0.0	0.9	1.0	1.0	0.2	0.0	0.3	0.7	0.0	0.4	0.6	0.1	0.7	0.3	0.3	0.3	0.4	0.3	0.0
	HALPCA	0.0	0.4	0.1	0.4	0.4	0.4	0.8	0.3	0.5	0.5	0.0	0.3	0.2	0.0	1.0	0.7	0.6	0.3	0.4	0.1	0.1	0.4	0.1	0.7	0.7	0.5	0.3	0.1	0.1	0.2	0.3
	HALKP	0.4	0.8	0.8	0.2	0.0	0.0	0.0	0.0	0.1	0.2	0.1	0.3	0.3	0.8	1.0	0.3	0.7	0.0	0.0	0.3	0.8	0.5	0.1	0.0	0.3	0.0	1.0	0.7	0.3	0.0	0.3
15%	O-HAL	0.1	0.1	0.1	0.1	0.1	0.0	0.2	0.6	0.9	0.5	0.0	0.3	0.5	0.2	1.0	0.2	0.2	0.0	0.3	0.5	0.3	0.4	0.1	0.1	0.6	0.2	1.0	0.1	0.7	0.0	0.0
	HALPCA	0.0	1.0	0.1	0.2	0.0	0.3	0.2	0.6	0.9	0.5	0.2	0.3	0.0	0.1	1.0	0.2	0.7	0.5	0.3	0.1	0.8	0.4	0.2	0.4	0.3	0.7	1.0	0.0	0.0	0.5	0.3
	HALKP	0.3	0.2	0.0	0.2	0.0	0.1	0.1	1.0	0.7	0.5	0.1	0.6	0.2	0.1	1.0	0.6	0.1	0.0	0.1	0.5	0.9	0.3	0.1	0.1	1.0	0.0	0.7	0.4	0.2	0.0	0.0
20%	O-HAL	0.1	0.5	0.1	0.1	0.0	0.0	0.4	0.1	0.3	0.5	0.0	0.3	0.3	0.1	1.0	0.6	0.2	0.0	0.3	0.6	0.6	0.4	0.1	0.1	0.6	0.2	1.0	0.2	0.4	0.0	0.0
	HALPCA	0.0	0.3	0.4	0.2	0.6	0.2	0.4	0.6	0.5	0.8	0.2	0.0	0.2	0.1	1.0	0.6	0.7	0.1	0.4	0.4	1.0	0.4	0.8	0.6	0.3	0.1	1.0	0.4	0.0	0.4	0.3
	HALKP	0.0	0.1	0.8	0.9	0.0	0.3	0.0	0.6	0.0	0.9	0.0	0.2	0.0	0.1	0.8	0.4	0.2	0.0	0.0	0.5	1.0	0.8	0.7	0.0	0.0	0.0	0.2	0.1	0.3	0.2	0.0

version. Among the three methods, O-KPCA obtains the best median values of all indicators compared with Ori and O-PCA. Since they differ in the preprocessing towards the features, the observation implies that the features extracted by non-linear method KPCA are beneficial to building a more effective model than the original features and the features extracted by linear method PCA. In addition, since all the three methods are outperformed by HALKP in terms of the three indicators under all four thresholds, it implies that the active learning can further improve the CVDP performance through supplementing useful structure information of current version into the prior version.

For the last three methods, the median values of the three indicators by HALKP are much higher than that by O-HAL, HALPCA under all thresholds. Since they differ in the preprocessing towards the features, the observation also implies that the features extracted by KPCA are still effective to improve CVDP performance after active learning.

In addition, we also report the optimal β values of O-HAL, HALPCA and HALKP for each cross-version pair under the four thresholds in Table IV. The index numbers in the first row correspond to the sequence numbers of the cross-version pairs in Table III from top to bottom. The table shows that the optimal β values vary across different cross-version pairs. For the same cross-version pair, the optimal β values also vary among different methods. In addition, $\beta = 1$ manifests that uncertainty measure based active learning achieves the best CVDP performance. From the table, we observe that there are only few cases in which the three methods with $\beta = 1$ achieve the best results. In terms of HALKP, there is only 1, 2, 3, 1 case under four thresholds, respectively. In most cases, HALKP with $\beta \in (0, 1)$ can achieve the best performance. These observations show that the candidate modules selected by HAL can provide more useful information into prior version to promote an more effective model compared with the modules selected by uncertainty measure based active learning.

