The Journal of Systems and Software 154 (2019) 59–78

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

TSTSS: A two-stage training subset selection framework for cross version defect prediction☆

Zhou Xu[a,b,i], Shuai Li[b], Xiapu Luo[b,*], Jin Liu[a,c,d,*], Tao Zhang[e], Yutian Tang[b], Jun Xu[f], Peipei Yuan[g], Jacky Keung[h]

[a] School of Computer Science, Wuhan University, Wuhan, China
[b] Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[c] Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[d] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China
[e] College of Computer Science and Technology, Harbin Engineering University, Harbin, China
[f] Inception Institute of Artificial Intelligence, Abu Dhabi, UAE
[g] School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China
[h] Department of Computer Science, City University of Hong Kong, Hong Kong
[i] College of Informatics, Huazhong Agricultural University, Wuhan, China

ARTICLE INFO

Article history:
Received 14 August 2018
Revised 21 March 2019
Accepted 22 March 2019
Available online 23 March 2019

MSC:
00–01
99-00

Keywords:
Cross version defect prediction
Spare modeling
Training subset selection
Weighted extreme learning machine

ABSTRACT

Cross Version Defect Prediction (CVDP) is a practical scenario by training the classification model on the historical data of the prior version and then predicting the defect labels of modules in the current version. Unfortunately, the differences of data distribution across versions may hinder the effectiveness of the trained CVDP model. Thus, it is not trivial to select a suitable training subset from the prior version to promote the CVDP performance. In this paper, we propose a novel method, called Two-Stage Training Subset Selection (TSTSS), to address this challenging issue. In the first stage, TSTSS utilizes a sparse modeling representative selection method to select an initial module subset from the prior version which can well reconstruct the data of the prior version. In the second stage, TSTSS leverages a dissimilarity-based sparse subset selection method to further refine the selected module subset, which enables the selected modules to well represent the modules of the current version. Finally, we use a novel weighted extreme learning machine classifier to construct the CVDP model. We evaluate the CVDP performance of TSTSS on 50 cross-version pairs using 6 indicators. The experiments show that TSTSS can efficiently improve the CVDP performance compared with 11 baseline methods.

1. Introduction

Nowadays, software products are pervasive in our daily life. It is an important goal to develop high-quality software (Moha, 2007). As the software functions are increasingly complicated, the software modules (such as the methods, classes, and packages) undergo frequent changes during the version update (Catolino et al., 2017). However, because of some unpredictable factors during the design, development or deployment process, it is inevitable to introduce the defects into the software product, which threatens the quality of the software products (Ghaffarian and Shahriari, 2017).

Before releasing the products, effectively detecting and fixing the defects is critical for software quality assurance. Thanks to the prevalence of the version control systems and the issue tracking systems in software development lifestyle, it is available to access the historical software development data from these software repositories. In the past decade, researchers have proposed various defect prediction methods to identify the potentially defective modules for improving software quality. These methods analyze the historical data and explore the inherent relationship between the software module features (such as code complexity features and process features) and the defect information which can be collected from the software repositories (Kamei and Shihab, 2016).

Many studies on defect prediction investigate the classification performance of machine learning methods for **W**ithin **P**roject **D**efect **P**rediction (**WPDP**) scenario (Lessmann et al., 2008; Ghotra et al., 2015; Xu et al., 2016a). Generally, WPDP trains a classification model on labeled software modules, and then tests on the

☆ Fully documented templates are available in the elsarticle package on http://www.ctan.org/tex-archive/macros/latex/contrib/elsarticle CTAN.

* Corresponding authors.
E-mail addresses: csxluo@comp.polyu.edu.hk (X. Luo), jinliu@whu.edu.cn (J. Liu).

https://doi.org/10.1016/j.jss.2019.03.027
0164-1212/

unlabeled modules within the same project. Most existing studies conduct WPDP by using cross-validation methods to partition the defect data of a specific project version into the training set and test set, which is also called **I**nner **V**ersion **D**efect **P**rediction (**IVDP**). For a mature project with multiple versions, it is more practical to use the historical defect data of the prior version to conduct defect prediction on the unlabeled modules of the upcoming version (a.k.a. current version), i.e., **C**ross **V**ersion **D**efect **P**rediction (**CVDP**) (Lu et al., 2014; Yang and Wen, 2018). However, not many studies have specialized in the CVDP topic.

1.1. Motivation

CVDP has some unique characteristics over IVDP. For example, in IVDP scenario, as the training set and test set are usually derived from the same defect data of a specific project version, the data distributions of the two sets are identical, which conforms to the similar distribution assumption of the training set and test set for most classification models (Bin et al., 2017). By contrast, in CVDP scenario, as the software functions are increasingly complicated, the modules undergo frequent changes during the version update. For example, the current version of the project inherits, refactories and deletes some existing modules from the prior version, or adds some new modules (Catolino et al., 2017; Lu et al., 2014). These operations can cause a certain degree of distribution differences of the data across versions. Such distribution differences may result in the CVDP model built on the data of the prior version failing to achieve satisfactory prediction performance for the current version.

To narrow the gap of the distribution difference between the cross-version data, we propose a novel **T**wo-**S**tage **T**raining **S**ubset **S**election (**TSTSS**) method to select a module subset from the prior version that is optimal for the data of the current version. More specifically, in the first stage, TSTSS employs a **S**parse **M**odeling **R**epresentative **S**election (**SMRS**) method (Vidal et al., 2012) to simplify the data of the prior version by selecting an important module subset. The goal is to find a compact representation of the data in the prior version and expect that these important modules can reconstruct the original data to the maximum extent. In addition, the advantage of this simplification is beneficial to save memory, improve the understanding and interpretation of the data (Vidal et al., 2012). As this step retains the important modules towards the whole data and eliminates the unprofitable modules, it is also helpful to improve the performance of models built on this module subset and save the computational time. Since this module reduction process just selects a module subset to well construct the original data without involving the participation of the data in the current version, thus we call this stage as a self-simplification process. However, the modules selected in this stage cannot guarantee to be representative for the data of the current version, which means that the distribution differences of the data between the two versions have not been addressed yet. Hence, in the second stage, we further utilize a **D**issimilarity-based **S**pare **S**ubset **S**election (**DS3**) method (Elhamifar et al., 2016) to select a representative module subset from the prior version based on the pairwise dissimilarities between the modules of the two versions. The selected module subset can effectively represent each module of the current version, which promotes to relieve the distribution differences. As we preserve the modules of the prior version that well represent the modules of the current version and eliminate the irrelevant modules, the classification models built on this selected subset are more targeted to the data of the current version. Since this module reduction process requires the assistance of the data in the current version to further refine the selected modules in the first stage, thus we call this stage as an auxiliary-refining process. After completing the two simplification processes, the selected module subset not only contains the important information of the original data in the prior version, but also well expresses the data of the current version. Thus, the resulting module subset is expected to achieve encouraging performance of CVDP.

In addition, software defect data usually present class imbalance characteristic, i.e., the numbers of the defective and non-defective modules are usually inequality. This will lead general classification model bias to classify the modules as the majority class which contains more modules. In this work, we employ a novel weighted classifier, called **W**eighted **E**xtreme **M**achine **L**earning (**WELM**), to build the CVDP model. WELM is a weighted variant of ELM which is a **S**ingle-hidden **L**ayer **F**eedforward **N**eural network**s** (**SLFNs**). It alleviates the negative influence of the class imbalance by assigning higher weights to the modules of the minority class and lower weights to the modules of the majority class.

The goal of our study is to propose a novel data reduction method TSTSS to select an optimal training subset for CVDP task. In our empirical experiments, we evaluate the effectiveness of our TSTSS method for CVDP performance on 67 versions of 17 projects from an open-source defect dataset. We conduct extensive experiments on total 50 cross-version pairs with 6 evaluation indicators, including 3 traditional and 3 effort-aware indicators. The 3 traditional indicators, i.e., F-measure, **M**atthews **C**orrelation **C**oefficient (**MCC**) and **A**rea **U**nder the **C**urve (**AUC**), do not consider the cost of quality assurance efforts required to review the modules (Zimmermann et al., 2009; Turhan et al., 2009; Chen et al., 2015). As inspecting all the modules is not always feasible due to the constraint of the test resources, Mende and Koschke (2010) suggested that it is more realistic to use effort-aware indicators that take into account the inspection efforts in defect prediction. Thus, in this work, we further employ 3 effort-aware indicators, i.e., **E**ffort-**A**ware **P**recision (**EAP**), **E**ffort-**A**ware **R**ecall (**EAR**), and **E**ffort-**A**ware **F**-measure (**EAF**) to evaluate the CVDP performance. We calculate the 3 effort-aware indicators by improving a novel module ranking strategy (Huang et al., 2017).

As a result, our method TSTSS achieves encouraging CVDP performance against total 11 baseline methods, including 2 downgraded variant methods and 9 training subset selection methods. Compared with the best average performance among the 11 methods, TSTSS achieve an improvement of 38.4%, 44.3%, 5.1%, 5.4%, 6.9% in terms of average F-measure, MCC, EAP, EAR, and EAF, respectively. While, the average AUC by TSTSS is a little lower than that by one baseline method.

This paper extends our preliminary study (Xu et al., 2018a) published as a conference paper. There are 6 main differences among the two papers: (1) We improve the original one-stage training subset selection method in conference version to a two-stage selection strategy. The new strategy further reduces the data size by selecting a more refined training subset from the prior version. (2) To alleviate the negative impact of class imbalance issue which is not considered in conference version, we construct the prediction model with a weighted classifier which is designed to handle the imbalanced dataset, instead of a general logistic regression model in conference version. (3) We add 11 project versions to extend our study corpus scale. (4) We replace 2 performance indicators in conference version to the ones which are more suitable for evaluation on imbalanced data. This enables us to have a full overview of the prediction ability of our method. (5) We add 8 baseline methods to enrich the experiment comparison. (6) We make a more detailed analysis of the experimental results.

1.2. Contributions

In this paper, we highlight the following main contributions:

(1) We propose a novel TSTSS method to select an optimized training subset from the prior version for building a more effective CVDP model. Our TSTSS method first conducts self-simplification stage to reserve the important modules that can well reconstruct the original data of the prior version and filter out the useless ones. Then TSTSS performs auxiliary-refining stage to alleviate the distribution differences of the data across versions by selecting the most representative modules that can well represent each module of the current version.
(2) We employ a novel weighted classifier WELM to relieve the adverse effect of the class imbalance on the CVDP performance since the module numbers of the two classes usually vary greatly in many defect data. To our best knowledge, this is the first work to explore the effectiveness of WELM in CVDP scenario.
(3) We perform extensive experiments on total 50 cross-version pairs, derived from 67 versions of 17 projects. To our best knowledge, it is a first attempt to conduct such a large-scale empirical study for CVDP task.
(4) We employ 6 indicators, including 3 typical and 3 effort-aware ones, as our performance measurement. This enables us to perform a comprehensive evaluation on the effectiveness of our TSTSS method for CVDP performance. In addition, we enhance a novel module ranking method to calculate the effort-aware indicators.
(5) Compared with 11 baseline methods, the experimental results show that our TSTSS method achieves encouraging performance with fewer modules (no more than 50% of the original data) over more than half of cross-version pairs and outperforms the baseline methods in average.

1.3. Paper organization

The rest of this article is structured as follows: In Section 2, we describe the related work on CVDP and training subset selection studies in defect prediction. In Section 3, we present the technique details of our TSTSS method. In Section 4, we introduce the experimental design of our empirical study, such as the benchmark dataset, the evaluation indicators, the parameter setting and research questions. In Section 5, we report and analyze the experimental results. In Section 6, we point out the potential threats to validity. In Section 7, we conclude this paper and state the future work.

2. Related work

In this section, we introduce the related work of CVDP and training subset selection methods in **C**ross **P**roject **D**efect **P**rediction (**CPDP**) scenario since no study has designed such methods for CVDP scenario.

2.1. Cross version defect prediction

IVDP and CVDP differ in the source of the training set. Given a project, IVDP employs part of the data in the same version as the training set, while CVDP uses the historical data in the prior version of the same project as the training set. Compared with IVDP, CVDP is closer to the deployment in real application scenario where the historical version data of a project are used to assist in the identification of the defect-prone modules in its on-going version. However, to the best of our knowledge, only a fewer related studies have been devoted to CVDP.

Bennin et al. (2016) evaluated the CVDP performance of 11 classification models on 25 open source software projects (each with 2 versions) with an effort-aware indicator. The experimental results showed that M5 and K* models achieved the best performance but were also greatly affected by the defect ratio and size of the defect data. However, the performance differences among all 11 methods were not statistically significant.

Holschuh et al. (2009) explored the CVDP performance on a large software system by collecting 4 types of features. The experiments on 6 projects (each with 3 versions) showed that the overall performance is unsatisfactory.

Shukla et al. (2018) treated CVDP as a multi-objective optimization problem with 2 objective functions: i.e., maximizing recall by minimizing misclassification cost and the cost of quality assurance activities on defect prone modules. They used 4 traditional classification models to conduct experiments on 11 open source projects with total 30 cross-version pairs and found that multi-objective logistic regression outperformed 4 single-objective algorithms.

Li et al. (2018a) compared the CVDP performance of a multi-objective approach which optimized 2 objectives and a single-objective approach which optimized the trade-off of the 2 objectives. They conducted experiments on 4 open source software projects with total 10 cross-version pairs. The results indicated that when the trade-off was known, single-objective approach can achieve better performance which was not consistent with the conclusion in Shukla et al. (2018).

Yang et al. (2018) investigated the CVDP performance of 2 models, i.e., ridge regression and lasso regression. They conducted experiments on 11 projects with total 30 cross-version pairs and found that the 2 methods achieved better performance compared with 4 baseline methods. But this work focused on the ranking task, not the classification task as we do.

Although some previous studies (Monden et al., 2013; Khoshgoftaar and Seliya, 2003; Yang et al., 2015b; Wang et al., 2016a) also mentioned the CVDP concept, they did not specialize in CVDP, just treated it as one of multiple defect prediction scenarios, like IVDP and CPDP. Thus, we do not discuss them here. All these mentioned studies fed all modules of the prior version into the classification model without taking the distribution differences into account and conducted a small scale experiments on a few project versions. Different from these studies, our work focuses on selecting an optimal training modules to relieve the gap of distribution differences across versions and conducts a large scale experiments on total 50 cross-version pairs for CVDP task.

Recently, there are 2 studies that have considered the issue of distribution differences for CVDP task. Lu et al. (2014) selected some candidate unlabeled modules from the current version with an active learning method and labeled these modules by querying the software experts, then added them into the prior version to form a mixed training set. They expected that these modules could alleviate the distribution differences by supplementing some distribution information of the current version into the prior version. The experimental results on 3 Eclipse projects with total 6 cross-version pairs showed that their method could improve the CVDP performance. However, the candidate modules selected by the active learning method were only informative while not representative for the data of the current version (Li and Guo, 2013; Huang et al., 2014). Inspected by Lu et al.'s work, Xu et al. (2018b) proposed a hybrid active learning method to select both informative and representative modules from the current version and merge them into prior version to construct an enhanced training set. They conducted experiments on 10 open source projects with total 31 cross-version pairs and found that their method could further improve the CVDP performance compared with Lu et al.'s method. Both studies selected some modules from the current version and added them into the prior version, however, their methods needed to label the selected modules which involves in additional efforts. Different from the two studies which alleviated the data

distribution differences by adding some important modules of the current version into the prior version, our work uses two optimization techniques to reduce the modules of the prior version by selecting the most important and representative modules to construct the CVDP model. In addition, our method does not need extra efforts for labeling modules.

