Information and Software Technology 149 (2022) 106939

Contents lists available at ScienceDirect

Information and Software Technology

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

ST-TLF: Cross-version defect prediction framework based transfer learning

Yanyang Zhao [a,b], Yawen Wang [a,b,*], Yuwei Zhang [c], Dalin Zhang [d], Yunzhan Gong [a], Dahai Jin [a]

[a] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*
[b] *Guangxi Key Laboratory of Cryptography and Information Security, Guilin, Guangxi, China*
[c] *Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China*
[d] *School of Software Engineering, Beijing Jiaotong University, Beijing, China*

ARTICLE INFO

Keywords:
Software testing
Defect prediction
Concept drift
Transfer learning
Clustering analysis

ABSTRACT

Context: Cross-version defect prediction (CVDP) is a practical scenario in which defect prediction models are derived from defect data of historical versions to predict potential defects in the current version. Prior research employed defect data of the latest historical version as the training set using the empirical recommended method, ignoring the concept drift between versions, which undermines the accuracy of CVDP.
Objective: We customized a **S**elected **T**raining set and **T**ransfer **L**earning **F**ramework (ST-TLF) with two objectives: a) to obtain the best training set for the version at hand, proposing an approach to select the training set from the historical data; b) to eliminate the concept drift, designing a transfer strategy for CVDP.
Method: To evaluate the performance of ST-TLF, we investigated three research problems, covering the generalization of ST-TLF for multiple classifiers, the accuracy of our training set matching methods, and the performance of ST-TLF in CVDP compared against state-of-the-art approaches.
Results: The results reflect that (a) the eight classifiers we examined are all boosted under our ST-TLF, where SVM improves 49.74% considering MCC, as is similar to others; (b) when performing the best training set matching, the accuracy of the method proposed by us is 82.4%, while the experience recommended method is only 41.2%; (c) comparing the 12 control methods, our ST-TLF (with BayesNet), against the best contrast method P15-NB, improves the average MCC by 18.84%.
Conclusions: Our framework ST-TLF with various classifiers can work well in CVDP. The training set selection method we proposed can effectively match the best training set for the current version, breaking through the limitation of relying on experience recommendation, which has been ignored in other studies. Also, ST-TLF can efficiently elevate the CVDP performance compared with random forest and 12 control methods.

1. Introduction

With the development of software technology, the reliability of software is increasingly required, especially with the increasing reliance on software in all areas of our society [1], not just aerospace and finance [2]. Before the release of the project at hand, effectively testing and repairing the defects are crucial for improving the quality of the software and lowering the technical debt [3]. Defect prediction based learning algorithm is one of the most important methods for software reliability maintenance [4], including defect-prone prediction [5], code vulnerability prediction [6], fragile code prediction [7]. Thanks to the popularization of project management tools [8], e.g., the version control system [9] and the defect tracking system [10], these tools can obtain historical defect data from the software repository. To improve the reliability of the software system, previous studies have proposed various defect prediction models based on available defect data to identify potential defects in targeted projects [11–13].

Most investigations watching defect prediction provide plentiful technical references and significant practical guidance for Cross-project defect prediction (CPDP) and Inner-version defect prediction (IVDP) [14–16]. Such studies, however, are rare regarding Cross-version defect prediction (CVDP) [17]. For the projects with multiple versions, CVDP is a practical scenario by training the defect prediction models on the defect data of the historical versions to predict the defect labels of modules in the upcoming version [17,18]. Contrasting other projects, the version at hand can inherit and refactor some modules from the prior version, which makes the distribution of defect data between

* Corresponding author.
E-mail address: wangyawen@bupt.edu.cn (Y. Wang).