RQ2: Is HALKP effective than other methods for CVDP?
Method: An simple method for selecting the candidate modules is random sampling from the current version. In this question, we investigate whether the candidate modules selected by HAL are more effective to enhance the data of prior version for adapting the remaining unlabeled modules compared with the modules selected by random sampling. For the purpose, we design three baseline methods based on the randomly selected candidate modules: the method that **O**nly uses **R**andom **S**ampling (**O-RS**), the method that incorporates **R**andom **S**ampling with **PCA** (**RSPCA**), and the method that combines **R**andom **S**ampling with **KPCA** (**RSKPCA**).

Recently, Lu et al. [7] designed a state-of-the-art method for CVDP. They used an uncertainty based **A**ctive **L**earning method to select the candidate modules after performing a dimensionality reduction with **MDS**. We name this method as **MDSAL**. Since the authors did not point out the feature dimensions they remained, we followed previous studies [74], [75] to assign the feature dimension. In [74], the authors suggested that selecting $\lceil \log_2 m \rceil$ features (where m denotes the total number of original features) is appropriate for defect prediction, while in [75], the authors pointed out that remaining 15% of the total number of features can also achieve satisfactory performance for defect prediction. Therefore, in this work, we implement two versions of MDSAL method according to the two selection strategies of feature dimension, and name the two baseline methods as MDSAL-1 and MDSAL-2, respectively.

Results: Table V presents the results of various indicators under threshold 5%. The table shows that HALKP achieves the best average performance over all three indicators. More concretely, in terms of F-measure, HALKP is superior to the baseline methods on 19 out of 31 cross-version pairs and the average F-measure by HALKP (0.480) gains improvements between 14.8% and 33.7% over the baseline methods; in terms of g-mean, HALKP outperforms the baseline methods on 23 out of 31 cross-version pairs and the average g-mean by HALKP (0.592) gains improvements between 9.7% and 27.6% over the baseline methods; in terms of Balance, HALKP is better than the baseline methods on 23 out of 31 cross-version pairs and the average Balance by HALKP (0.580) gains improvements between 9.0% and 22.4% over the baseline methods. Since the difference between RSKPCA and HALKP is the strategy of selecting the candidate modules of the current version, the experimental results indicate that HAL selects more useful unlabeled modules for the enhanced training data for CVDP compared with random sampling.

From the detailed results of the six methods on three indicators under threshold 10%, 15% and 20% in our additional materials [73], HALKP also achieves the best average results in terms of three indicators compared with the five baseline methods. The p and d values indicate that there exist statistically significant and substantial differences between HALKP and the five baseline methods under all four thresholds.

RQ3: How does the sequence of active learning and feature extraction impact the performance of CVDP?

TABLE V: The Detailed Results for HALKP and Five Baseline Methods on Each Cross-Version Pair under Threshold 5%