2.2. Training subset selection

Although no studies have proposed a customized training subset selection method for CVDP, there are some related studies for CPDP which focuses on selecting a module subset from a project (a.k.a. source project) to build a classification model for the modules of the project at hand(a.k.a. target project). The essential distinction between CVDP and CPDP is that data distribution across versions is more similar than across projects since the current version usually carries plenty of information from the prior version (Shukla et al., 2018).

Turhan et al. (2009) proposed a nearest neighbor filter method, called **T**urhan **F**ilter (**TF**), to select a module subset from the source project. More specifically, for each module in the target project, TF first selected its top-k' nearest modules, then these candidate modules without duplication constituted the training set. They found that CPDP with TF achieved similar classification performance compared with IVDP on some cross-project pairs.

Peters et al. (2013) proposed a subset selection strategy, called **P**eter **F**ilter (**PF**), with a clustering algorithm. More concretely, PF first combined the data of the two projects, then used k-means clustering algorithm to cluster the mixed data and discarded the clusters that did not contain any module of the target project. For each module of the source project in the remaining clusters, the method found its nearest neighbor module in the target project (called popular modules). Then, for each popular module, PF selected its greatest fan (the nearest modules in the source project). Finally, these selected fans were fed into the classification model. The results showed that PF led to the performance improvement compared with the TF method and IVDP performance.

Kawata et al. (2016) proposed a relevancy filter method, called **K**awata **F**ilter (**KF**), for source project data simplification. KF first used the DBSCAN algorithm to cluster the mixed project data. For the clusters that contained at least one module of the target project, the modules of the source project in such clusters formed the final training data.

Following Kawata et al.'s work, Yu et al. (2017) proposed a new subset selection method, called **Y**u **F**ilter (**YF**), by just replacing the DBSCAN clustering algorithm with agglomerative clustering. The experimental results showed that YF achieved a small improvement compared with KF.

All these methods were designed for CPDP task and took Euclidean distance or data density as filter criterion to select the modules from the source project as the training set. Different from these studies, we utilize a two-stage optimization strategy to select the vitally valuable modules as the training set for CVDP.

3. Method

3.1. The flow chart of our TSTSS framework for CVDP

Fig. 1 depicts the flow chart of our proposed two-stage training subset selection method TSTSS for CVDP task. In the first stage (i.e., the self-simplification stage), TSTSS utilizes a novel SMRS method to select a simplified module subset from the prior version whose elements are important to reconstruct the whole data of the prior version. Since the feature distributions of the non-defective and defective modules are different (otherwise no classifier can distinguish them), which means that the non-defective modules could not well represent the defective ones and vice versa, thus we apply SMRS separately to the two classes of modules. Then, we mix the two selected module subsets as the output of the first subset selection stage. This stage aims to select the important modules which can reconstruct the original module set with minimal error and remove the useless modules towards the original set. In the second stage (i.e., the auxiliary-refining stage), TSTSS employs a novel DS3 method to weed out the modules from the mixed module set which may not be able to well represent the modules of the current version. This stage aims to narrow the gap of the distribution differences of the data between the two versions by further purifying the modules selected in the first stage. After the two-stage selection process, we obtain an enhanced training module subset. As the defect data usually have imbalanced numbers of the two classes of modules, finally, we use a novel class imbalanced classifier WELM as our basic classification model.

3.2. The SMRS method

As the goal of the first stage of TSTSS is to simplify the data of prior version by only retaining part of valuable modules, we employ a self-representation learning method SMRS to select a module subset that minimizes the reconstruction errors without losing important information. To the best of our knowledge, there are no literatures related to the self-simplifications task in defect prediction studies.

SMRS is a modification version of dictionary learning framework aiming at finding a sparse representation of the original data. Considering traditional dictionary learning framework usually selects some important data points which do not coincide with the

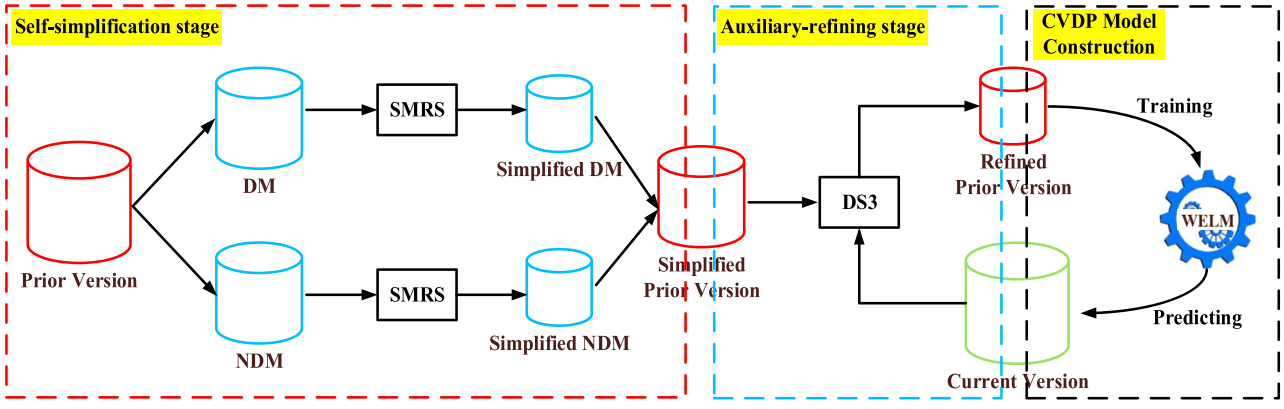


Fig. 1. Overview of our TSTSS framework.

actual ones in the original data, Vidal et al. (2012) improved it by replacing the dictionary with the matrix of the data points. This guarantees to select the important points from the actual data space. This section details the SMRS method as follows.

We denote the modules of the prior version as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M] \in \mathbb{R}^{r\times M}$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{ir}]^{\mathrm{T}} \in \mathbb{R}^r$, M and r indicate the number of modules and features of the prior version, respectively. SMRS aims to minimize the following objective function:

$$\sum_{i=1}^{M} \|\mathbf{x}_i - \mathbf{X}\mathbf{c}_i\|_2^2 = \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2. \tag{1}$$

where $\mathbf{C} = [c_1, c_2, \ldots, c_M] \in \mathbb{R}^{M\times M}$ represents the coefficient matrix, $\mathbf{c}_i = [c_{i1}, c_{i2}, \ldots, c_{iM}] \in \mathbb{R}^M$ denotes the coefficient vector of $\mathbf{x}_i$ over $\mathbf{X}$, and $\|\cdot\|_F$ denotes the Frobenius norm (or l_2 norm). The motivation is to select a module subset whose linear combination is used to restructure the original data of the prior version. For this purpose, we add a term $\|\mathbf{C}\|_{0,q} \le k_1$ to constrain the selected module number, where the $\|\cdot\|_{0,q}$ denotes the l_0/l_q norm and is defined as $\|\mathbf{C}\|_{0,q} = \sum_{i=1}^{M} \mathbb{I}(\|\mathbf{c}^i\|_q)$, c^i denotes the ith row of the coefficient matrix $\mathbf{C}$ and $\mathbb{I}(\cdot)$ denotes the indicator function. In addition, we add a term $\mathbf{1}^{\mathrm{T}}\mathbf{C} = \mathbf{1}^{\mathrm{T}}$ to constrain that the coefficient sum of each column in matrix $\mathbf{C}$ is equal to 1. (Elhamifar and Vidal, 2009). Thus, we have the following optimization formulation:

$$\begin{aligned} \min_{\mathbf{C}} \quad & \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \|\mathbf{C}\|_{0,q} \le k_1, \mathbf{1}^{\mathrm{T}}\mathbf{C} = \mathbf{1}^{\mathrm{T}}. \end{aligned} \tag{2}$$

However, Eq. (2) is a NP-hard problem as it needs to search all the subset of the k_1 columns of $\mathbf{X}$. To alleviate this problem, an alternative way is to use some strategies to appropriately relax the harsh constraints, such as using l_1 relaxation to replace the l_0/l_q norm with l_1/l_q norm and relaxing the positive integer k_1 to real number μ (Vidal et al., 2012). Then Eq. (2) is relaxed to the formulation as

$$\begin{aligned} \min_{\mathbf{C}} \quad & \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \|\mathbf{C}\|_{1,q} \le \mu, \mathbf{1}^{\mathrm{T}}\mathbf{C} = \mathbf{1}^{\mathrm{T}}. \end{aligned} \tag{3}$$

Using Lagrange multiplier method, Eq. (3) can be rewritten as:

$$\begin{aligned} \min \quad & \lambda_1\|\mathbf{C}\|_{1,q} + \tfrac{1}{2}\|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \mathbf{1}^{\mathrm{T}}\mathbf{C} = \mathbf{1}^{\mathrm{T}}, \end{aligned} \tag{4}$$

where λ_1 is the regularization parameter used to control the number of the selected modules. Eq. (4) can be solved under the **A**lternating **D**irection **M**ethod of **M**ultipliers (**ADMM**) optimization framework (Gabay and Mercier, 1976; Boyd et al., 2011).

Given a specific parameter λ_1, we will select a simplified but important module subset $\mathbf{S}$ with m modules as $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m] \in \mathbb{R}^{r\times m}$, where $\mathbf{s}_i = [s_{i1}, s_{i2}, \ldots, s_{ir}]^{\mathrm{T}} \in \mathbb{R}^r$.

3.3. The DS3 method

As the goal of the second stage of TSTSS is to reserve the modules of the prior version that are representative to the ones in the current version, we utilize a cross-domain subset selection method DS3 (Elhamifar et al., 2016) to pick up a more refined module subset from $\mathbf{S}$. This section details the DS3 method based on the pairwise dissimilarities between the modules of the two versions as follows.

Here, we denote the modules of the current version as $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n] \in \mathbb{R}^{q\times n}$, where $\mathbf{t}_j = [t_{j1}, t_{j2}, \ldots, t_{jq}]^{\mathrm{T}} \in \mathbb{R}^q$, n and q indicate the number of modules and features of the current version, respectively. Note that r is equal to q in our CVDP scenario. In addition, we define a dissimilarity matrix $\mathbf{D} = [d_{ij}]$ (i=1,...,m and j=1,...,n) between the modules of the two versions, where d_{ij} indicates the dissimilarity between the module $\mathbf{s}_i$ and $\mathbf{t}_j$ which indicates how well $\mathbf{s}_i$ represents $\mathbf{t}_j$. The purpose of DS3 is to find a representative module subset of $\mathbf{S}$ that can effectively represent each module of $\mathbf{T}$.

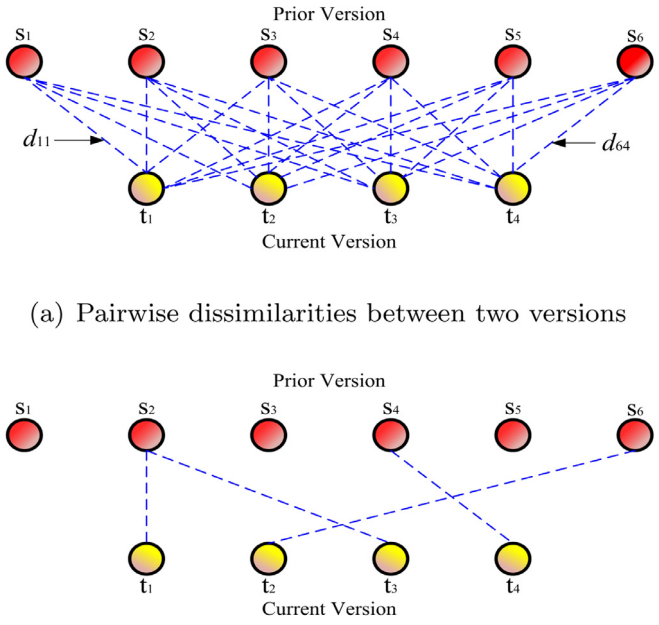


Fig. 2. Illustration of the function of DS3.

Dissimilarity matrix $\mathbf{D}$ is formulated as

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{d}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \in \mathbb{R}^{m\times n}, \tag{5}$$

where $\mathbf{d}_i \in \mathbb{R}^n$ is the ith row of $\mathbf{D}$, i.e., the dissimilarities between the ith module of $\mathbf{S}$ and each module of $\mathbf{T}$.

To indicate whether a module in $\mathbf{S}$ is a representative of the module in $\mathbf{T}$, we define a 0–1 indicator matrix $\mathbf{P}$ as

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{p}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix} \in \mathbb{R}^{m\times n}, \tag{6}$$

where $p_{ij} \in \{0, 1\}$ is the indicator of $\mathbf{s}_i$ representing $\mathbf{t}_j$. $p_{ij} = 1$ indicates that $\mathbf{s}_i$ is a representative of $\mathbf{t}_j$, otherwise not. To guarantee that each $\mathbf{t}_j$ is represented by one $\mathbf{s}_i$, we have the constrain as $\sum_{i=1}^{m} p_{ij} = 1$.

To have a better understanding of the above description, we provide a graphic depiction in Fig. 2. Each red circle denotes a module of the prior version while each yellow circle represents a module of the current version, and the dotted line signifies the correlation (i.e., dissimilarity in DS3) between two modules. Note that if there exists a dotted line between a red circle and a blue circle after conducting DS3 method, it means that the red circle can well represent the yellow circle. Fig. 2(a) shows the inputs of DS3 method, i.e., the pairwise dissimilarities between the modules of the two versions. Fig. 2(b) depicts the results after running DS3 method. It shows that DS3 method selects 3 modules (i.e., $\mathbf{s}_2$, $\mathbf{s}_4$ and $\mathbf{s}_6$) from the prior version as the representatives to represent each module of the current version. Note that there is only one dotted line connecting with each yellow circle since we assign each module of the current version with only one module of the prior

version. However, there can have multiple dotted lines connecting with each red circle since each module of the prior version can represent multiple modules of the current version, for example, $\mathbf{s}_2$ can represent $\mathbf{t}_1$ and $\mathbf{t}_3$.

To select a representative subset of $\mathbf{S}$ based on the dissimilarity matrix $\mathbf{D}$, DS3 simultaneously optimizes the following two terms: minimize the representing cost towards $\mathbf{T}$ by $\mathbf{S}$ and the number of selected modules from $\mathbf{S}$ as follows:

$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda_2 \sum_{i=1}^{m} \mathbb{I}(\|\mathbf{p}_i\|_q) \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in \{0,1\}, \forall i, j, \end{aligned} \tag{7}$$

where the first term refers to the representing cost towards $\mathbf{T}$ by $\mathbf{S}$, and the second term denotes the number of selected representatives from $\mathbf{S}$ which corresponds to the number of nonzero rows in matrix $\mathbf{P}$. More specifically, if $\mathbf{s}_i$ is selected as a representative of $\mathbf{t}_j$, then the representing cost towards $\mathbf{t}_j$ by $\mathbf{s}_i$ is calculated as $d_{ij}p_{ij} \in \{0, d_{ij}\}$, thus the representing cost towards $\mathbf{T}$ by $\mathbf{S}$ is $\sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij}$. In addition, if $\mathbf{s}_i$ is a representative towards some modules of $\mathbf{T}$, then not all elements in the ith row of $\mathbf{P}$ are zeros, i.e., $\sum_{j=1}^{n} p_{ij} \neq 0$. Lower control parameter λ_2 means more candidates will be selected.