[1] The NLE is the complexity of the method expressing as the depth of the maximum embeddedness of conditional, iteration, and exception handling block scopes, wherein the 'if-else-if' construct only the first 'if' instruction is considered. The following instructions are taken into account: if, else, for, while, dowhile, switch, try, catch, finally and block statements that are directly inside another block statement.

https://doi.org/10.1016/j.infsof.2022.106939
Received 14 November 2021; Received in revised form 3 May 2022; Accepted 5 May 2022
Available online 14 May 2022

versions more similar [17]. Unfortunately, the statistical properties of the current version (e.g., dependencies and cyclomatic complexity) change over time in an arbitrary way against the historical version, which is called concept drift [19–21]. For example, the change of Nesting Level Else-If (NLE)[1] for the *getResourceStream* in class *AntClassLoader* for Ant1.3 & Ant1.4. The NLE changed from 5 (i.e., try ⇢ if⇢ try ⇢ if ⇢ while) in Ant1.3 to 3 (i.e., try ⇢ if⇢ if) in Ant1.4. Learning under concept drift is highly susceptible to producing labile prediction results [22]. And, concept drift is unavoidable in the CVDP, attributing to the change of requirements [23], the refactoring of code [24], and the repair of defects. A core assumption of any prediction model is that test data distribution does not differ from the training data distribution [21]. As a result, concept drift may result in decreased accuracy and reliability of the defective model, which will bring inevitable troubles to the testing work [25–28].

Prior work raised concerns regarding similar problems that are training and testing data from different versions. Turhan indicated that concept drift of training and test data is the main reason for the volatile results of defect prediction [19]. Dong et al. [20] demonstrated that concept drift makes the mean square error of training and test set high, which leads to the classification model with high accuracy but low recall. The studies argued that the difference in data distribution between versions reduces the reliability of the prediction model [17,29–31].

To mitigate the risk of concept drift, some researchers suggested implementing the practical of cross-project approaches in cross-version scenarios [29,32,33]. However, the defect data of CVDP has higher similarity than CPDP. Although some CPDP approaches are effective in CVDP, they favor the mixed data of source domain and target domain for data processing, which can dilute the defect information of the target version. Besides, transfer learning also provides a new direction to solve concept drift between versions [34]. Transfer learning aims to solve challenging learning problems between the target domain and the source domain [35–37], which is consistent with the scenario application of CVDP. Previous studies have demonstrated the ability of transfer learning to solve such problems [38,39]. For instance, Nam et al. applied a Transfer Component Analysis (TCA) approach to enhance feature distributions in source and target projects similar [40]. Chen et al. proposed two phases of collective transfer learning for the distribution differences between the source and target projects, including the source data expansion phase and adaptive weighting phase [41]. Xu et al. used the balanced distribution adaptation (BDA) based transfer learning method to narrow the data gap between versions [42]. These techniques that improve performance for CVDP focus on removing the invalid instances in historical defect data that can interfere with classifier decisions. While they ignore the adaptation between the new version and multiple historical versions.

Previous studies have also considered the problem of selecting training sets for the target version. They have employed the defect data of the (accessible) latest prior release as the training set of the defect model according to the practical experience [17,43,44]. And, they considered that the concept drift of data distribution between the last and current versions is the minimum. However, in our prior work, we found that the data diversities between the new version and different historical versions are different in the same project [45]. In other words, even between near versions, there may be an acute or slight concept drift problem. In practice, it is also unrealistic to construct defect models on each historical defect data for the target version. Choosing an optimal training set for the targeted version is one of the effective ways to avoid acute concept drift. Unfortunately, no such approach is currently available.

To solve the problems mentioned above, we propose a CVDP **F**ramework through two strategies: the **S**elected **T**raining set and **T**ransfer **L**earning, called **ST-TLF**. Firstly, we design a voting-based feature similarity detection method, named **ST**, combining multiple feature evaluators to obtain feature comparability between versions and then selected a training set with minor feature drift for the target version from multiple historical versions. Our ST maps the complex multi-dimensional defect features to a comparable dimension to avoid strong concept drift between source and target domains. And then, we customize an instance-based non-inductive transfer learning approach to select more effective defect instances in the dataset we selected for training the defect predictor. This method utilizes a cluster analyzer derived from defect data of the new version to screen instances in the historical version to an upcoming version, which perfectly employs the distribution characteristics of defect data in the target version. Combining the two techniques mentioned above, our ST-TLF overcomes concept drift between versions in CVDP.

To prove the validity of this method, we evaluated the effectiveness of our ST-TLF for CVDP performance using three performance metrics on 37 versions of 10 projects from an open-source defect dataset [46–48]. We first evaluated the universality of the framework by eight algorithms and then performed comparative experiments using the random forest and BayesNet as the basic classifiers of our framework. The results show that (a) our ST-TLF with different classifiers can work better than baseline, and (b) our ST-TLF achieves encouraging CVDP performance against the total of 13 approaches, including random forest and 12 control methods. Finally, the universality and limitations of the ST-TLF are analyzed. On this basis, the direction of further work is discussed.