Cross-Version Pair			F-measure						g-mean						Balance					
Project	Prior	Current	O-RS	RS-PCA	RS-KPCA	MDSAL-1	MDSAL-2	HALKP	O-RS	RS-PCA	RS-KPCA	MDSAL-1	MDSAL-2	HALKP	O-RS	RS-PCA	RS-KPCA	MDSAL-1	MDSAL-2	HALKP
ant	1.3	1.4	0.187	0.162	0.286	0.098	0.328	**0.359**	0.340	0.315	0.473	0.235	0.496	**0.538**	0.386	0.369	0.472	0.333	0.483	**0.527**
	1.4	1.5	0.294	**0.333**	0.230	0.182	0.298	0.316	0.522	0.550	0.563	0.358	0.466	**0.674**	0.504	0.523	0.562	0.387	0.452	**0.671**
	1.5	1.6	0.286	0.282	0.369	0.198	0.198	**0.383**	0.416	0.412	0.509	0.339	0.336	**0.530**	0.418	0.417	0.491	0.375	0.373	**0.510**
	1.3+1.4+1.5	1.6	0.290	0.242	0.368	0.299	0.255	**0.422**	0.419	0.379	0.512	0.430	0.388	**0.565**	0.421	0.397	0.495	0.427	0.401	**0.542**
camel	1.2	1.4	0.316	0.317	0.361	**0.423**	0.404	0.401	0.507	0.480	0.624	**0.659**	0.631	0.645	0.491	0.464	0.622	**0.646**	0.616	0.638
	1.4	1.6	0.175	0.156	0.369	0.240	0.340	**0.406**	0.320	0.296	0.558	0.388	0.494	**0.581**	0.367	0.356	0.539	0.403	0.475	**0.557**
	1.2+1.4	1.6	0.242	0.222	0.361	0.382	0.342	**0.429**	0.397	0.370	0.543	0.538	0.512	**0.583**	0.409	0.393	0.525	0.512	0.493	**0.552**
ivy	1.1	1.4	0.151	**0.204**	0.128	0.186	0.179	0.165	0.570	**0.675**	0.518	0.664	0.626	0.583	0.569	**0.663**	0.517	0.655	0.625	0.583
	1.4	2.0	0.123	0.084	0.252	0.056	0.105	**0.349**	0.268	0.197	0.514	0.171	0.239	**0.586**	0.349	0.327	0.502	0.314	0.333	**0.558**
	1.1+1.4	2.0	0.294	0.298	0.273	**0.306**	0.203	0.300	0.488	0.487	0.563	0.536	0.421	**0.583**	0.471	0.470	0.550	0.511	0.427	**0.564**
jedit	3.2	4.0	0.533	0.536	0.565	0.570	0.614	**0.634**	0.678	0.685	0.718	0.730	0.755	**0.772**	0.662	0.673	0.715	0.724	0.749	**0.769**
	4.0	4.1	0.556	0.523	0.519	0.486	0.483	**0.591**	0.646	0.624	0.660	0.598	0.601	**0.732**	0.601	0.583	0.643	0.560	0.565	**0.724**
	3.2+4.0	4.1	0.578	0.545	0.531	**0.647**	0.569	0.564	0.689	0.667	0.674	**0.739**	0.691	0.708	0.659	0.640	0.662	**0.707**	0.665	0.699
log4j	1.0	1.1	0.573	0.624	0.564	0.717	**0.759**	0.632	0.654	0.693	0.656	0.759	**0.792**	0.712	0.618	0.654	0.635	0.712	**0.749**	0.683
	1.1	1.2	0.518	0.458	0.597	0.481	0.510	**0.725**	**0.537**	0.496	0.479	0.511	0.528	0.401	**0.525**	0.489	0.478	0.502	0.518	0.410
	1.0+1.1	1.2	0.407	0.384	0.491	0.563	0.549	**0.613**	0.490	0.473	0.535	0.607	0.559	**0.622**	0.472	0.460	0.513	0.569	0.545	**0.598**
poi	1.5	2.0	0.172	0.163	0.193	0.222	0.235	**0.259**	0.449	0.418	0.489	0.556	0.554	**0.585**	0.451	0.421	0.490	0.553	0.552	**0.584**
	2.0	2.5	0.096	0.061	0.347	0.127	0.163	**0.604**	0.223	0.176	0.434	0.259	0.296	**0.616**	0.329	0.315	0.443	0.341	0.356	**0.606**
	1.5+2.0	2.5	0.145	0.133	0.498	0.562	0.567	**0.675**	0.276	0.264	0.544	0.604	0.603	**0.683**	0.349	0.344	0.536	0.579	0.582	**0.667**
synapse	1.0	1.1	0.083	0.192	0.342	0.250	0.182	**0.426**	0.207	0.330	0.507	0.392	0.329	**0.574**	0.325	0.373	0.504	0.405	0.372	**0.559**
	1.1	1.2	0.408	0.412	0.422	0.424	**0.504**	0.479	0.525	0.523	0.546	0.534	0.587	**0.591**	0.504	0.498	0.534	0.508	0.543	**0.584**
	1.0+1.1	1.2	0.391	0.359	0.383	0.427	0.407	**0.458**	0.501	0.475	0.510	0.526	0.521	**0.568**	0.477	0.458	0.498	0.492	0.496	**0.544**
velocity	1.4	1.5	0.719	0.733	0.697	0.781	**0.806**	0.751	0.264	0.310	0.369	0.317	0.402	**0.493**	0.341	0.364	0.396	0.368	0.413	**0.488**
	1.5	1.6	0.562	0.546	0.532	0.560	**0.597**	0.596	0.587	0.549	0.569	0.578	0.650	**0.657**	0.570	0.533	0.559	0.562	0.636	**0.647**
	1.4+1.5	1.6	0.534	0.535	0.507	0.577	**0.578**	0.527	0.392	0.375	0.488	0.588	**0.601**	0.494	0.405	0.395	0.485	0.565	**0.578**	0.485
xalan	2.4	2.5	0.161	0.119	0.320	0.191	0.250	**0.345**	0.295	0.252	0.435	0.326	0.379	**0.454**	0.356	0.339	0.438	0.370	0.398	**0.452**
	2.5	2.6	0.520	0.540	0.602	0.638	**0.678**	0.639	0.571	0.595	0.612	0.664	**0.694**	0.671	0.568	0.589	0.611	0.663	**0.694**	0.668
	2.4+2.5	2.6	0.276	0.276	0.522	**0.635**	0.590	0.610	0.401	0.400	0.584	**0.685**	0.648	0.636	0.411	0.409	0.574	**0.657**	0.625	0.636
xerces	1.2	1.3	0.152	0.116	0.203	0.177	0.124	**0.294**	0.291	0.248	0.414	0.341	0.299	**0.469**	0.354	0.337	0.426	0.378	0.359	**0.457**
	1.3	1.4	0.243	0.210	0.445	0.275	0.269	**0.536**	0.371	0.342	0.500	0.397	0.394	**0.584**	0.391	0.376	0.488	0.406	0.403	**0.555**
	1.2+1.3	1.4	0.092	0.095	0.399	0.171	0.194	**0.402**	0.219	0.223	0.464	0.302	0.326	**0.468**	0.327	0.328	0.458	0.360	0.369	**0.466**
	AVG		0.325	0.318	0.409	0.382	0.396	**0.480**	0.436	0.428	0.534	0.495	0.510	**0.592**	0.454	0.450	0.528	0.501	0.511	**0.580**
	$p(10^{-4})$		0.020	0.043	0.012	2.776	8.940		0.047	0.045	0.104	7.229	6.735		0.045	0.027	0.149	2.382	4.860	
	d		0.520	0.518	0.287	0.295	0.295		0.626	0.605	0.407	0.341	0.318		0.630	0.635	0.377	0.367	0.366	