However, Eq. (7) is a non-convex optimization problem. Elhamifar et al. (2016) suggested to replace $\sum_{i=1}^{m}\mathbb{I}(\|\mathbf{p}_i\|_q)$ with l_q norm $\|\mathbf{p}_i\|_q$ and relax the 0–1 constraint $p_{ij} \in \{0, 1\}$ to continuous interval constraint $p_{ij} \in [0, 1]$. Then Eq. (7) is relaxed to the following formulation:

$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda_2 \sum_{i=1}^{m} \|\mathbf{p}_i\|_q \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in [0,1], \forall i, j. \end{aligned} \tag{8}$$

By using trace function, Eq. (8) is rewritten as the following matrix form:

$$\begin{aligned} \min_{\mathbf{P}} \quad & \mathrm{tr}(\mathbf{D}^{\mathsf{T}}\mathbf{P}) + \lambda_2 \|\mathbf{P}\|_{1,q} \\ \text{s.t.} \quad & \mathbf{1}^{\mathsf{T}}\mathbf{P} = \mathbf{1}^{\mathsf{T}}, \mathbf{P} \in [0,1], \end{aligned} \tag{9}$$

where $\|\mathbf{P}\|_{1,q} = \sum_{i=1}^{m} \|\mathbf{p}_i\|_q$, $\mathbf{1}$ is a vector whose elements are all 1, and $\mathrm{tr}(\cdot)$ denotes the trace function used to calculate the sum of the diagonal elements in the matrix. Eq. (9) can also be solved by the ADMM framework. After obtaining the optimal solution $\widehat{\mathbf{P}}$, the line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ correspond to the indexes of the modules in $\mathbf{S}$ that are selected to represent the modules of $\mathbf{T}$. For example, for Fig. 2(b) that selected 3 representative modules of the prior version (i.e., $\mathbf{S}$) to represent the 4 modules of the current version (i.e., $\mathbf{T}$), the optimal solution $\widehat{\mathbf{P}}$ is like the following formula $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, where the values in the positions with 1 are nonzero. The line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ is 2, 4, 6, which indicates that the second (i.e., $\mathbf{s}_2$), the fourth (i.e., $\mathbf{s}_4$) and the sixth (i.e., $\mathbf{s}_6$) module of the prior version (i.e., $\mathbf{S}$) are selected as the representatives.

To have an intuitive feeling about the effect of DS3 for selecting the representative modules, we conduct a case study on two synthetic datasets to simulate CVDP scenario. We generate the non-defective modules of the prior version by drawing data points (red hexagrams) from a mixture of Gaussians with means (2, 3.5) and (4, 5.5) with 120 points in each set, and the defective modules by drawing 90 data points (green rhombuses) from a mixture of Gaussians with means (6.5, 2.5), as showed in Fig. 3(a). To reflect the distribution differences between two versions, we generate the non-defective modules of the current version by drawing data points (purple triangles) from a mixture of Gaussians with means (1.5, 3) and (4.5, 6) with 60 points in each set, and the defective modules by drawing 40 data points (blue pentagrams) from a mixture of Gaussians with means (6.5, 2.5), as showed in Fig. 3(b). Fig. 3(c), (d), and (e) depict the selected non-defective modules (black hexagrams) and defective modules (black rhombuses) from the prior version by DS3 with 3 different λ_2 values. From the last 3 subfigures, we can see that the selected modules are close to the positions of the modules in Fig. 3(b). In addition, we observe that as the λ_2 decreases, the number of selected modules by DS3 increases.

After the second subset selection process, we obtain a more refined modules subset with m'' modules from the prior version and denote it as $\mathbf{S}' = [\mathbf{s}'_1, \mathbf{s}'_2, \ldots, \mathbf{s}'_{m''}] \in \mathbb{R}^{r \times m''}$, where $\mathbf{s}'_i = [s'_{i1}, s'_{i2}, \ldots, s'_{ir}]^{\mathsf{T}} \in \mathbb{R}^r$. In addition, the corresponding label set is defined as $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{m''}] \in \mathbb{R}^{d' \times m''}$, where $\mathbf{y}_i = [y_{i1}, \ldots, y_{id'}]^{\mathsf{T}} \in \mathbb{R}^{d'}$. In this work, $d' = 2$ since we only have two classes of modules: non-defective and defective ones.

3.4. WELM classifier

As defect data are usually class imbalanced, we use a weighted classifier WELM as our base classification model to relieve this issue. Previous study (Xu et al., 2019) has shown that WELM performs better that the traditional classifiers on imbalanced dataset.

Before introducing the WELM, we first introduce the basic ELM classifier. ELM is a SLFNs proposed by Huang et al. (2004). Different from the traditional neural networks, its hidden-node parameters and input weights are randomly generated without artificially specifying. The main core of ELM is to calculate the output matrix which connects the hidden and output layers.

Given the selected dataset $\{\mathbf{s}'_i, \mathbf{y}_i\} \in \mathbb{R}^r \times \mathbb{R}^{d'} (i = 1, 2, \ldots, m'')$, the output of the typical neural network based on the forward transmission rule with q' hidden nodes and activation function $\boldsymbol{\phi}(\cdot)$ is formally expressed as

$$\sum_{j=1}^{q'} \boldsymbol{\beta}_j \boldsymbol{\phi}(\mathbf{s}'_i) = \sum_{j=1}^{q'} \boldsymbol{\beta}_i \phi(\mathbf{w}_j \cdot \mathbf{s}'_i + \mathbf{b}_j) = o_i, i = 1, \ldots, m'', \tag{10}$$

where $\mathbf{w}_j = [w_{i1}, w_{i2}, \ldots, w_{jr}] \in \mathbb{R}^r$ is the weight vector connecting the jth hidden node and all input nodes, $\mathbf{b}_j$ is the bias term of the jth hidden node, $\boldsymbol{\beta}_j = [\beta_{j1}, \beta_{j2}, \ldots, \beta_{jd'}] \in \mathbb{R}^{d'}$ is the weight vector connecting the jth hidden node and all output nodes, $\phi(\mathbf{w}_j \cdot \mathbf{s}'_i + \mathbf{b}_j)$ is the output of the hidden layer, and o_i denotes the expected output of the ith module. Fig. 4 depicts the architecture of ELM. Eq. (10) can be converted to the following linear system:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{O}, \tag{11}$$

where $\mathbf{H}$ denotes the output vector of the hidden layer as

$$\begin{aligned} \mathbf{H} &= \mathbf{H}(\mathbf{w}_1, \ldots, \mathbf{w}_{q'}, \mathbf{b}_1, \ldots, \mathbf{b}_{q'}, \mathbf{s}'_1, \ldots, \mathbf{s}'_{m''}) \\ &= [\boldsymbol{\phi}(\mathbf{s}'_1), \ldots, \boldsymbol{\phi}(\mathbf{s}'_{m''})]^{\mathsf{T}}. \end{aligned} \tag{12}$$

$\boldsymbol{\beta}$ denotes the weight matrix connecting the hidden and output layer as

$$\boldsymbol{\beta} = [\boldsymbol{\beta}_1^{\mathsf{T}}, \ldots, \boldsymbol{\beta}_{q'}^{\mathsf{T}}]^{\mathsf{T}}_{q' \times d'}. \tag{13}$$

$\mathbf{O}$ denotes the expected label matrix as

$$\mathbf{O} = [\mathbf{o}_1^{\mathsf{T}}, \ldots, \mathbf{o}_{m''}^{\mathsf{T}}]^{\mathsf{T}} \tag{14}$$

Training the SLFNs is to find a least-squares solution $\hat{\boldsymbol{\beta}}$ of $\mathbf{H}\boldsymbol{\beta} = \mathbf{O}$ as

$$\min_{\boldsymbol{\beta}} \|\mathbf{O} - \mathbf{Y}\|_2 = \min_{\boldsymbol{\beta}} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}\|_2. \tag{15}$$

The minimum norm least-square solution of Eq. (15) is

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{Y}, \tag{16}$$

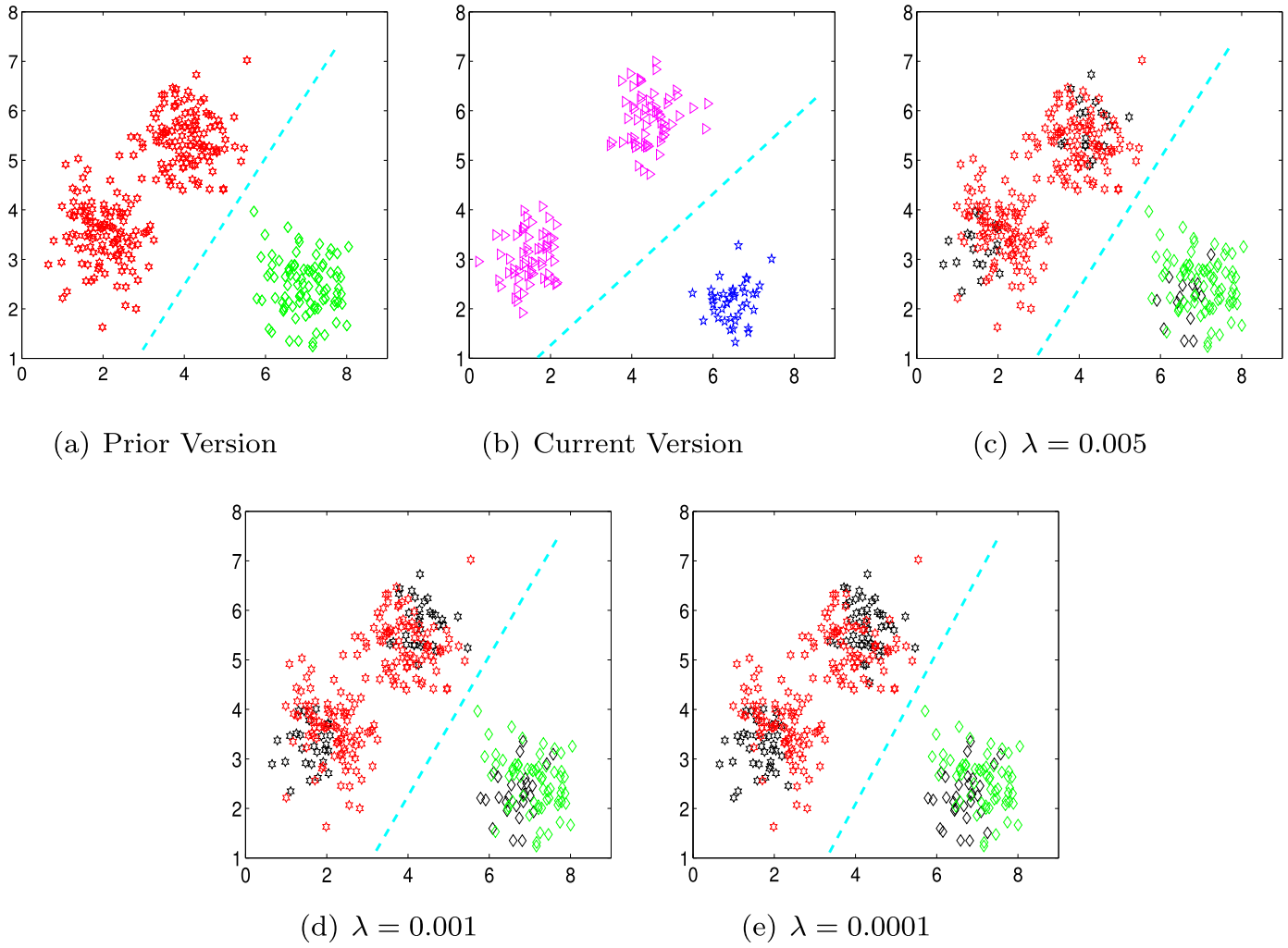


Fig. 3. An example for the modules selected by DS3 with different λ_2 on synthetic data.

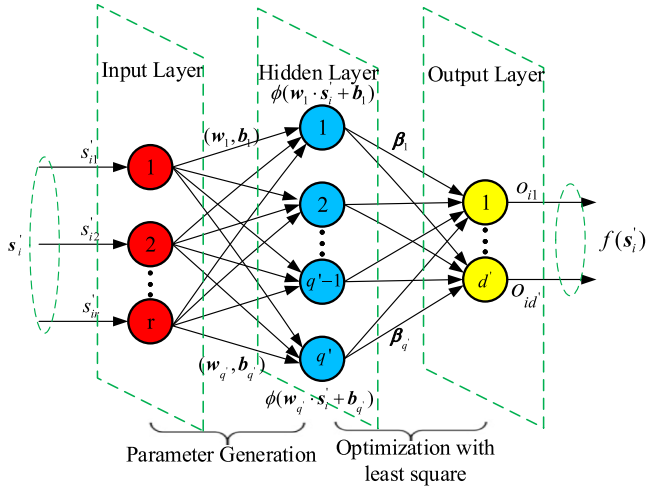


Fig. 4. The architecture of ELM.

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of $\mathbf{H}$ (Prasad and Bapat, 1992).

Finally, we get the classification function of ELM as

$$f(\mathbf{s}') = \boldsymbol{\phi}(\mathbf{s}')\hat{\boldsymbol{\beta}} = \boldsymbol{\phi}(\mathbf{s}')\mathbf{H}^{\dagger}\mathbf{Y}. \tag{17}$$

Like many general classifiers, ELM classifier works well when the training set has balanced module numbers of different classes. To deal with the imbalanced defect data, we employ a novel **W**eighted **ELM** (**WELM**) (Zong et al., 2013) to construct the CVDP model to alleviate the negative impact of the class imbalance on the prediction performance.

We define a $m'' \times m''$ diagonal matrix $\mathbf{W}$, whose diagonal element $\mathbf{W}_{ii}$ denotes the weight of training sample $\mathbf{s}'_i$. More precisely, if $\mathbf{s}'_i$ belongs to the class with larger module number, we assign it smaller $\mathbf{W}_{ii}$, otherwise, larger $\mathbf{W}_{ii}$. According to the KKT theorem (Fletcher, 2013), Eq. (16) is modified as

$$\hat{\boldsymbol{\beta}} = (\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{18}$$

Then, Eq. (17) is rewritten as

$$f(\mathbf{s}') = \boldsymbol{\phi}(\mathbf{s}')\hat{\boldsymbol{\beta}} = \boldsymbol{\phi}(\mathbf{s}')(\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{19}$$

In this work, we use the same weighting scheme in Xu et al. (2019) as follows:

$$\mathbf{W}2 = \mathbf{W}_{ii} = \begin{cases} 0.618/n_P & \text{if } s'_i \in \text{minority class} \\ 1/n_N & \text{if } s'_i \in \text{majority class} \end{cases}, \tag{20}$$

where n_P and n_N indicate the module numbers of the minority and majority class, respectively.

Table 1
Statistic of benchmark dataset.