In sum, this paper makes the following contributions:

- We found that defect data of the latest historical version suggested by the empirical recommended method was not always the optimal training set for the new version in CVDP.
- A dataset matching method based on the importance of variables is proposed to match the best training set for the target version in CVDP, which breaks through the limitations of empirical recommendation in previous studies.
- A sample-based non-inductive transfer learning is customized to solve concept drift in CVDP, which focuses on the distribution of defects in the target version.
- To improve the accuracy and reliability of CVDP, we designed the ST-TLF framework. The experimental results show that the ST-TLF method effectively improves the performance of the defect model comparing random forest and 12 control methods.

The remainder of this paper is organized as follows. Section 2 introduces technical background about transfer learning and related work of training set selection technology and CVDP. Section 3 illustrates the technical details of our ST-TLF method for solving the concept drift in CVDP. Section 4 presents the design of our case study, while Section 5 analyzes the results of each research problem. Section 6 discusses the importance of our research for research and practice alike, and Section 7 shows the threats to the validity of our study. Finally, Section 8 draws conclusions and prospects.

2. Related work

In this chapter, we provide the research background for CVDP, including transfer learning, training set matching technology, and related research of CVDP.

2.1. Transfer learning

Transfer learning is a machine learning paradigm [49]. The feature space X and edge probability distribution P^X of the target domain D_t and the source domain D_s for traditional machine learning are the same, whereas that of transfer learning is different [50]. There are many forms of transfer learning, where we are only concerned about instance-based transfer learning.

An implicit assumption behind the instance-based approaches is that the source domain D_s and the target domain D_t have a lot of overlapping features, which means that the domains share the same or similar support [50]. This is, instance-based transfer learning can reuse labeled data from the source domain D_s to help train a more accurate model for the target domain D_t [38,39], which can apply to the scenario for CVDP. In the source domain D_s, some labeled samples favor the training more accurate defect predictor for the target domain D_t, and some labeled instances can impair the generalization ability of the classifier. Stakeholders directly employ all samples from the source domain D_s to train the prediction model, which may not promote the target task in many applications. Therefore, the participator needs to filter out samples from the source domain D_s that meet the similar distribution of the target domain D_t, and applies them to training to reduce the bias of the new model. This is the same intent as cross-version defect prediction. There is no customized transfer learning method for CVDP, but similar research appears in CPDP. For example, Peters et al. designed a transferring subset strategy with a clustering algorithm on the mixed project data [51]. Ma et al. proposed Transfer Naive Bayes (TNB) using the information of all the proper features in training data [52]. Liu et al. [53] proposed a two-phase transfer learning model (TPTL) for CPDP to release the limitation of TCA+ proposed by Nam et al. [40]. And, prior studies apply select methods to select a representative module subset from the latest historical version based on the pairwise dissimilarities between the modules of the two versions [17,30]. Kawata et al. proposed a relevancy filter method on the mixed data of the target project and the original project to transfer qualified instances from the source domains for improving the accuracy of CPDP [54]. These studies provide support for designing transfer learning methods to improve the accuracy of CVDP. Based on these studies, we customized a defect prediction framework for CVDP through non-inductive transfer learning, which improved the accuracy of defect prediction through two data processes: training set matching and instance transferring.

2.2. Training set matching technology

Prior researches have fewer concerns about selecting training sets for CVDP. Amasaki evaluated the settings of two training sets in CVDP: single older version(SOV) and multiple older versions(MOV) [29]. The results show that MOV works better than SOV for 14 out of 20 CPDP approaches, recommending that practitioners employ mixed defect data from multiple historical versions as training sets. And then, in 2020, he subdivided the scenarios acquired by the training set, considering defect data of cross-projects [33]. The expanding study shows that Single Prior Version (SPV) without cross-project data (referring to the latest published Version) was the best scenario, which is inconsistent with previous research conclusions. Some studies recommended the defect data from the latest historical version as the training set according to practical experience, considering its data distribution to be most similar to that of the version at hand. However, in our other work, we found that the diversities between the new version and different historical versions are varying in the same project [45]. Therefore, the accuracy of defect predictors derived from different historical versions of defect data for the upcoming version also varies considerably, and even adjacent versions do not necessarily obtain reliable prediction results [45]. Currently, no research has proposed a customized training set matching method for CVDP. When considering related work, we investigated the research on CPDP. Only, He et al. [55] proposed a two-stage data filtering method riTDS composed of rTDS and iTDs for CPDP, in which the rTDS mainly selects source projects with similar data distribution to the target project from multiple source projects based on the target project. In this study, we focused on the difference in defect distribution between versions in the same project. Meanwhile, we design an ST method for matching training sets considering mapping the features in complex space to comparable space, which breaks through the limitations of experience recommendation in previous studies.