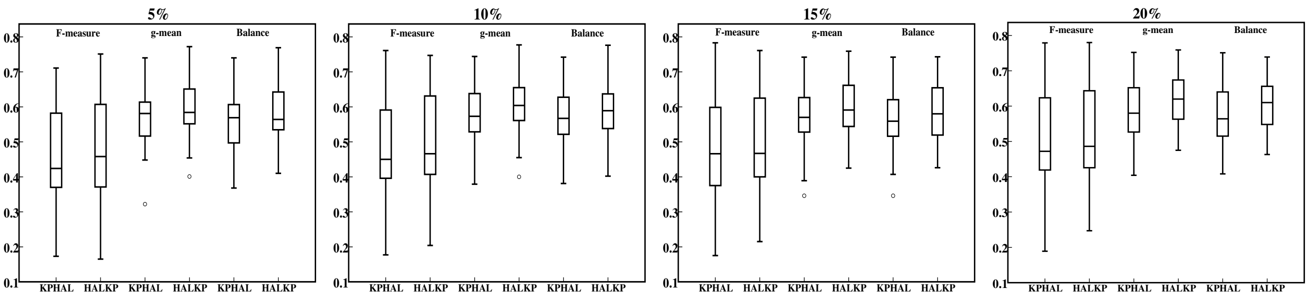

Fig. 5: Box-plots of KPHAL and HALKP on three indicator values under four thresholds.

Method: As mentioned above, our two phased CVDP framework consists of active learning stage and feature extraction stage. Different from the CVDP method [7] that conducted dimensionality reduction before active learning, we first apply a hybrid active learning method on the original data of the two versions, then employ a non-linear extraction method to the new data of two versions. In this question, we investigate whether the sequence of active learning and feature extraction impacts our experimental results. For this purpose, we design a baseline method that first uses feature extraction method **KP**CA to map the original data of two versions, and then apply the active learning method **HAL** to the mapped data. We name this baseline method as **KPHAL**.