Project	Version	# Modules	# Defective	% Defective	Project	Version	# Modules	# Defective	% Defective
ant	1.3	126	20	15.9%	velocity	1.4	196	147	75.0%
	1.4	178	40 (30.0%)	22.5%		1.5	214	142 (53.7%)	66.4%
	1.5	293	32 (17.5%)	10.9%		1.6.1	229	78 (40.1%)	34.1%
	1.6	352	92 (53.1%)	26.1%	xerces	init	162	77	47.5%
	1.7	745	166 (67.4)	22.3%		1.2	440	71 (41.6%)	16.1%
camel	1.0	339	13	3.8%		1.3	453	69 (23.9%)	15.2%
	1.2	608	216 (84.6%)	35.5%		1.4.4	588	437 (50.7%)	74.3%
	1.4	872	145 (44.9%)	16.6%	prop-1	9	4455	149	3.3%
	1.6	965	188 (56.6%)	19.5%		44	4081	376	9.2%
ivy	1.1	111	63	56.8%		92	3670	1287	35.1%
	1.4	241	16 (12.7%)	6.6%		128	3619	220	6.1%
	2.0	352	40 (0.0%)	11.4%		164	3541	319	9.0%
jedit	3.2.1	272	90	33.1%		192	3692	85	2.3%
	4.0	306	75 (52.2%)	24.5%	prop-2	225	1864	147	7.9%
	4.1	312	79 (65.3%)	25.3%		236	2403	76	3.2%
	4.2	367	48 (36.7%)	13.1%		245	2023	103	5.1%
	4.3	492	11 (6.3%)	2.2%		256	2025	625	30.9%
log4j	1.0	135	34	25.2%		265	2372	229	9.7%
	1.1	109	37 (64.7%)	33.9%	prop-3	285	1709	177	10.4%
	1.2	205	189 (89.2%)	92.2%		292	2330	209	9.0%
lucene	2.0	195	91	46.7%		305	2388	89	3.7%
	2.2	247	144 (71.4%)	58.3%		318	2440	365	15.0%
	2.4	340	203 (72.2%)	59.7%	prop-4	347	2906	162	5.6%
poi	1.5	237	141	59.5%		355	2802	924	33.0%
	2.0	314	37 (13.5%)	11.8%		362	2865	213	7.4%
	2.5.1	385	248 (73.0%)	64.4%	prop-5	4	3514	264	7.5%
	3.0	442	281 (77.8%)	63.6%		40	3815	466	12.2%
synapse	1.0	157	16	10.2%		85	3509	930	26.5%
	1.1	222	60 (56.3%)	27.0%		121	3445	425	12.3%
	1.2	256	86 (50.0%)	33.6%		157	2863	367	12.8%
xalan	2.4	723	110	15.2%		185	3260	268	8.2%
	2.5	803	387 (64.5%)	48.2%	prop-42	452	317	33	10.4%
	2.6	885	411 (57.1%)	46.4%		453	259	20	7.7%
	2.7	909	898 (96.8%)	98.8%		454	295	13	4.4%

4. Experimental setup

4.1. Benchmark dataset

In this work, we use 67 versions of 17 software projects provided by Madeyski and Jureczko (2015) as our benchmark dataset. Each software module denotes a class file of the Java project and is characterized by 20 module features with a defect label. The 20 features are calculate by a tool called CKJM and the label is identified by a tool called BugInfo. The original defect label is the defect number of the module. In this work, we transform it into a binary label by annotating the modules without defect as 0, otherwise as 1.

Detailed statistic description of each version for all 17 projects is reported in Table 1, where # Modules, # Defective, % Defective denote the number of modules, the number of defective modules and the defect ratio, respectively. Note that the number in the bracket of the column with # Defective measures the ratio of the common defective module number across the versions to the defective module number of the prior version. Note that since the data of project Prop1, Prop2, Prop3, Prop4, Prop5, Prop42 do not contain the module name, we could not count this measure for the 6 projects. As a result, we count the percentages on total 30 cross-version pairs. From the table, we observe that the percentages on 22 out of 30 pairs are higher than 40%, which means that in most cross-version pairs, at least 40% defective modules in the prior version still remain in the next version. This observation indicates the similar uneven distribution of defects across versions.

Note that our benchmark dataset contains 11 open-source projects (the first 11 projects), 5 industrial projects belonging to the insurance domain (prop1-prop5), and 1 industrial project (prop42) which is a support tool for quality assurance in software development. The benchmark dataset and the feature descriptions are available in our online supplementary materials.[1]

4.2. Evaluation indicators

In this section, we describe the 6 evaluation indicators used to measure the CVDP performance. In terms of the traditional evaluation indicators, we choose F-measure, MCC, and AUC which are widely used in previous defect prediction studies (Wang et al., 2016b; Nam et al., 2013; Jing et al., 2015; Xu et al., 2016b; Rahman et al., 2012; Li et al., 2012; Menzies et al., 2007; He et al., 2012; Jing et al., 2014; Song et al., 2011; Wang and Yao, 2013; Xu et al., 2018b). The 3 indicators are derived from the basic indicators listed in Table 2 and described as follows

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

$$\text{F-measure} = \frac{(1+\theta^2) * \text{recall} * \text{precision}}{\theta^2 * \text{precision} + \text{recall}}. \tag{21}$$

MCC measures the correlation coefficient between the actual and predicted outputs which is defined as

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}. \tag{22}$$

AUC calculates the area under and ROC curve which is a two-dimensional plane with TPR as y-axis and FPR as the x-axis.

The θ in Eq. (21) is a bias parameter to measure the relative importance of recall and precision. There are 3 widely-used **F**-measure (**F**) variants, i.e., F_1 ($\theta = 1$) which treats precision and recall equally, $F_{0.5}$ ($\theta = 0.5$) which prefers precision, and F_2 ($\theta = 2$)

[1] https://github.com/sailerzhou/TSTSS.

Table 2
Basic indicators for defect prediction.

	# Predicted defective	# Predicted non-defective
# Actual defective	**T**rue **P**ositive (**TP**)	**F**alse **N**egative (**FN**)
# Actual non-defective	**F**alse **P**ositive (**FP**)	**T**rue **N**egative (**TN**)
True **P**ositive **R**ate (**TPR**) or recall		$\frac{TP}{TP+FN}$
False **P**ositive **R**ate (**FPR**)		$\frac{FP}{FP+TN}$
precision		$\frac{TP}{TP+FP}$

which prefers recall. In defect prediction task, the defective modules are the main concerns and we always want to accurately detect as many defective modules as possible (Watanabe et al., 2008). Thus, the performance measurement should be evaluated bias to this purpose which is consistent with the definition of recall. Thus, in this work, we emphasize more on the importance of recall and choose F_2 variant following the previous studies (Jiang et al., 2008; Jing et al., 2017).

The 3 traditional binary classification indicators do not consider the quality assurance efforts required to review the modules (Zimmermann et al., 2009; Turhan et al., 2009; Chen et al., 2015). However, inspecting all the modules is not always practical due to the limited test resources. As suggested by Mende and Koschke (2010), it is more realistic to use effort-aware indicators to evaluate the performance of defect prediction.

Effort-aware indicators evaluate the defect prediction performance within a limited effort required to review the predicted defective modules (Arisholm et al., 2010; Kamei et al., 2013; Yang et al., 2016b). In reality, we always want to maximize the profit of any effort for quality assurance. Generally, the efforts denote the LOC that need to be inspected, and the profit is the number or percentage of defective modules discovered. In this work, we set the efforts as 20% of total LOC following previous studies (Jiang et al., 2013; Yang et al., 2015a; Xia et al., 2016).

To calculate the effort-aware indicators, the general way is to rank the modules according to a specific rule first, then simulate an expert to inspect these modules one at a time in order. In the meanwhile, the percentage of LOC reviewed and the number of detected defective modules are counted. The process is terminated when 20% of total LOC have been inspected. The proportion of detected defective modules among all the actual defective modules is treated as an effort-aware indicator, which is called **PofB**20 (**P**ercentage **of B**ugs) or **CE**20 (**C**ost **E**ffectiveness) in Jiang et al. (2013), Xia et al. (2016), Yang et al. (2015b) and Yang et al. (2016a).

In previous studies, researchers ranked the modules in a descending order based on the degree of their predicted risk values. The risk values are defined as the probability outputs of a classification model for the modules (Mende and Koschke, 2010; Jiang et al., 2013; Xia et al., 2016) or the ratios of the probability outputs to the corresponding LOC (Yang et al., 2015a, 2015b, 2016a). Recently, Huang et al. (Huang et al., 2017) proposed a novel ranking method to calculate the effort-aware indicators for defective change prediction, called **C**lassifier **B**efore **S**orting (**CBS**). Their experimental results showed that the derived indicator values significantly were improved based on their ranking method. The basic idea of CBS is that among the changes that are predicted to be potentially defective by a classifier, small changes that are measured by the modified LOC should be inspected first, since they give the best bang for the buck (Huang et al., 2017). However, this ranking method only ranks the predicted defective changes. It may underestimate the performance since the predicted non-defective changes can also contain actual defective changes. To remedy this limitation, in this work, we propose an improved ranking method based on CBS to rank the modules in our CVDP scenario. Fig. 5 show the calculation process of the effect-aware indicators based on our module ranking method. More specifically, ① we divide the modules into two parts (i.e., the predicted defective and non-defective modules) according to their predicted labels by WELM classifier; ② we rank the predicted defective modules in an ascending order based on their LOC values, this process is identical to CBS. In addition, we also rank the predicted non-defective module with the same process; ③ we concatenate the latter ranking results behind the former ranking results; ④ we obtain the checked modules (i.e., the top 9 modules in Fig. 5) by inspecting 20% of total LOC; ⑤ we count some statistical values to calculate the effort aware indicators.

We concisely describe how to calculate the 3 effort-aware indicators. Given a current version of a project with n_1 defective modules. After inspecting 20% of total LOC based on our improved ranking strategy, n' modules and $n_1{}'$ defective modules have been inspected.

The first effort-aware indicator is defined as the proportion of the inspected defective modules among all actual defective modules, which is called Recall by Huang et al. (2017). To distinguish it from the traditional Recall indicator, we name it **E**ffort-**A**ware **R**ecall (**EAR**). EAR is defined as

$$\mathrm{EAR} = \frac{n_1{}'}{n_1}. \tag{23}$$

The second effort-aware indicator is defined as the proportion of the inspected defective modules among all inspected modules, called **E**ffore-**A**ware **P**recision (**EAP**). EAP is defined as

$$\mathrm{EAR} = \frac{n_1{}'}{n'}. \tag{24}$$

Given the definitions of EAR and EAP, like traditional F-measure indicator, the third effort-aware indicator **E**ffort-**A**ware **F**-measure (**EAF**) is defined as

$$\mathrm{EAF} = \frac{(1+\theta^3) * \mathrm{EAR} * \mathrm{EAP}}{\theta^3 * \mathrm{EAP} + \mathrm{EAR}}. \tag{25}$$

In this work, we also set $\theta^3 = 2$ like the setting for the tradition F-measure. The worst case is that EAR (or recall) and EAP (or precision) are all equal to 0. Thus, the value of EAF (or F-measure) makes no sense since the denominator in Eq. (25) (or Eq. (21)) is 0. In this case, we set the value of EAF (or F-measure) as 0 which indicates the worst CVDP performance.

4.3. Parameter configuration

Regarding the implementation of the second stage of TSTSS, i.e., the DS3 method, we measure the dissimilarity with Chi-square distance following the original work (Elhamifar et al., 2016). The Chi-square distance of module s_i and t_j is defined as $\sum_{o=1}^{r} \frac{(s_{io}-t_{jo})^2}{2*(s_{io}+t_{jo})}$, where r denotes the feature dimension. In terms of the λ_2 in Eq. (9) which is used to control the final number of selected modules, a lower λ_2 value means more representative modules will be chosen. In this work, we determine the λ_2 value based on a threshold which denotes the desired proportion of the remaining

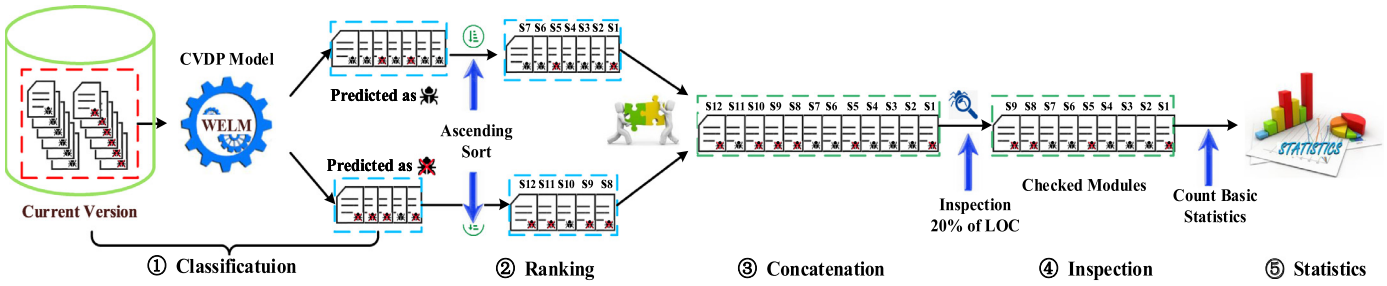


Fig. 5. The process to calculate the effort aware indicators.

modules. More specifically, we set the initial λ_2 value as 0.05 and gradually reduce it until the proportion of the selected modules towards the reversed modules in the first stage is larger than the threshold for the first time. We set 9 thresholds, i.e., 10%, 20%,..., 90%, to determine the λ_2 value. Note that we do not set threshold to 100% as it means that only the first stage SMRS is used. According to the above description, the final proportion of the selected modules may be a little higher than the corresponding threshold.

Regarding the implementation of WELM, only one parameter, i.e., the number of hidden nodes q, needs to be assigned. Since there are no theoretical guidances to determine the optimal q for all situations and prior study (Huang et al., 2006) stated that the q' value with far less than the number of training module (i.e., m'' in this work) can usually achieve the best performance, in this work, we set q' from 5 to $\frac{m''}{2}$ with an increment of 5 to search the optimal q' value for each cross-version pair.

4.4. Cross version scenario design

In this work, we conduct the CVDP experiment between two nearest versions. The reason of this setting is that the two closest versions share more identical architectural and design characteristics, which results in a certain degree of similarity (Shukla et al., 2018). As a result, we have total 50 cross-version pairs.

4.5. Statistic test method

The reason of using statistic test is that, if the average performance values of multiple methods are different, we still cannot claim that the method with higher average performance is superior to that with lower average performance in the statistical sense as this performance differences may be explained by randomness (Herbold et al., 2017). In this case, statistic test is used to identify whether the differences in performance between various methods are randomness or statistically significant. In this work, we perform the non-parametric Frideman test with the Nemenyi post-hoc test (Demšar, 2006) at significant level 0.05 over all cross-version pairs. This test holds no assumption on the distribution of the performance values and is less susceptible to outliers (Lessmann et al., 2008; Mende and Koschke, 2010; Herbold et al., 2017). The Friedman test evaluates whether the average rankings of different methods exist statistically significant differences. The test statistic of the Friedman test can be calculated as follows:

$$\tau_{\chi^2} = \frac{12Q}{L(L+1)}\left(\sum_{j=1}^{L} AR_j^2 - \frac{L(L+1)^2}{4}\right), \tag{26}$$

where Q denotes the total number of cross-version pairs, L denotes the number of methods needed to be compared, $AR_j = \frac{1}{Q}\sum_{i=1}^{Q} R_i^j$ denotes the average ranking of method j on all cross-version pairs and R_i^j denotes the ranking of jth method on the ith cross-version pair. τ_{χ^2} obeys the χ^2 distribution with $L-1$ degree of freedom (Zar et al., 1999). Since the original Friedman test statistic is too conservative, its variant τ_F is usually used to conduct the statistic test. τ_F is calculated as the following formula:

$$\tau_F = \frac{(Q-1)\tau_{\chi^2}}{Q(L-1)-\tau_{\chi^2}}. \tag{27}$$

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

If the null hypothesis is rejected, it means that the performance differences among different methods are nonrandom, then a so-called Nemenyi post-hoc test is performed to check which specific method differs significantly (Lessmann et al., 2008). For each pair of methods, this test uses the average ranking of each method and checks whether the ranking difference exceeds a **C**ritical **D**ifference (**CD**) which is calculated with the following formula:

$$CD = q_{\alpha,L}\sqrt{\frac{L(L+1)}{6Q}}, \tag{28}$$

where $q_{\alpha,L}$ is a critical value that related to the method number L and the significance level α. The critical values are available online[3]. In our experiments, $L = 50$, $\alpha = 0.05$, thus $q_{\alpha,L} = 3.99$. The Frideman test with the Nemenyi post-hoc test is widely used in previous studies (Lessmann et al., 2008; Mende and Koschke, 2010; Jiang et al., 2008; DAmbros et al., 2012; Nam and Kim, 2015; Li et al., 2017; 2018b).