2.3. Cross-version defect prediction

As everyone knows, the performance of a machine learning-based classification model depends primarily on the data that trains it. The main difference between cross-version and inner-version defect predictions is derived from the training set. Therefore, empirical studies for IVDP are not fully applicable to CVDP. To investigate the performance of cross-version defect models, Bennin et al. evaluated the performance of CVDP by 11 machine learning algorithms with an effort-aware performance measure [43]. The experimental results on 25 open-source projects showed that M5 and Kstar models achieved the best performance. Besides, non-parametric statistical results show that there was no significant difference between the defect models they examined. Also, Shukla et al. proposed a multi-objective logistic regression(MOLR) algorithm concerning maximizing efficiency and minimizing cost [44], comparing single-target algorithms: Logistic Regression, Naive Bayes, Decision Tree, and Random Forest in MATLAB by F-measure and recall. The experimental results with 11 projects show that the MOLR model is superior to other single-target algorithms. Noting that Li et al. evaluated the performance of the MOLR and the single objective logistic regression for CVDP [56]. The experimental results show that the performance of the single-target method is better than the multi-objective method, which is not consistent with the results of [44]. Additionally, Yang and Wen compared Ridge regression (RR) and the performance of other algorithms [57], where three the ridge parameters are selected: GCV [58], HKB [59], and LW [60]. The results show that GCV is slightly better for choosing the ridge parameter of RR for CVDP than LW and HKB, but not significant, in which RR can achieve better results than linear regression(LR) and NBR [61]; slightly (not significant) better results than Lasso regression(LAR), principal component regression(PCR) and learning-to-rank(LTR); and slightly (not significant) worse results than Random Forest. Although we are not a customized defect model survey for CVDP, we examined a variety of defect models in comparative experiments using common performance metrics, expanding the experience of CVDP in the software community.

The surveys mentioned above mainly focus on the general performance of defect models in CVDP but do not contribute to solving concept drift. The concept drift across versions can degrade the performance of the classification model, which brings trouble to the test work. Xu et al. approached this issue by leveraging a state-of-the-art Dissimilarity-based Sparse Subset Selection (DS3) method [30]. Meanwhile, they combined Hybrid Active Learning and Kernel PCA (HALKP) to address the challenge [62]. In 2019, they also proposed a two-stage training subset selection framework for CVDP, called TSTSS, to overcome the conundrum [17]. Although the approaches proposed by Xu et al. are interesting [17,30,62], the contrasting technologies focus on the dataset processing methods ignoring the diversity of ordinary models (e.g., Random Forest and Adaboost), which confounds us about understanding the generalizability ability of these techniques. And, Yang et al. devised a new integration method based on [63]'s LTR for CVDP [64], which is called LTRE. Additionally, they also executed five ensemble approaches for demonstrating the effectiveness of LTRE, covering Random Forest, Bagging, Extra-Trees, gradient boosting, and Adaboost. The experimental results over 30 sets of cross-version data show that the performance of LTRE is similar to that of random forest, where other algorithms are worse than them. The principle of LTR is the sensitivity of defect model to fault percentage, in which LTRE achieves significantly larger mean FPA values than LTR at the 0.05 significance level, but without concerning concept drift. Noting that this research is deficient, namely that the technical route of the subject method is different from that of the contrast method, where LTRE, Random Forest, and remaining (i.e., Bagging, Extra-Trees, gradient boosting, and Adaboost) are implemented in Java, R, and Python, respectively. Prior studies have verified that there are discrepancies in the performance of defect prediction models implemented by toolkits (e.g., Weka written in Java [65], R [66], MATLAB [67], Scikit-learn