Result: Figure 5 depicts the box-plots of HALKP and KPHAL on the three indicators under four thresholds. It shows that the median F-measure by HALKP is higher than that by KPHAL, while the median g-mean and Balance by two methods have very small differences under threshold 5%; the median values of all indicators by HALKP are superior to that by KPHAL under threshold 10%, but the differences are not obvious; the median g-mean and Balance by HALKP are better than that by KPHAL, while the median F-measure by the two methods are the same under threshold 15%; the median g-mean and Balance by HALKP are much higher than that by KPHAL, while the median F-measure by HALKP is slightly higher than that by KPHAL under threshold 20%. In summary, HALKP is better than KPHAL, especially under larger threshold.

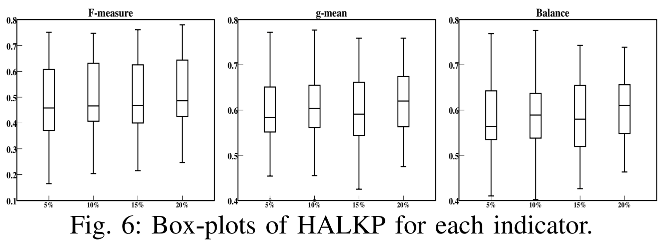

Fig. 6: Box-plots of HALKP for each indicator.

VI. Discussion

A. *How Many Candidate Modules Selected From Current Version Are Sufficient?*

Figure 6 depicts the box-plots of HALKP under the four thresholds for each indicator. From the figure, we observe that the median indicator value gradually increases as the threshold increases except for the threshold 15%. In addition, we observe that the median values of the three indicators under threshold

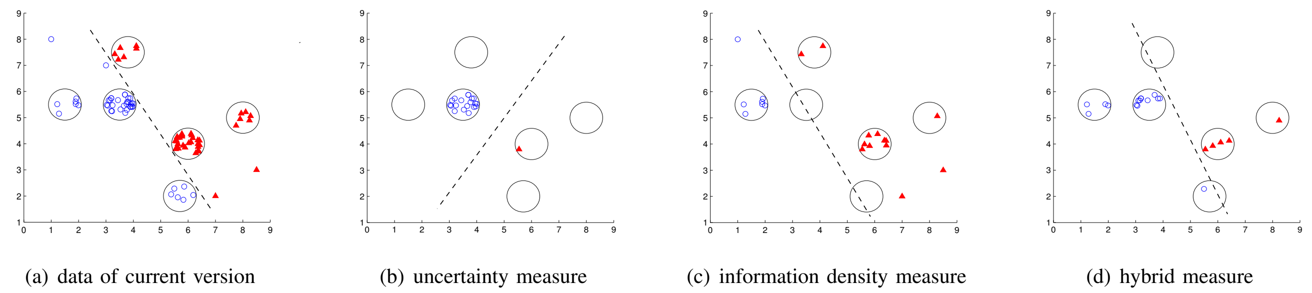

Fig. 7: An synthetic example for presenting the selected samples by three active learning methods.

10% have no big differences compared with the corresponding median values under threshold 20%. Since active learning is expected to improve the CVDP performance with minimum manual annotation, selecting 10% candidate modules for labeling may be enough for HALKP to achieve acceptable performance considering the labeling cost in practice.

B. Why Does Our Hybrid Active Learning Framework Work?

Since the differences between active learning methods lie in the candidate samples selected, we perform an empirical study on a synthetic dataset to manifest the importance of the selected samples. Subfigure 7(a) is a synthetic dataset with two-class samples. The samples of each class are represented by different legends and colors. The dotted line denotes the decision boundary. To simulate CVDP, we take the synthetic dataset as the data of current version. Since plenty of information of current version are derived from its prior version, we make a perturbation on the coordinate values of the data points to construct the data of prior version. We observe the differences of the three active learning methods by sequentially selecting 20 samples and show them on the plane.

Subfigure 7(b) presents the selected samples by uncertainty measure based active learning method that favors informative samples. We can see that most selected samples are close to actual decision boundary but are not balanced in two classes, leading to incorrect decision boundary. Subfigure 7(c) presents the selected samples by information density measure based active learning method that favors representative samples. It shows that this method selects representative samples from most data clusters. As the isolated sample can be treated as a simple cluster, this method may select such sample as the representative of the special cluster. These representative samples could obtain the decision boundary that approaches to the actual one. Subfigure 7(d) presents the selected samples by our HAL method with $\beta = 0.5$ that favors both informative and representative samples. We can see that the selected samples have the potential to obtain the decision boundary that is very close to the actual one.