However, the main drawback for post-hoc Nemenyi test is that it may generate overlapping groups for the methods that are compared, which means that a method may belong to multiple significantly different groups (Ghotra et al., 2015; Herbold et al., 2017). In this work, we utilize the strategy in Herbold et al. (2017) to address this issue. Fig. 6 depicts an example of the division rules as follows.

The post-hoc Nemenyi test divides the methods to a same group without significant differences if the discrepancy of their ranks is lower that CD value. Thus, in Fig. 6(a), as the discrepancy of the best rank (1.2 for Metho 1) and the worse rank (2 for Method 4) among the method set is less than CD value, all methods belong to one group (linked with red line).

In Fig. 6(b), as the discrepancy of the best rank (1.2 for Method 1) and the worse rank (3 for Method 4) among the method set is more than CD value but less than 2 CD values, these methods belong to 2 groups. More specifically, the method (Method 2) belongs to the top group because its rank (0.4) is closer to the best rank. Similarly, Method 3 belongs to the bottom group because its rank (2.5) is closer to the worst rank.

In Fig. 6(c), as the discrepancy of the best rank (1.2 for Method 1) and the worse rank (3.8 for Method 4) among the method set

[2] http://www.socr.ucla.edu/applets.dir/f_table.html.
[3] http://www.cin.ufpe.br/~fatc/AM/Nemenyi_critval.pdf.

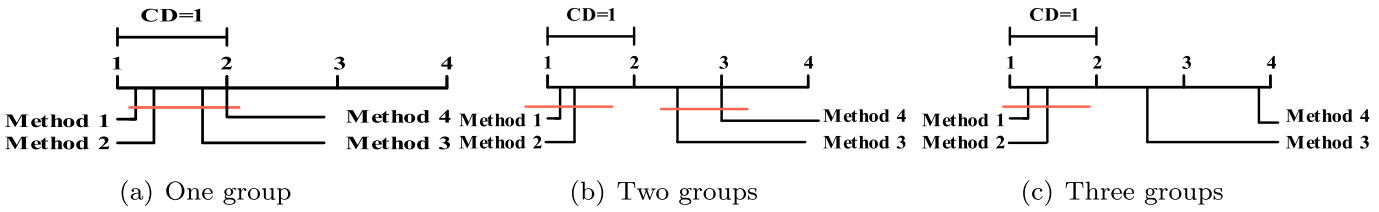


Fig. 6. An example of group division for Nemenyi test.

is more than 2 CD values, these methods belong to 3 groups. More specifically, Method 2 belongs to the top group because its rank discrepancy to the best rank is less than CD value. Only Method 4 belongs to the bottom group because there exist no methods whose rank discrepancy to the worst rank is less than CD value. Other methods (i.e., Method 3) whose rank discrepancies to the best rank and worst rank are all larger than CD value belong to the middle group.

Using the rules, the generating groups are non-overlapping with significantly different.

4.6. Research questions

We empirically evaluate our devised method TSTSS by answering the following 4 **R**esearch **Q**uestion**s** (**RQs**).

RQ1: How different classifiers impact the effectiveness of TSTSS on CVDP performance?

The goal of TSTSS is to select a module subset from the prior version which is used to train a defect prediction model. This question investigates whether the selected subset with WELM classifier described in Section 3.4 perform better than that with other classifiers for CVDP performance.

RQ2: Does our two-stage selection strategy select more effective module subset for CVDP compared with only one-stage selection strategy?

As TSTSS contains two subset selection stages: the self-simplification process with SMRS method and the auxiliary-refining process with DS3 method. This question is designed to study whether the module subset selected by TSTSS is more effective for CVDP than that selected by its 2 downgraded methods, i.e., with only one selection stage of TSTSS.

RQ3: Can TSTSS achieve better CVDP performance than other training subset selection methods?

Various subset selection methods select the module subsets based on different mechanisms. This experiment compares the effectiveness of TSTSS with other training subset selection methods.

RQ4: What are the differences of the modules selected by the training subset selection methods in RQ3?

Distinct training subset selection methods differ in the selected modules which lead to different CVDP performance. This question explores the differences among the module subsets selected by the methods in RQ3.

Our experiments are conducted on a computer with 16 × Inter(R)Core(TM), 3.60 GHz Intel i7-7820X CPU and 32GB Memory.

5. Experimental results

5.1. Results for RQ1

Before studying this question, we design an experiment to determine how many modules are reserved in the first subset selection stage of our TSTSS method. Since keeping too many modules may not attain the goal of modules reduction while keeping too fewer modules may lead to excessive loss of important information and increasing the reconstruction error, we empirically set 4 thresholds with our basic classifier WELM for comparison, including 80%, 70% 60%, 50%. As the parameter λ_1 in Eq. (4) is used to control the number of selected modules, we initial λ_1 as 5 and increase it with 5 at each time until the percentage of the selected modules first no less than the thresholds.

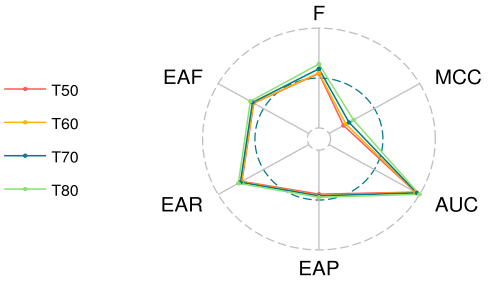


Fig. 7. Radar charts of average indicator values for TSTSS under different thresholds.

As mentioned in Section 4.3, we set 9 percentages (from 10% to 90%) for DS3 method to control the proportion of selected modules in the second stage subset selection process. Thus, we obtain total 9 sets of results for each indicator on each cross-version pair. In this work, as we emphasize the importance and practicability of the effort-aware indicators on CVDP performance especially for EAF, we only record the results corresponding to the best EAF value among the 9 sets of results for each cross-version pair.

Figs. 7 and 8 show the average performance values across all cross-version pairs for TSTSS under the 4 thresholds and the statistic test results, respectively. The detailed results are available in our online supplementary materials.

From Fig. 7, we observe that TSTSS with threshold 80% always achieves the best average performance among all indicators. From Fig. 8, we see that TSTSS with threshold 80% always obtains the best rank on all indicators. It is significantly superior to other 3 thresholds in terms of F-measure, MCC, and EAF but has no significant differences compared with EAP and EAR. Overall, 80% is the optimum threshold observed for the selected subset scale. We use this threshold for the following experiments.

To investigate the effect of different classifiers on the CVDP performance with our method TSTSS, we compare the used WELM classifier with 4 traditional base classifiers including ***k*-N**earest-**N**eighbor (***k*NN**), **L**ogistic **R**egression (**LR**), **C**lassification **A**nd **R**egression **T**ree (**CART**), and **R**andom **F**orest (**RF**), which are commonly used in defect prediction studies (Ghotra et al., 2015). Since using the default parameters for the classifiers may make the performance on these classifiers be underestimated, we follow the guidelines in Tantithamthavorn et al. (2018) to conduct a fine parameter tuning for them. More specifically, the number of non-overlapping clusters to produce in *k*NN is set to five options, i.e., {1, 5, 9, 13, 17}, and the number of classification trees in RF is set to five options, i.e., {10, 20, 30, 40, 50}.

Figs. 9 and 10 show the average performance values across all cross-version pairs for TSTSS with the 5 classifiers and the statistic test results, respectively.

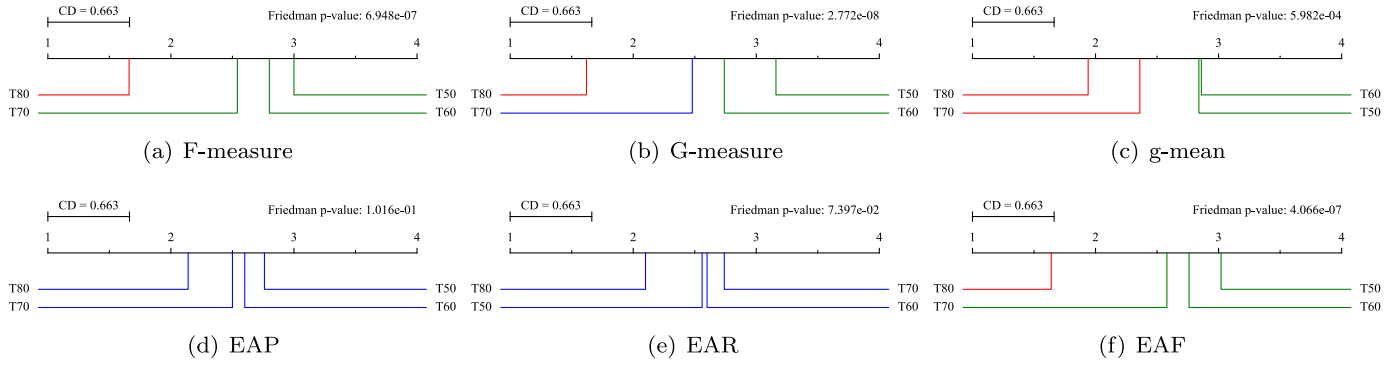


Fig. 8. Comparison of TSTSS under different thresholds using Friedman test with Nemenyi post-hoc test in terms of 6 indicators. Different colors represent different groups.

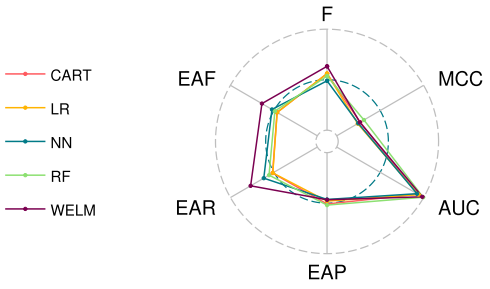


Fig. 9. Radar charts of average indicator values for TSTSS with different classifiers.

From Fig. 9, we observe that WELM achieves the best average F-measure, EAR, and EAF, whereas RF achieves the best average MCC, and EAP. From Fig. 10, we see that WELM belongs to the top group in terms of F-measure, AUC, EAR and EAF, whereas RF belongs to the top group in terms of MCC, AUC, and EAP. Overall, WELM and RF are the appropriate classifiers for CVDP with our proposed method TSTSS. As we emphasis more on the importance of the effort-aware indicators in this work, thus we choose WELM as our basic classifier for the following experiments.

Answer to RQ1: Overall, our CVDP method TSTSS with WELM classifier under threshold 80% can achieve satisfactory performance, especially in terms of effort-aware indicators.

5.2. Results for RQ2

To answer this question, we choose the subset selection scheme that only uses SMRS method and the scheme that only uses DS3 method as the downgraded methods for comparison. The parameter settings of the two baseline methods are the same as TSTSS for a fair comparison.

Fig. 11 shows the radar charts of the average values of each performance indicator on the cross-version pairs for each project as well as across all projects in terms of TSTSS and the 2 baseline methods. In the radar chart, each axis denotes a performance indicator and the value of the circle point (a.k.a vertex) on the axis corresponds to the indicator value. The vertex far away from the center has greater value. All the vertices are connected into a polygon. Each method corresponds to a polygon with a specific color. For the cross-version pairs on each project, we draw a radar chart. The values of the points on the charts are the average indicator values of these cross-version pairs. Since we have 17 different software projects, we draw 17 radar chart from Fig. 11(a)–(q). In addition, for all the cross-version pairs, we also draw a radar chart, i.e., Fig. 11(r). So we have total 18 radar charts in Fig. 11.

For the ease of replicating our experimental results, we list the threshold and the corresponding λ_2 value corresponding to the best EAF in Table 3. From Fig. 11 and Table 3, we have the following observations:

First, in terms of the 3 traditional indicators (i.e., F-measure (F), MCC, and AUC), from Fig. 11(r), we observe that TSTSS achieves the best average values on the 3 indicators across the 50 cross-version pairs compared with the 2 downgraded baseline methods. More specifically, TSTSS improves the SMRS and DS3 by 39.8% and 50.4% in terms of average F-measure respectively, by 49.2% and 93.4% in terms of average MCC respectively, and by 0.9% and 3.3% in terms of average AUC respectively.

Second, in terms of the 3 traditional indicators, from Fig. 11(a), (c), (d), (f), (h), (k), (m), (n), and (q), the results show that all 3 average indicator values of TSTSS are higher than the 2 baseline methods on project ant, ivy, jedit, lucene, synapse, xerces, prop2, prop3, prop42, respectively. In addition, Fig. 11(b), (g), (l), and (o) show that TSTSS achieves the best average F-measure and MCC than the two baseline methods on project camel, poi, prop1, and

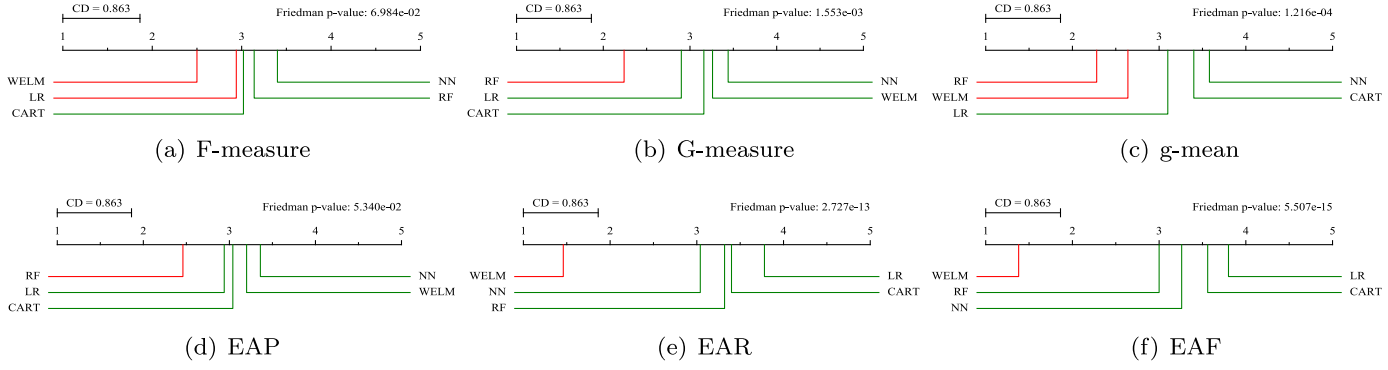


Fig. 10. Comparison of TSTSS with different classifiers using Friedman test with Nemenyi post-hoc test in terms of 6 indicators.