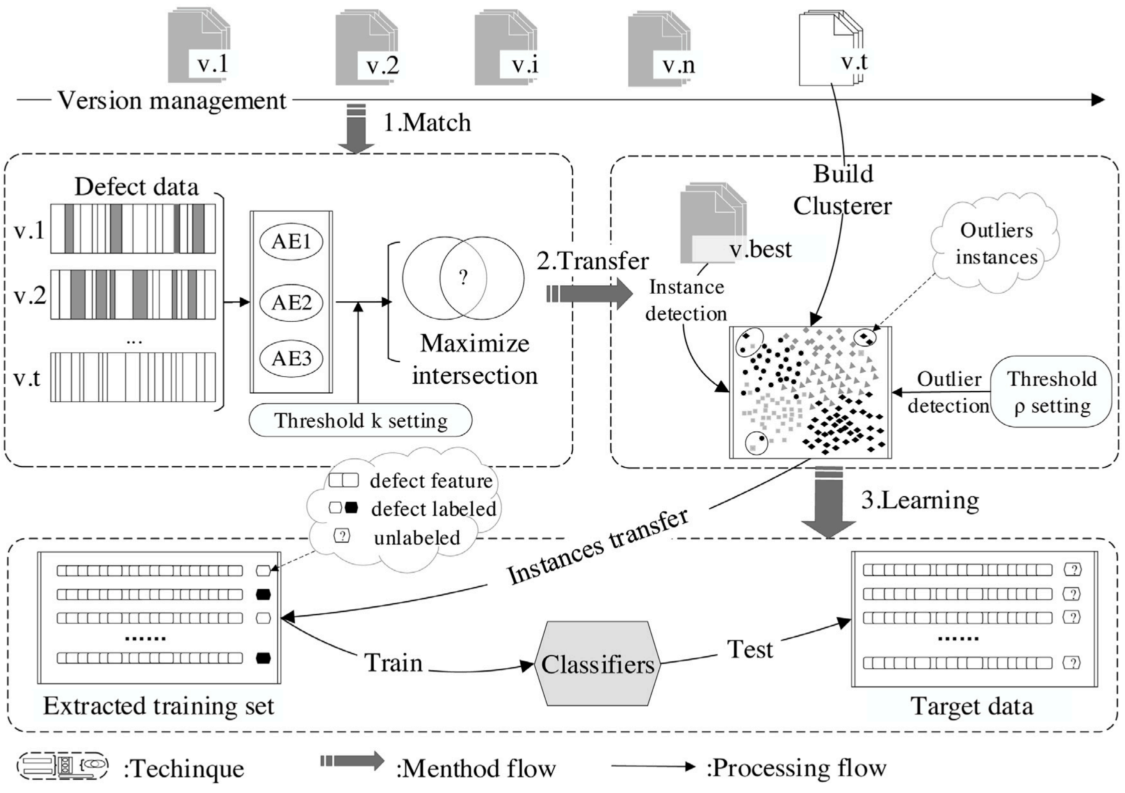

Fig. 1. Overview of our ST-TLF framework.

programmed by Python [68]), which can reduce the reliability of the comparison test [14]. In this study, we designed a framework for the CVDP to solve concept drift, which was validated in publicly available datasets by contrasting multiple defect prediction models using familiar metrics. These defect prediction models were implemented in Java by Weka, enhancing experimental reliability.

Also, Amasaki first proposed the usefulness of the cross-project approach in CVDP [29]. In 2018, he reviewed the performance of 17 cross-project approaches in CVDP [32]. And then, in 2020, he strengthened the study that investigated the performance of 24 cross-version methods in CVDP [33]. When the CPDP approaches apply to cross-version scenarios, these methods still employ the mixed data of source domain and target domain for data processing. Considering the similarity of defect data between versions, these CPDP approaches are pretty difficult to distinguish the information required by the target version. Our ST-TLF considers the distribution characteristics of defect data for the target version through a cluster analyzer.

Above all, CVDP has attracted much attention from practitioners lately. These researches enrich the practical experience in CVDP, which provides support for improving the accuracy of CVDP, but these investigations and studies are not perfect. To solve these problems, we designed a defect prediction framework ST-TLF for CVDP. This research endeavors to improve the deficiencies of the above studies in CVDP.