VII. Threats to Validity

External Validity. Although we conduct experiments on an open source software defect dataset that has been extensively used in previous studies, we still cannot claim that our experimental findings can be generalized to all kinds of defect datasets. Further investigations on other datasets are needed.
Internal Validity. Since the scholars in [7] did not provide their experimental codes, we carefully replicate their method according to the descriptions and implement the random forest's similarity based MDS with standard R package as suggested by the scholars. For the parameter setting, such as β in HAL, more cautious controlled experiments should be performed. In addition, we just simply label the candidate unlabeled modules with the actual labels instead of acquiring their oracles from the experts. This may not well represent the real application scenario.
Construct Validity. We use three extensively-used measurement indicators to evaluate the performance of our HALKP framework and baseline methods for CVDP. Other comprehensive indicators, such as AUC can also be considered. In addition, we rigorously verify the superiority of our method towards the baseline methods with statistical test and effect size.

VIII. Conclusion and Future Work

We propose a novel CVDP framework HALKP via a hybrid active learning strategy HAL with a non-linear feature extraction method KPCA. HAL selects some unlabeled modules from current version, which are informative and representative for the data of the current version, for querying the labels. These selected modules and the modules of prior version form an enhanced training set for the remaining unlabeled modules of the current version. KPCA converts the data of two versions into an embedding space where the mapped features reveal the intrinsic structure of the original ones. We evaluate the new framework on 31 versions of 10 projects from a public defect dataset. The experimental results show that HALKP outperforms the baseline methods, including five variation methods, three random sampling based methods, and two versions of a state-of-the-art method. The results also indicate that active learning prior to feature extraction tends to achieve relatively better CVDP performance.

Our future work involves applying the proposed framework to other types of defect datasets and exploring how HALKP can be extend to CPDP and HCPDP scenarios.

Acknowledgment

The authors acknowledge the support provided by the grants of the National Natural Science Foundation of China (61572374, 61472423, U1636220, 61602258), Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012), China Postdoctoral Science Foundation (No. 2017M621247), Hong Kong GRF (No. 152279/16E, 152223/17E), Hong Kong RGC Project (No. CityU C1008-16G).

References

[1] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[2] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24rd International Conference on Software Engineering*. IEEE, 2002, pp. 547–550.

[3] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[4] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. ACM, 2006, pp. 8–17.

[5] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[6] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," *arXiv preprint arXiv:1708.02368*, 2017.

[7] H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," in *Proceedings of the 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 312–322.

[8] S. J. Huang, R. Jin, and Z. H. Zhou, "Active learning by querying informative and representative examples." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 10, pp. 1936–1949, 2014.

[9] X. Li and Y. Guo, "Adaptive active learning for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 859–866.

[10] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering*, vol. 23, no. 4, pp. 569–590, 2016.

[11] F. Liu, X. Gao, B. Zhou, and J. Deng, "Software defect prediction model based on pca-isvm," *Computer Simulation*, 2014.

[12] H. Cao, Z. Qin, and T. Feng, "A novel pca-bp fuzzy neural network model for software defect prediction," *Advanced Science Letters*, vol. 9, no. 1, pp. 423–428, 2012.

[13] C. Zhong, "Software quality prediction method with hybrid applying principal components analysis and wavelet neural network and genetic algorithm," *International Journal of Digital Content Technology and Its Applications*, vol. 5, no. 3, 2011.

[14] Y. Rathi, S. Dambreville, and A. R. Tannenbaum, "Statistical shape analysis using kernel pca." Georgia Institute of Technology, 2006.

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

[16] F. Zhang, I. Keivanloo, and Y. Zou, "Data transformation in cross-project defect prediction," *Empirical Software Engineering*, pp. 1–33, 2017.

[17] L. Shi, P. He, and E. Liu, "An incremental nonlinear dimensionality reduction algorithm based on isomap," *AI 2005: Advances in Artificial Intelligence*, pp. 892–895, 2005.

[18] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 1997, pp. 583–588.

[19] C. Liu, "Gabor-based kernel pca with fractional power polynomial models for face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 5, pp. 572–581, 2004.

[20] S. W. Choi, C. Lee, J.-M. Lee, J. H. Park, and I.-B. Lee, "Fault detection and identification of nonlinear processes based on kernel pca," *Chemometrics and Intelligent Laboratory Systems*, vol. 75, no. 1, pp. 55–67, 2005.