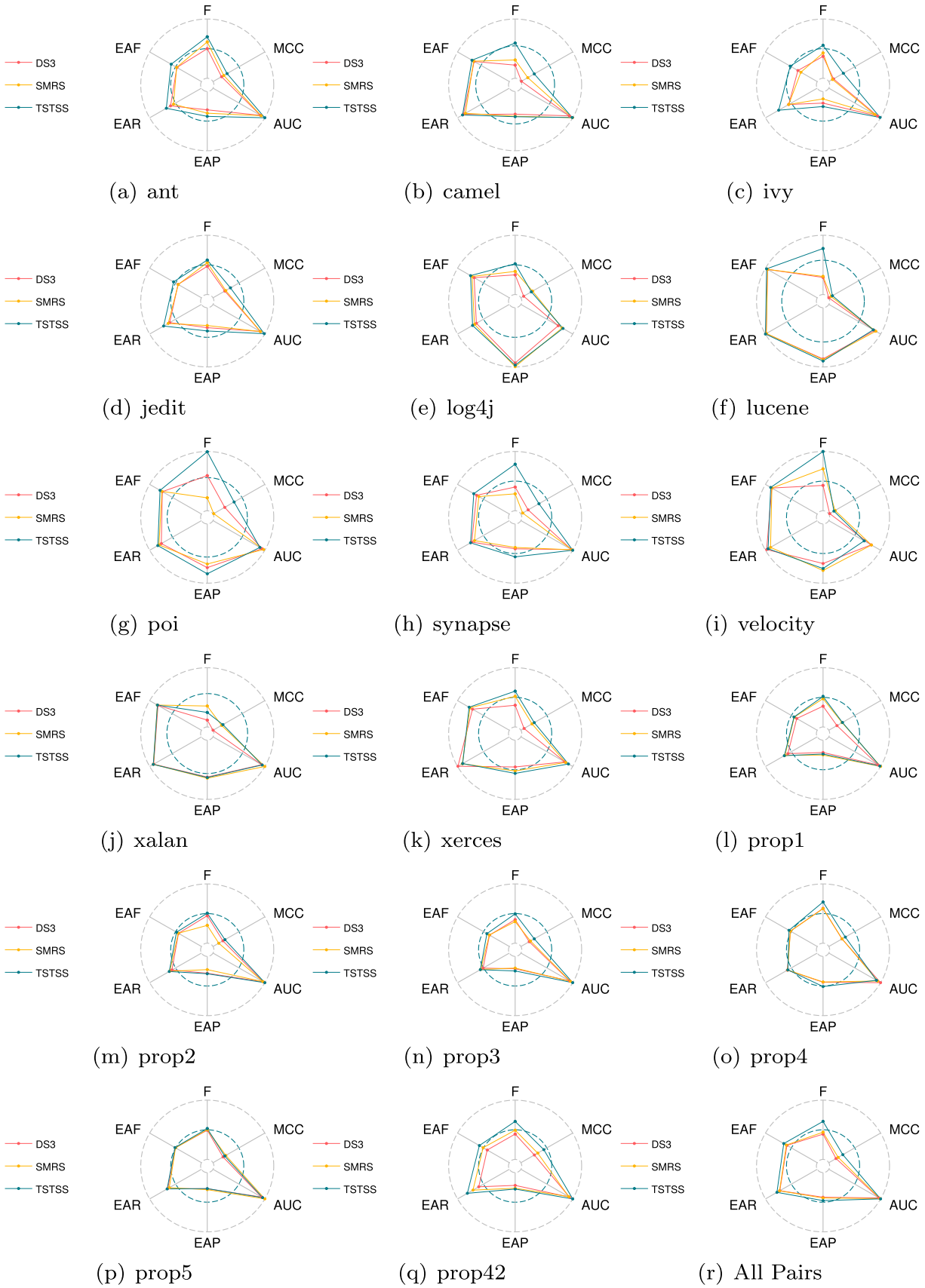


Fig. 11. Radar charts of average values of the 6 indicators on the cross-version pairs of each project and across all projects in terms of TSTSS, SMRS and DS3.

Table 3
The threshold of the selected module numbers and the corresponding λ_2 value.

Cross version pairs			Threshold	λ_2	Cross version pairs			Threshold	λ_2
ant	1.3	1.4	90%	2.0E−04	velocity	1.4	1.5	60%	5.0E−04
	1.4	1.5	80%	1.0E−03		1.5	1.6	80%	3.0E−04
	1.5	1.6	40%	4.0E−03	xalan	2.4	2.5	90%	4.0E−04
	1.6	1.7	50%	3.8E−03		2.5	2.6	50%	2.4E−03
camel	1.0	1.2	50%	2.9E−03	prop1	9	44	20%	2.0E−05
	1.2	1.4	50%	1.4E−03		44	92	30%	5.0E−06
	1.4	1.6	60%	6.0E−04		92	128	30%	2.0E−05
ivy	1.1	1.4	20%	1.0E−02		128	164	20%	7.0E−05
	1.4	2.0	90%	1.0E−05		164	192	30%	1.0E−05
jedit	3.2.1	4.0	80%	3.0E−04	prop2	225	236	90%	1.0E−05
	4.0	4.1	90%	1.0E−04		236	245	60%	1.0E−04
	4.1	4.2	90%	4.0E−05		245	256	80%	1.0E−04
	4.2	4.3	80%	2.0E−04		256	265	70%	2.0E−04
log4j	1.0	1.1	50%	4.0E−03	prop3	285	292	80%	4.0E−05
	1.1	1.2	30%	1.4E−02		292	305	80%	7.0E−06
lucene	2.0	2.2	30%	1.0E−02		305	318	20%	3.6E−03
	2.2	2.4	40%	3.5E−03	prop4	347	355	70%	2.0E−05
poi	1.5	2.0	70%	4.0E−04		355	362	40%	3.0E−04
	2.0	2.5.1	90%	2.0E−04	prop5	4	40	60%	3.0E−04
	2.5.1	3.0	20%	1.4E−02		40	85	20%	2.7E−03
synapse	1.0	1.1	40%	0.006		85	121	70%	3.0E−04
	1.1	1.2	40%	4.0E−03		121	157	60%	3.0E−04
xerces	init	1.2	70%	1.3E−03		157	185	50%	7.0E−04
	1.2	1.3	50%	1.7E−03	prop42	452	453	70%	6.0E−04
	1.3	1.4.4	40%	3.4E−03		453	454	80%	1.7E−03

prop4, respectively; Fig. 11(e) shows that TSTSS achieves the best average *F*-measure and AUC than the two baseline methods on project log4j.

Third, in terms of the 3 effort-aware indicators, from Fig. 11(r), we observe that TSTSS achieves the best average values on the 3 indicators across the 50 cross-version pairs compared with the 2 baseline methods. More specifically, TSTSS improves the SMRS and DS3 by 10.4% and 13.7% in terms of average EAP respectively, by 7.0% and 7.6% in terms of average EAR respectively, and by 8.2% and 10.5% in terms of average EAF respectively.

Fourth, in terms of the 3 effort-aware indicators, from Fig. 11(a)–(d), (f)–(h), (m), (n), and (q), the results show that all the 3 average indicator values of our method TSTSS are higher than the 2 baseline methods on project ant, camel, ivy, jedit, lucene, poi, synapse, prop2, prop3, and prop42, respectively. In addition, Fig. 11(e), (l), and (p) show that TSTSS achieves the better average EAR and EAF than the two baseline methods on project log4j, prop1, and prop5, respectively. Fig. 11(k) and (o) show that TSTSS achieves the better average EAP and EAF than the two baseline methods on project xerces and prop4, respectively.

Fifth, in terms of the threshold in Table 3 which indicates the approximate percentage of the selected modules in the second stage of TSTSS, we observe that TSTSS obtains the best EAF values with less than half of original module number (corresponding to the threshold no more than 60% since we have already selected 80% of the original modules in the first stage) on 29 out of 50 cross-version pairs. It indicates that TSTSS can select only a small proportion of modules from the prior version to achieve encouraging CVDP performance on more than half of all cross-version pairs.

Sixth, Fig. 12 visualizes the results of Friedman test and Nemenyi post-hoc test for TSTSS and the 2 baseline methods in terms of the 6 indicators. The *p* values (all less than 0.05) of Friedman test above the sub-figures show that there exist significant differences among the 3 methods on all indicators. The Nemenyi test results show that TSTSS significantly performs better than the 2 variant methods on all indicators.

Discussion: The takeaway lesson of the fifth finding is that when conducting CVDP, it is not necessary to use all the modules of the prior version to train the defect prediction model for the current version, since some modules have no discriminant ability or even negative impact on the classification performance. The reason why our method TSTSS outperforms SMRS is that, since SRMS selects candidates from the prior version to self-represent the original data without involving in the module information of the current version, it cannot guarantee that the candidates well represent the current version data. While TSTSS improves SMRS by combining a novel DS3 method to further refine the candidates with the participation of the current version data.

Since using a module subset to conduct CVDP can save a large amount of memory and computing overheads compared with using the whole module set, the advantage of TSTSS will become more apparent when the scale of the defect data increases.

Answer to RQ2: To summary, TSTSS is superior to its 2 downgraded methods. This means that the module subset selected by the two-stage selection strategy is more effective than that selected by only one-stage selection strategy.

5.3. Results for RQ3

To our best knowledge, no subset selection method is tailored for CVDP. In this work, we select some typical subset selection methods that are original designed for CPDP task (as mentioned in Section 2.2) as our baseline methods. The modules of the prior (current) version in CVDP scenario are treated as the ones of the source (target) project in CPDP scenario. In this work, we also consider the variants of these methods as our baseline methods to enrich our experiments. The descriptions of these baseline methods are as follows:

TF1: A variant of the TF method (Turhan et al., 2009) as mentioned in Section 2.2 which set the k' as 1. This comes from the fact the this parameter is directly related to the number of selected modules, thus may affect the CVDP performance.

TF2: The original TF method (Turhan et al., 2009) with $k' = 10$.

PF1: The original PF method (Peters et al., 2013). We follow the original study to set the parameter k of k-means algorithm as $\frac{M+n}{10}$ (where M and n denote the initial number of modules in the prior and current version respectively), expecting that each cluster tends to have 10 modules in average. Note that the k-means algorithm needs to randomly initialize the central points, thus, we run PF1 30 times and record the average indicator values.

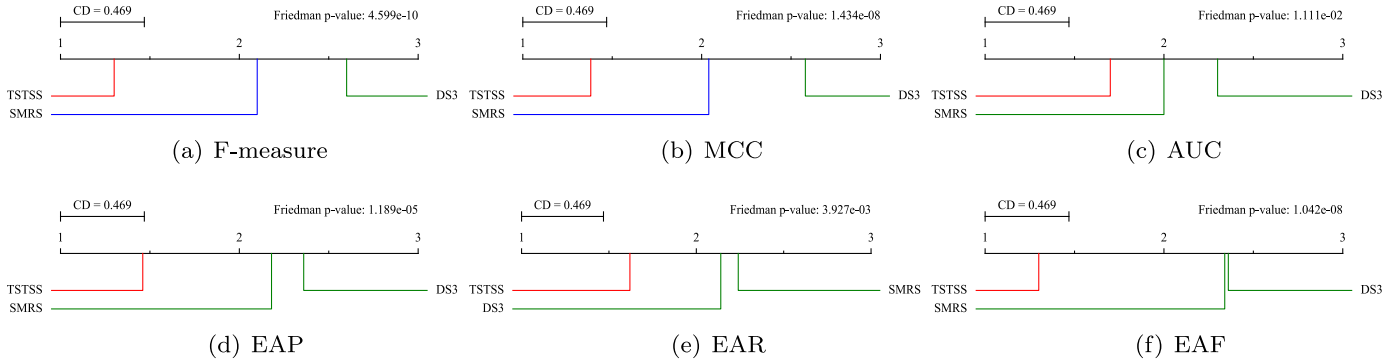


Fig. 12. Comparison of TSTSS against SMRS and DS3 with Friedman test and Nemenyi post-hoc test in terms of all 6 indicators.

PF2: A variant of PF method. The difference between PF1 and PF2 is that, for each popular module in the current version, PF1 only reserves its nearest neighbor module in the prior version, while PF2 reserves all the modules of the prior version in the clusters which contain at least one module of the current version. The setting is following the original KF method (Kawata et al., 2016).

YF1: A variant of original YF method (Yu et al., 2017). For each popular module in the current version, YF1 only select its nearest neighbor module of the prior version in the reserved clusters as PF1.

YF2: The original YF method (Yu et al., 2017). We also follow the original study to set the cluster number as $\frac{M+n}{10}$ as the PF method.

KF1: The original KF method (Kawata et al., 2016). We follow the original study to set the two parameters, i.e., the number of minimum records and the distance, as 10 and 1, respectively.

KF2: A variant of original KF method (Kawata et al., 2016) that sets the two parameters as 6 and 0.9, respectively. This comes from the fact that the performance of KF is sensitive to the parameter setting. So we also employ the default parameter in Weka tool (Bouckaert et al., 2009) to implement the KF2 method for comparison.

In addition, we consider the method ALL that does not perform any subset selection strategy as the most basic method for comparison.

Fig. 13(a) shows the radar charts of the average performance values on the cross-version pairs for each project as well as across all projects in terms of TSTSS and the 9 baseline methods. From the figure, we observe that:

First, in terms of the 3 traditional indicators, from Fig. 13(r), we find that TSTSS achieves the best average values in terms of F-measure and MCC across the 50 cross-version pairs. More specifically, average F-measure by TSTSS gains the improvement of 38.4% compared with the best average F-measure (for KF2) among the 9 baseline methods; average MCC by TSTSS gains the improvement of 44.3% compared with the best average MCC (YF2) among the 9 baseline methods. While average AUC by TSTSS is 0.007 lower that the best average AUC (for TF1) among the 9 baseline methods.

Second, in terms of the 3 traditional indicators, from Fig. 13(a), (c), (k), (m), and (n), the results show that all 3 average indicator values by TSTSS are better than that by the 9 baseline methods on project ant, ivy, xerces, prop2, and prop3, respectively. In addition, Fig. 13(b), (j), and (o) show that TSTSS achieves the best average F-measure and MCC than the 9 baseline methods on project camel, xalan, and prop4, respectively.

Third, in terms of the 3 effort-aware indicators, we also observe that TSTSS achieves the best average values on the 3 indicators across the 50 cross-version pairs from Fig. 13(r). More specifically, average EAP by TSTSS gains the improvement of 5.1% compared with the best average EAP (for TF1) among the 9 baseline methods; average EAR by TSTSS gains the improvement of 5.4% compared with the best average EAR (ALL, TF2 and YF1) among the 9 baseline methods; average EAF by TSTSS gains the improvement of 6.9% compared with the best average EAF (TF1 and YF2) among the 9 baseline methods.

Fourth, in terms of the 3 effort-aware indicators, from Fig. 13(a), (c), (d), (f), and (m), the results show that all 3 average effort-aware indicator values of TSTSS are higher than the 9 baseline methods on project ant, ivy, jedit, lucene, and prop2, respectively. In addition, Fig. 13(b), (e), (h), and (p) show that TSTSS achieves the best average EAR and EAF than the 9 baseline methods on project camel, log4j, synapse, and prop5, respectively. Fig. 13(g), (k), and (o) show that TSTSS achieves the best average EAP and EAF than the 9 baseline methods on project poi, xerces, and prop4, respectively.