3. Methodology

As shown in Fig. 1, the transfer approach ST-TLF proposed by us for CVDP mainly consists of four components: data preparation, train dataset matching, instance transfer, and training classifiers, in which the performance of defect models we trained depends primarily on train dataset matching and instance transfer. In the diagram, on the version management line, v.1, v.2, ...,v.n respectively represents multiple historical versions of the project P, and v.t signals the upcoming version, which is also the target version. The transfer framework ST-TLF is a homogeneous transfer learning, which is a transfer learning where the feature space X and label space Y of the source domain D_s and the target domain D_t are isomorphous [49,50,69]. We describe the important part below.

3.1. Data preparation

For a project with multiple versions, the defect data of the historical versions have rich defect labels, while that of the upcoming version has no or few defect labels [17]. The deployment of our framework requires that upcoming versions provide partial, even rarely, defect-tagged data. During software development, these limited defect-labeled data that the new version contains tend to be extracted from defect tracking systems, e.g., Bugzilla, Trac, and MantisBT [9,10]. When executing defect prediction using ST-TLF, the feature variables of defect data between versions should be consistent, which is the prerequisite of homogeneous transfer learning.

3.2. Train dataset matching method

Prior studies considered the selection of training sets for the current version, which was not a deliberately designed approach but a conclusion from empirical statistics [30,44]. Empirical recommended methods assume that the last version has the most similar data distribution to the upcoming version for CVDP, which is not always accurate in a project with multi-version owing to the concept drift among versions [45]. We propose a novel strategy ST to identify and select the optimal training set for the target version in the same project. To the best of our knowledge, the ST is the first dataset matching technique designed for the CVDP. The goal of the technique is to avoid strong concept drift that may exist between training and test sets. Our ST evaluates the similarity of distribution for defect data through the similarity of the feature tendency of defect data between the historical and target versions. This method assumes that if the training set and test set come from the same data domain, their defect data have the same tendency of defect features, this is if the domain differences are larger (i.e., the greater the concept drift between the source and target domains), the less the similarity in their characteristic trends [49,50,69]. When considering the traditional approaches, our method breaks through the limitation of relying on experience and recovers the deficiency of previous research, which is described in detail below.

Algorithm 1 Train dataset matching method: ST

```
Input: The defect data of upcoming version $D_t = \{X_t, Y_t\}$; the historical defect data $D_s = \{D_{s1}, ..., D_{si}, ..., D_{sn}\}$; the defect data of the specific historical version $D_{si} = \{X_{si}, Y_{si}\}$; the number of historical versions n; Attribute evaluator $AE_j$; the number of the attribute evaluators j; the threshold parameter k.
Output: Training sets matched for target version $D_{best}$.
1: $F_m$ is defect feature where m is the number of variables, $F_{ranking} = \emptyset$.
2: Let the threshold $k < m$,
3: while $D_t, D_s (i = 1; i < n; i++)$ do
4:     while $1 \leqslant l \leqslant j$ do
5:         $F_{(ranking,l)} \leftarrow AE\{l\}$ for $D\{\bullet\}$
6:     end while
7:     $F_{ranking} = F_{(ranking,1)} \cap F_{(ranking,\bullet)} \cap F_{(ranking,l)}$ within the threshold k.
8: end while
9: while i=1;i<n;i++ do
10:     $FFR = \frac{|F_{ranking,D_t} \cap F_{ranking,D_{si}}|}{k}$
11: end while
12: Obtaining $\max_{D_s} D_t \cap D_{si}$, $D_{best} \leftarrow D_{si}$
13: Return the $D_{best}$.
```

The flowchart and pseudocode of our ST are provided by the second part of Fig. 1 and Algorithm 1, respectively. There is a project P with multiple versions, in which $D_t = \{X_t, Y_t\}$ is the defect dataset of the upcoming version $v.t$, and $D_s = \{D_{s1}, \ldots, D_{si}, \ldots, D_{sn}\}$ is the historical defect dataset where $D_{si} = \{X_{si}, Y_{si}\}$ is the defect dataset of the historical version $v.i$, X is the feature space, and Y is the label space. The purpose of our method is to select the D_{si} with the highest similarity to D_t, to satisfy the maximization of the following objective function (1):

$$\max_{D_s} D_t \cap D_{si} \tag{1}$$

Considering the comparability between alternative training sets in ST, we employ the variable evaluators to map complex multidimensional defect features to comparable spaces. We then quantify the feature overlap between them and the target domain, called the feature folding ratio (FFR). To put it simply, we set the threshold parameter k to compare the ratio of identical variables between cross-datasets within the cut-off range. The FFR is calculated by the following formula (2).

$$FFR = \frac{\#Numberofthesamevariables}{\#threshold\ (k)} * 100\% \tag{2}$$

Among, the higher the FFR, that is, the more the same variables within the threshold k, the higher the distribution similarity of the cross-versions of defect data in the feature space.