[21] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[22] 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*. IEEE Press, 2015, pp. 789–800.

[23] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[24] 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, 2017.

[25] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, 2017.

[26] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 35th International Conference on Software Engineering*. IEEE Press, 2013, pp. 382–391.

[27] 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.

[28] 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 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 496–507.

[29] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[30] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, 2002.

[31] 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, 2009.

[32] 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*. ACM, 2009, pp. 91–100.

[33] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 61.

[34] C. Mao, "Software faults prediction based on grey system theory," *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 2, pp. 1–6, 2009.

[35] B. Caglayan, A. Bener, and S. Koch, "Merits of using repository metrics in defect prediction for open source projects," in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. IEEE Computer Society, 2009, pp. 31–36.

[36] K. E. Bennin, K. Toda, Y. Kamei, J. Keung, A. Monden, and N. Ubayashi, "Empirical evaluation of cross-release effort-aware defect prediction models," in *Proceedings of the International Conference onSoftware Quality, Reliability and Security*. IEEE, 2016, pp. 214–221.

[37] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research*, vol. 2, no. Nov, pp. 45–66, 2001.

[38] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, "Selective sampling using the query by committee algorithm," *Machine learning*, vol. 28, no. 2, pp. 133–168, 1997.

[39] N. Roy and A. McCallum, "Toward optimal active learning through sampling estimation of error reduction," pp. 441–448, 2001.

[40] S. Dasgupta and D. Hsu, "Hierarchical sampling for active learning," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 208–215.

[41] H. T. Nguyen and A. Smeulders, "Active learning using pre-clustering," in *Proceedings of the 21st International Conference on Machine Learning*. ACM, 2004, p. 79.

[42] G. Luo, K. QIN *et al.*, "Active learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 95, no. 6, pp. 1680–1683, 2012.

[43] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.

[44] H. Lu and B. Cukic, "An adaptive approach with active learning in software fault prediction," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*. ACM, 2012, pp. 79–88.

[45] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[46] J. Peng and D. R. Heisterkamp, "Kernel indexing for relevance feedback image retrieval," in *Proceedings of the International Conference on Image Processing*, vol. 1. IEEE, 2003, pp. I–733.

[47] J.-B. Li, S.-C. Chu, and J.-S. Pan, "Kernel principal component analysis (kpca)-based face recognition," *Kernel Learning Algorithms for Face Recognition*, pp. 71–99, 2014.

[48] Smola and Alexander, *Learning with kernels*. MIT Press, 2002.

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

[50] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, and Z. Zhang, "Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 331–357, 2015.

[51] Y. Yang, M. Harman, J. Krinke, S. Islam, D. Binkley, Y. Zhou, and B. Xu, "An empirical study on dependence clusters for effort-aware fault-proneness prediction," in *Proceedings of the 31st International Conference on Automated Software Engineering*. IEEE, 2016, pp. 296–307.

[52] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering*. IEEE, 2014, pp. 164–173.

[53] A. Mockus, P. Zhang, and P. L. Li, "Predictors of customer perceived software quality," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 225–233.

[54] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 852–864, 2010.

[55] S. C. Hoi, R. Jin, and M. R. Lyu, "Large-scale text categorization by batch mode active learning," in *Proceedings of the 15th International Conference on World Wide Web*. ACM, 2006, pp. 633–642.

[56] A. I. Schein and L. H. Ungar, "Active learning for logistic regression: an evaluation," *Machine Learning*, vol. 68, no. 3, pp. 235–265, 2007.

[57] "Morph dataset," http://openscience.us/repo/defect/ck/.

[58] 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.

[59] S. Herbold, "Training data selection for cross-project defect prediction," in *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*. ACM, 2013, p. 6.

[60] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Proceedings of the 9th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2015, pp. 96–103.

[69] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[61] 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*. ACM, 2010, p. 9.

[62] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[63] Z. Xu, J. Xuan, J. Liu, and X. Cui, "Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering," in *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*, vol. 1. IEEE, 2016, pp. 370–381.

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

[65] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[66] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 414–423.

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

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

[70] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*. IEEE, 2011, pp. 1–10.

[71] G. Macbeth, E. Razumiejczyk, and R. D. 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.

[72] N. Cliff, *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[73] "Supplementary material," https://defect-prediction.github.io/halkp/.

[74] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579–606, 2011.

[75] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, 2013.