Fifth, Fig. 14 visualizes the results of Friedman test and Nemenyi post-hoc test for the 10 methods in terms of the 6 indicators. The p values of Friedman test are all less than 0.05, which indicates that the differences among the 10 methods can be explained by statistically significant in terms of all 6 indicators. The Nemenyi test results show that TSTSS achieves the best average rank on 5 indicators except for AUC. TSTSS performs significantly better than the 9 baseline methods in terms of F-measure, MCC, and EAF, whereas has no significant differences compared with 6, 3, 3 baseline methods in terms of AUC, EAP, and EAR, respectively.

Analysis: The principles of TSTSS and the 9 baseline methods are different. TSTSS selects the subset by solving an two-stage optimization problem, i.e., optimizing the reconstruction error and the representing cost, instead of simple distance based and clustering based selection as the baseline methods do. The characteristic of TSTSS is that the selected modules are not only important to the data of the prior version but also representative to the data of the current version, therefore, the elaborately selected module subset by TSTSS is more refined and targeted. In addition, TSTSS can constrain the number of the selected modules in each stage by adjusting the corresponding parameters, while all the 8 baseline subset selection methods cannot determine how many modules are selected. This means that TSTSS is more flexible to control the proportion of the training set according to the practice conditions, such as the constraints of the memory and computing power.

Answer to RQ3: To sum up, TSTSS is more effective to select an optimum training subset to enhance CVDP performance than other subset selection methods.

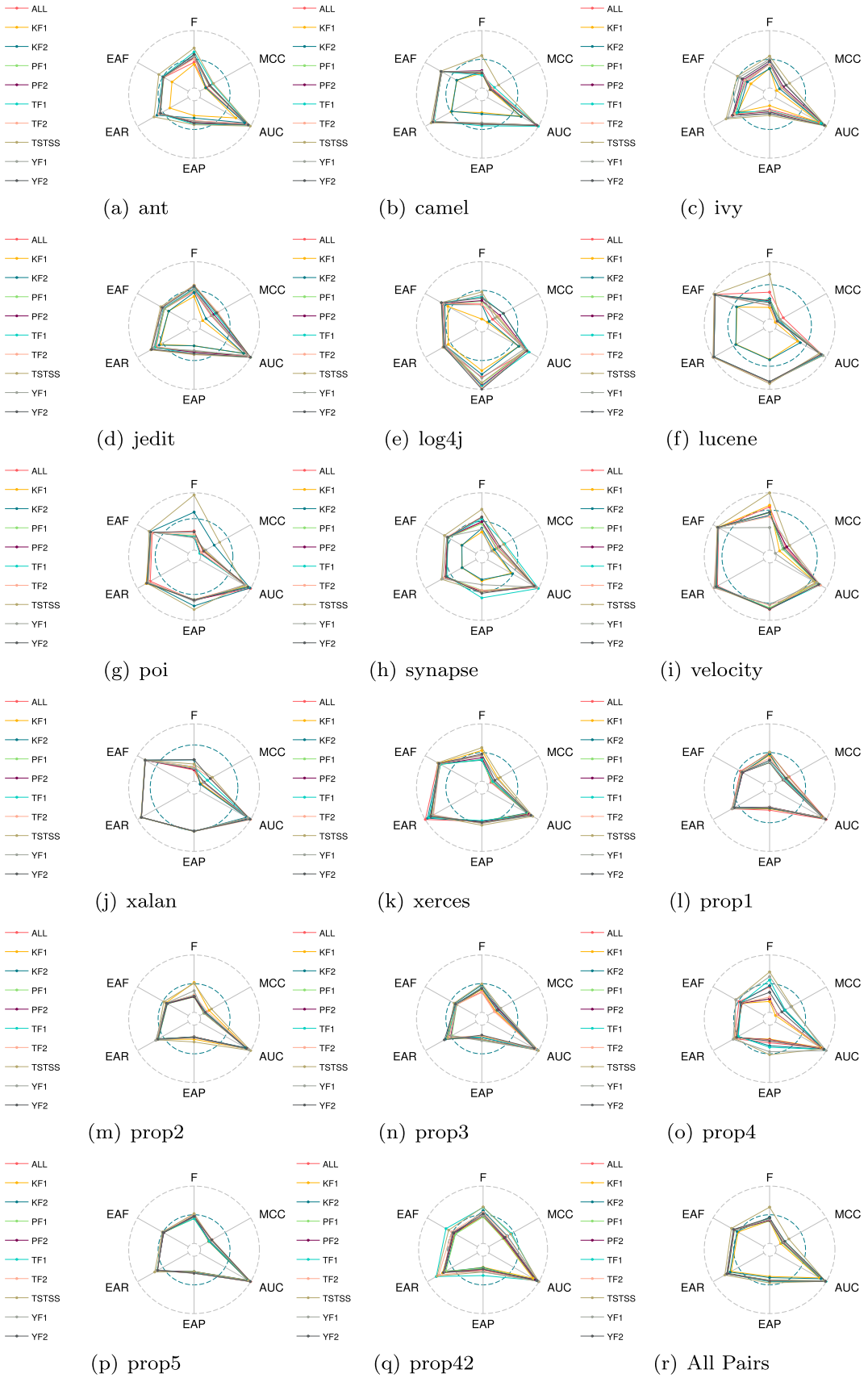


Fig. 13. Radar charts of average values of the 6 indicators on the cross-version pairs of each project and across all projects in terms of TSTSS and 9 baseline methods.

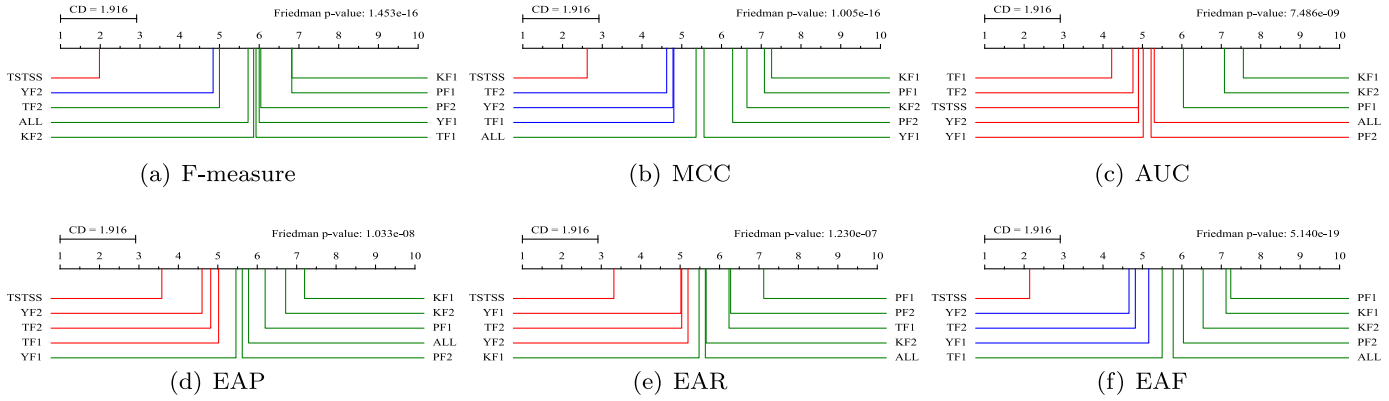


Fig. 14. Comparison of TSTSS against SMRS and DS3 with Friedman test and Nemenyi post-hoc test in terms of all 6 indicators.

5.4. Results for RQ4

This question mainly focuses on two differences among these training subset selection methods. The first one is the number of selected candidates since the method that does not degrade performance with fewer selected candidate modules is desired. The second one is the defect ratios of the selected training sets by these methods since training data with higher defect ratios are helpful to train a more effective model for the identification of the defective modules. For these purposes, we record the number of selected modules by these methods and the corresponding percentages of the defective modules.

The table is available online,[4] where # SM and % SD denote the number of the selected modules and the corresponding defect ratios. The defect ratio values are with gray shade or bold when they are higher or the same compared with the original ones, respectively. From the table, we have the following observations:

First, TSTSS selects the fewest modules on 42 out of 50 cross-version pairs compared with the first 6 baseline methods. This observation indicates that TSTSS is more effective for data reduction while still maintains the competitive CVDP performance as mentioned in Section 5.3.

Second, in terms of KF1 and KF2, they select fewer modules on 24 out of total 29 cross-version pairs than TSTSS over the first 11 open-source projects. The reason is that they treat many modules as noises and discard them. However, they may select only one-class modules on some cross-version pairs, such as on pairs ⟨camel 1.0, camel 1.2⟩, ⟨lucene 2.0, lucene 2.2⟩, and ⟨synapse 1.0, synapse 1.1⟩. This drawback makes the classifier fail to build a discriminative CVDP model. On the contrary, TSTSS selects fewer modules than KF1 and KF2 on 18 out of total 21 cross-version pairs over the last 6 proprietary projects. As the difference between the open-source and proprietary projects is that the scale of the latter (expect for the prop42 project) is much larger than that of the former, this indicates that TSTSS is more effective to reduce the training set on large-scale defect data than KF1 and KF2.

Third, the new training set by TSTSS has higher or similar defect ratios compared with the original one on 33 out of 50 cross-version pairs, and the defect ratios of the new training sets by the first 5 baseline methods are also improved on more than half of the total cross-version pairs. While the last 3 baseline methods improve the defect ratios of the new training set on few pairs. This indicates that the training subset selection methods have greater advantages to improve the defect ratio, which is beneficial for the classification models to the identification of defective modules.

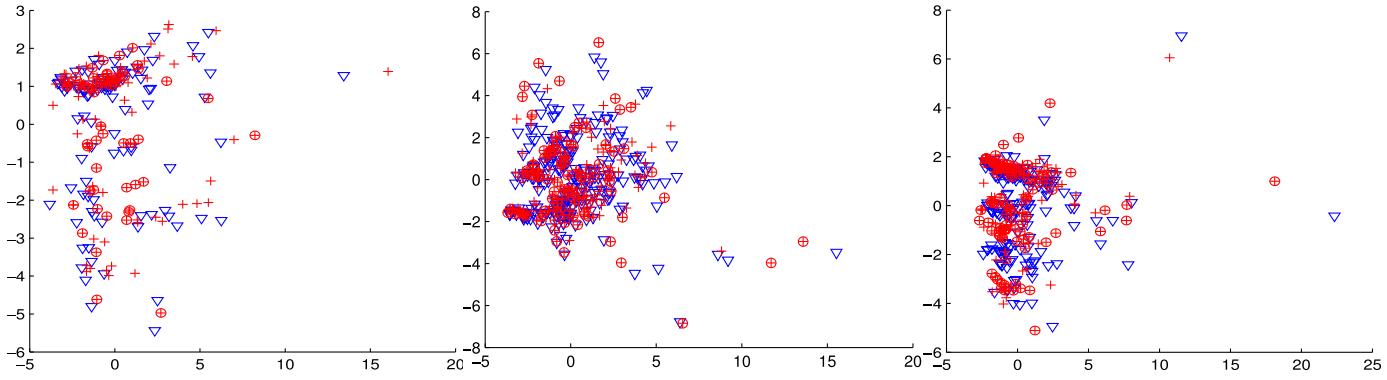
Analysis: as the baseline methods select the nearest modules directly, or first cluster the modules and then select the nearest modules in each cluster as the candidates, thus, the positions of the selected modules are nearest to the ones of the current version in the feature space. Here, we carry out an initial experiment to analyze what kind of modules are selected by our TSTSS method. Since the defect data consist of multidimensional features, we need to use a dimensional reduction technique to convert the original data into a visual plane for easy observation. For this purpose, we utilize a classic and commonly-used method **P**rincipal **C**omponent **A**nalysis (**PCA**) to transform the defect data into a two-dimensional data, and then randomly select several cross-version pairs (such as ⟨log4j-1.0, log4j-1.1⟩, ⟨synapse-1.1, synapse-1.2⟩, and ⟨velocity-1.4, velocity-1.5⟩) for observation. First, we apply our proposed TSTSS method to select the candidates from the prior version, then use the PCA to convert the original data of two versions by maintaining 2 features (i.e., the two largest eigenvectors corresponding to the two largest eigenvalues), finally, visualize the scatter plots on a two-dimensional plane and mark the selected candidates. Fig. 15 show the visualization results on the 3 cross-version pairs. In the figure, the blue triangle '∇' represents the module in the current version, the red cross '+' represents the module in the prior version, and the red cross with a red circle '$\oplus$' represents the selected candidate from the prior version. From the figure, we observe two general rules about the selected modules by TSTSS: first, TSTSS selects more modules from the prior version in the density regions; second, in sparse region, not all the nearest modules towards the current version modules will be selected, instead, TSTSS chooses a few modules from the neighbors as the representatives. This maybe the reason why TSTSS can select fewer modules from the prior version.

Answer to RQ4: From the above observations, TSTSS selects fewer modules (except compared with KF1 and KF2 on small scale defect data) with promising CVDP performance. Thus, TSTSS is a more preferred training subset selection method for CVDP task. Besides, TSTSS and the first 5 baseline methods have the potential to relieve the class imbalance issue to a certain extent by increasing the defect ratios of the selected training sets on most cross-version pairs.

6. Threats to validity

Recently, some software engineering researchers have shown that the experimental components, such as the experimental setup, may have impacts on the conclusions of the studies (Menzies and Shepperd, 2012; Tantithamthavorn, 2016; Tantithamthavorn et al., 2017). In this paper, we identify some potential threats to the validity of our work.

[4] https://figshare.com/s/c267a5aefb89aafd917e.



(a) ⟨log4j-1.0, log4j-1.1⟩ (b) ⟨synapse-1.1, synapse-1.2⟩ (c) ⟨velocity-1.4, velocity-1.5⟩

Fig. 15. Visualization results of the selected candidates on 3 cross-version pairs.

6.1. Threats to external validity

External validity focuses on the generalization of the experimental conclusions to other datasets. To minimize the threats, we carefully choose publicly available defect data including both open-source and industrial projects as our benchmark dataset. However, the features of the projects are all object-oriented static code features, thus, whether our findings are identical to other datasets with process features, network features and text features is unclear. In addition, since the 17 projects are developed with Java language, replication studies on other projects that developed with C, C++, or Python may prove fruitful.

6.2. Threats to internal validity

Internal validity pays attention to the potential limitations of the experiments, such as the parameter settings, and the implementation of baseline methods. Regarding parameter settings for TSTSS and WELM, we only set some coarse-grained options to search the optimal parameters. A fine-grained settings will be considered in our future work. Regarding the reproduction of the baseline methods, we carefully implement them by following the corresponding studies. However, our implementation may not be able to fully reproduce the original methods which may lead to a bias in the comparison between our method and the baseline methods.

6.3. Threats to construct validity

Construct validity relates to the suitability of the evaluation indicators and the statistic test methods. In this study, we choose 3 typical and 3 effort-aware indicators as our measurement criteria, which enables us to have a comprehensive evaluation of the CVDP performance. Measuring the CVDP performance with other indicators is left for future work. In addition, we use the Frideman test with Nemenyi post-hoc test to statistically analyze the differences between TSTSS and the baseline methods. The two statistic tests have been successfully applied to previous defect prediction studies (Lessmann et al., 2008; Mende and Koschke, 2010; Jiang et al., 2008; DAmbros et al., 2012; Nam and Kim, 2015; Li et al., 2017; 2018b).

7. Conclusion

CVDP is a more practical defect prediction scenario than cross-validation prediction and CPDP, but only a few studies specialize in this topic. To relieve the adverse effect of the distribution differences between the data of two consecutive versions on the CVDP performance, in this work, we propose a novel two-stage subset selection framework TSTSS to meticulously pick up a favorable module subset from the prior version with two optimization methods. The selected module subset can both well reconstruct the original data of the prior version and effectively represent the data of the current version. In addition, a novel weighted based classifier WELM is introduced to build a more ideal CVDP model targeted the imbalanced defect data. Extensive experiments are conducted on 50 cross-version pairs from 67 versions of 17 projects with total 6 indicators. The experimental results prove the effectiveness of our TSTSS method for improving CVDP performance compared with 11 baseline methods.

In the future, we plan to evaluate our method on more defect data with other types of features and consider more fine-grained parameter settings. In addition, we will investigate the applicability of TSTSS for CPDP scenario and other software engineering problems, such as test case selection.

Acknowledgements

The authors would like to acknowledge the support provided by National Key R&D Program of China (2018YFC1604000), the grands of the National Natural Science Foundation of China (61572374, U163620068, U1135005, U1636220, 61572371, 61772525, 61472423, 61602258), the Open Fund of Key Laboratory of Network Assessment Technology from CAS, Guangxi Key Laboratory of Trusted Software (kx201607), the Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012), the National Science Foundation (DGE-1522883), Hong Kong GRC Project (PolyU 152223/17E, PolyU 152239/18E, CityU C1008-16G), and the China Postdoctoral Science Foundation (2017M621247).

References

Arisholm, E., Briand, L.C., Johannessen, E.B., 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J. Syst. Software (JSS) 83 (1), 2–17.

Bennin, K.E., Toda, K., Kamei, Y., Keung, J., Monden, A., Ubayashi, N., 2016. Empirical evaluation of cross-release effort-aware defect prediction models. In: Proceedings of the 2nd International Conference on Software Quality, Reliability and Security (QRS), pp. 214–221.

Bin, Y., Zhou, K., Lu, H., Zhou, Y., Xu, B., 2017. Training data selection for cross-project defection prediction: which approach is better? In: Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 354–363.

Bouckaert, R.R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., Scuse, D., 2009. Weka manual for version 3-6-1. The University of Waikato: Hamilton, New Zealand, pp. 1–341.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. 3 (1), 1–122.

Catolino, G., Palomba, F., De Lucia, A., Ferrucci, F., Zaidman, A., 2017. Developer-related factors in change prediction: an empirical assessment. In: Proceedings of the 25th International Conference on Program Comprehension (ICPC), pp. 186–195.
Chen, L., Fang, B., Shang, Z., Tang, Y., 2015. Negative samples reduction in cross–company software defects prediction. Inf. Software Technol. (IST) 62, 67–77.
Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7 (Jan), 1–30.
DAmbros, M., Lanza, M., Robbes, R., 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empir. Software Eng. (ESE) 17 (4–5), 531–577.
Elhamifar, E., Sapiro, G., Sastry, S.S., 2016. Dissimilarity-based sparse subset selection. Trans. Pattern Anal. Mach.Intell. (TPAMI) 38 (11), 2182–2197.
Elhamifar, E., Vidal, R., 2009. Sparse subspace clustering. In: Proceedings of the 22nd International Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2790–2797.
Fletcher, R., 2013. Practical methods of optimization.
Gabay, D., Mercier, B., 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. Comput. Math. Appl. 2 (1), 17–40.
Ghaffarian, S.M., Shahriari, H.R., 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a survey. Comput. Surv. (CSUR) 50 (4), 1–36.
Ghotra, B., McIntosh, S., Hassan, A.E., 2015. 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 (ICSE), pp. 789–800.
He, Z., Shu, F., Yang, Y., Li, M., Wang, Q., 2012. An investigation on the feasibility of cross-project defect prediction. Autom. Software Eng. (ASE) 19 (2), 167–199.
Herbold, S., Trautsch, A., Grabowski, J., 2017. A comparative study to benchmark cross-project defect prediction approaches. IEEE Trans. Software Eng.
Holschuh, T., Pauser, M., Herzig, K., Zimmermann, T., Premraj, R., Zeller, A., 2009. Predicting defects in sap java code: An experience report. In: Proceedings of the 31st International Conference on Software Engineering (ICSE), pp. 172–181.
Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., 2004. Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN), Vol. 2, pp. 985–990.
Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., 2006. Extreme learning machine: theory and applications. Neurocomputing 70 (1–3), 489–501.
Huang, Q., Xia, X., Lo, D., 2017. Supervised vs unsupervised models: a holistic look at effort-aware just-in-time defect prediction. In: Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME), pp. 159–170.
Huang, S.-J., Jin, R., Zhou, Z.-H., 2014. Active learning by querying informative and representative examples. Trans. Pattern Anal. Mach.Intell. (TPAMI) 10 (36), 1936–1949.
Jiang, T., Tan, L., Kim, S., 2013. Personalized defect prediction. In: Proceedings of the 28th International Conference on Automated Software Engineering (ASE), pp. 279–289.
Jiang, Y., Cukic, B., Ma, Y., 2008. Techniques for evaluating fault prediction models. Empir. Software Eng. (ESE) 13 (5), 561–595.
Jing, X., Wu, F., Dong, X., Qi, F., Xu, B., 2015. Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE), pp. 496–507.
Jing, X.-Y., Wu, F., Dong, X., Xu, B., 2017. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. Trans. Software Eng. (TSE) 43 (4), 321–339.
Jing, X.-Y., Ying, S., Zhang, Z.-W., Wu, S.-S., Liu, J., 2014. Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering (ICSE), pp. 414–423.
Kamei, Y., Shihab, E., 2016. Defect prediction: accomplishments and future challenges. In: Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 5, pp. 33–45.
Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N., 2013. A large-scale empirical study of just-in-time quality assurance. Trans. Software Eng. (TSE) 39 (6), 757–773.
Kawata, K., Amasaki, S., Yokogawa, T., 2016. Improving Relevancy Filter Methods for Cross-project Defect Prediction. In: Applied Computing & Information Technology, pp. 1–12.
Khoshgoftaar, T.M., Seliya, N., 2003. Fault prediction modeling for software quality estimation: comparing commonly used techniques. Empir. Software Eng. (ESE) 8 (3), 255–283.
Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. Trans. Software Eng. (TSE) 34 (4), 485–496.
Li, M., Zhang, H., Wu, R., Zhou, Z.-H., 2012. Sample-based software defect prediction with active and semi-supervised learning. Autom. Software Eng. (ASE) 19 (2), 201–230.
Li, X., Guo, Y., 2013. Adaptive active learning for image classification. In: Proceedings of the 26th Conference on Computer Vision and Pattern Recognition (CVPR), pp. 859–866.
Li, Y., Su, J., Yang, X., 2018a. Multi-objective vs. single-objective approaches for software defect prediction. In: Proceedings of the 2nd International Conference on Management Engineering, Software Engineering and Service Sciences, pp. 122–127.
Li, Z., Jing, X.-Y., Wu, F., Zhu, X., Xu, B., Ying, S., 2018b. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Autom. Software Eng. (ASE) 25 (2), 201–245.
Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., Ying, S., 2017. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. Trans. Software Eng. (TSE).
Lu, H., Kocaguneli, E., Cukic, B., 2014. Defect prediction between software versions with active learning and dimensionality reduction. In: Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE), pp. 312–322.
Madeyski, L., Jureczko, M., 2015. Which process metrics can significantly improve defect prediction models? an empirical study. Software Quality Journal (SQJ) 23 (3), 393–422.
Mende, T., Koschke, R., 2010. Effort-aware defect prediction models. In: Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR), pp. 107–116.
Menzies, T., Greenwald, J., Frank, A., 2007. Data mining static code attributes to learn defect predictors. Trans. Software Eng. (TSE) 33 (1), 2–13.
Menzies, T., Shepperd, M., 2012. Special issue on repeatable results in software engineering prediction. Empir. Software Eng. (ESE) 1–17.
Moha, N., 2007. Detection and correction of design defects in object-oriented designs. In: Companion to the 22nd Sigplan Conference on Object-oriented Programming Systems and Applications Companion, pp. 949–950.
Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M., Matsumoto, K., 2013. Assessing the cost effectiveness of fault prediction in acceptance testing. Trans. Software Eng. (TSE) 39 (10), 1345–1357.
Nam, J., Kim, S., 2015. Clami: Defect prediction on unlabeled datasets. In: Proceedings of the 30th International Conference on Automated Software Engineering (ASE), pp. 452–463.
Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp. 382–391.
Peters, F., Menzies, T., Marcus, A., 2013. Better cross company defect prediction. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), pp. 409–418.
Prasad, K.M., Bapat, R., 1992. The generalized Moore–Penrose inverse. Linear Algebra Appl. 165, 59–69.
Rahman, F., Posnett, D., Devanbu, P., 2012. Recalling the imprecision of cross-project defect prediction. In: Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE), pp. 1–11.
Shukla, S., Radhakrishnan, T., Muthukumaran, K., Neti, L.B.M., 2018. Multi-objective cross-version defect prediction. Soft Comput. 22 (6), 1959–1980.
Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J., 2011. A general software defect-proneness prediction framework. Trans. Software Eng. (TSE) 37 (3), 356–370.
Tantithamthavorn, C., 2016. Towards a better understanding of the impact of experimental components on defect prediction modelling. In: Proceedings of the International Conference on Software Engineering Companion (ICSE-C), pp. 867–870.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. Trans. Software Eng. (TSE) 43 (1), 1–18.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2018. The impact of automated parameter optimization on defect prediction models. Trans. Software Eng. (TSE) to appear.
Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J., 2009. On the relative value of cross-company and within-company data for defect prediction. Empir. Software Eng. (ESE) 14 (5), 540–578.
Vidal, R., Sapiro, G., Elhamifar, E., 2012. See all by looking at a few: sparse modeling for finding representative objects. In: Proceedings of the 25th Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1600–1607.
Wang, S., Liu, T., Tan, L., 2016a. Automatically learning semantic features for defect prediction. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), pp. 297–308.
Wang, S., Yao, X., 2013. Using class imbalance learning for software defect prediction. Trans. Reliab. (TR) 62 (2), 434–443.
Wang, T., Zhang, Z., Jing, X., Zhang, L., 2016b. Multiple kernel ensemble learning for software defect prediction. Autom. Software Eng. (ASE) 23 (4), 569–590.
Watanabe, S., Kaiya, H., Kaijiri, K., 2008. Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, pp. 19–24.
Xia, X., Lo, D., Pan, S.J., Nagappan, N., Wang, X., 2016. Hydra: massively compositional model for cross-project defect prediction. Trans. Software Eng. (TSE) 42 (10), 977–998.
Xu, Z., Li, S., Tang, Y., Luo, X., Zhang, T., Liu, J., Xu, J., 2018a. Cross version defect prediction with representative data via sparse subset selection. In: Proceedings of the 26th International Conference on Program Comprehension (ICPC).
Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y., Zhang, T., 2019. Software defect prediction based on kernel pca and weighted extreme learning machine. Inf. Software Technol. (IST) 106, 182–200.
Xu, Z., Liu, J., Luo, X., Zhang, T., 2018b. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In: Proceedings of the 25th International Conference on Software Analysis, Evolution, and Reengineering (SANER), p. to appear.

Xu, Z., Liu, J., Yang, Z., An, G., Jia, X., 2016a. The impact of feature selection on defect prediction performance: an empirical comparison. In: Proceedings of the 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 309–320.
Xu, Z., Xuan, J., Liu, J., Cui, X., 2016b. 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 (SANER), Vol. 1, pp. 370–381.
Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J., 2015a. Deep learning for just-in-time defect prediction. In: Proceedings of the International Conference on Software Quality, Reliability and Security (QRS), pp. 17–26.
Yang, X., Wen, W., 2018. Ridge and lasso regression models for cross-version defect prediction. Trans. Reliab. (TR) 67 (3), 885–896.
Yang, Y., Harman, M., Krinke, J., Islam, S., Binkley, D., Zhou, Y., Xu, B., 2016a. An empirical study on dependence clusters for effort-aware fault-proneness prediction. In: Proceedings of the 31st International Conference on Automated Software Engineering (ASE), pp. 296–307.
Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., Leung, H., 2016b. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: Proceedings of the 24th SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp. 157–168.
Yang, Y., Zhou, Y., Lu, H., Chen, L., Chen, Z., Xu, B., Leung, H., Zhang, Z., 2015b. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. Trans. Software Eng. (TSE) 41 (4), 331–357.
Yu, X., Zhou, P., Zhang, J., Liu, J., 2017. A data filtering method based on agglomerative clustering. In: Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 392–397.
Zar, J.H., et al., 1999. Biostatistical analysis.
Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B., 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the SIGSOFT Symposium on The Foundations of Software Engineering (FSE), pp. 91–100.
Zong, W., Huang, G.-B., Chen, Y., 2013. Weighted extreme learning machine for imbalance learning. Neurocomputing 101, 229–242.

Zhou Xu is a joint PhD student in the School of Computer Science at Wuhan University, China and in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also a temporary research assistant at the College of Computer Science and Technology, Harbin Engineering University, China. His research interests include software defect prediction, feature engineering, data mining.

Shuai Li is a PhD student in the Department of Computing at The Hong Kong Polytechnic University. He received his BS degree in Computer Science from Dalian University of Technology in 2017. His research interests are in classification and detection both in theory and applications.

Xiapu Luo received the PhD degree in computer science from The Hong Kong Polytechnic University. He was a post-doctoral research fellow in the Georgia Institute of Technology. He is currently a research assistant professor in the Department of Computing and an associate researcher with the Shenzhen Research Institute, The Hong Kong Polytechnic University. His current research focuses on smartphone security and privacy, network security and privacy, and Internet measurement.

Jin Liu received the PhD degree in computer science from the State Key Lab of Software Engineering, Wuhan University, China, in 2005. He is currently a professor at School of Computer Science, Wuhan University. He has authored or coauthored a number of research papers in international journals. His research interests include mining software repositories and intelligent information processing.

Tao Zhang is an associate professor in the College of Computer Science and Technology at Harbin Engineering University. He spent one year as a postdoctoral research fellow in the Department of Computing at The Hong Kong Polytechnic University. He got his Ph.D. in Computer Science from the University of Seoul, South Korea while he received his master degree and bachelor degree at Northeastern University, China. His research interest includes mining software repositories and empirical software engineering.

Yutian Tang received his Ph.D. degree from The Hong Kong Polytechnic University in 2018 and his BSc from Jilin University in 2013. Currently, he is a research staff in The Hong Kong Polytechnic University. His research interests include software product line, system security and machine learning.

Jun Xu received the B.Sc. degree in Pure Mathematics and the M. Sc. Degree in Information and Probability both from the School of Mathematics Science, Nankai University, China, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree in the Department of Computing, The Hong Kong Polytechnic University. His research interests include image restoration, subspace clustering, sparse and low rank models.

Peipei Yuan received the B.Sc. degree in information and computation science from Huazhong Agricultural University in 2014. She is currently pursuing her Ph.D. degree in the School of Electronic Information and Communications, Huazhong University of Science and Technology, China. Her research interests include computer vision, pattern recognition, machine learning and data mining.

Jacky Keung received the BSc (Hons) degree in computer science from the University of Sydney, and the PhD degree in software engineering from the University of New South Wales, Australia. He is assistant professor in the Department of Computer Science, City University of Hong Kong. His main research interests include software effort and cost estimation, empirical modeling and evaluation of complex systems, and intensive data mining for software engineering datasets. He has published papers in prestigious journals including the IEEE Transactions on Software Engineering, the Empirical Software Engineering, and many other leading journals and conferences. He is a member of the IEEE.