We implement the matching technology of the training set, ST, by the Wrapper methods (i.e., single variable evaluator) to explain the FFR between versions. To ensure the objectivity and accuracy of feature evaluation, we customized a strategy for full voting combined with multiple feature evaluation techniques to score the FFR. Namely, within the threshold K, determining the importance of a variable in datasets requires multiple methods to confirm together. Relevant studies have proved that evaluating the worth of an attribute by the information gain, the chi-squared statistic, and the information gain ratio concerning the class [17,34,70–72]. We also examined the validity of these evaluation techniques with massive experiments and confirmed the information gain, the chi-squared statistic, and the correlation coefficient as alternative evaluators. We have completed three technologies through weka's API, where the interface names for three methods are *i.e. InfoGainAttributeEval [73], ChiSquaredAttributeEval [74], CorrelationAttributeEval [75]*, respectively.

We first experiment to evaluate the importance of variables in both historical and new versions under three attribute evaluators $\{AE_1, AE_2, AE_3\}$ in the same project. And then, the variables within the threshold k in each version are confirmed by full voting. Finally, after the relatively objective variable ranking of all versions of the same project is obtained, the FFR between the historical and the new versions is calculated by formula (2). In an experiment, when FFR is the same between more than two historical versions and the new version, we depend on practical experience that recommends the defect data of the latest historical version on the timeline as a training set. Ant project, for example, has historical and new versions with the same 60-dimensional defect features. After ranking the importance of variables, the historical version Ant1.5 of the defect data has 35 attributes that are the same as the current version Ant1.7 within the threshold k = 50, so their FFR = 35/50 = 70%, and also like this to Ant1.6. As a result, we selected the defect data of Ant1.6 as the training set to predict the defects in Ant1.7. If the FFR is less than 70% for ant16 & ant17, we chose Ant1.3 as the training set.

3.3. Transfer instance

Algorithm 2 A sample-based non-inductive transfer learning: TL

```
Input: The test set $D_t = \{x_{t1}, x_{t2}, ..., x_{te}\}$; the training set $D_{best} = \{x_1, x_2, ..., x_g\}$; cluster analyzer EM; EM parameters $\theta$; estimated parameter $\hat{\theta}^{\{\bullet\}}$; learner$\pounds\{\bullet\}$; Clustering cluster $z_r$; Number of clusters r.
Output: $\pounds\{D_{best}\}$.
1: Initialize EM parameters $\theta$ with the K-means running 10 times;
2: repeat
3:     $x_{te}$ from $D_t$ to obtain $p(z_r|x_{te}, \hat{\theta}^e)$.
4:     Re-estimating parameters $\hat{\theta}^{e+1} \leftarrow \hat{\theta}^e$
5: until the log likelihood $lnp(D_t|\theta)$ converges (i.e., $\|\hat{\theta}^{e+1} - \hat{\theta}^e\| < \varepsilon$)
6: $EM(\theta, D_t) = z_1 \cup z_2 \cup ... \cup z_r$
7: $x_g$ from $D_{best}$ to compute $p(x_g \in z_r) = p(z_r|x_g, \theta)$
8: for $i = 1$; $i < g$; $i++$ do
9:     for $j = 1$; $j < r$; $j++$ do
10:         $P(Z|x_i) = \arg\max_{z_r}\{p(z_r|x_i, \theta)\}$
11:     end for
12:     if $P(Z|x_i) > \mu$ then
13:         Keep $x_i$ in $X_{best}$
14:     else
15:         remove $x_i$ in $D_{best}$
16:     end if
17: end for
18: Defect predictor $\pounds\{D_{best}\}$
```

How to filter labeled samples with similar distribution from the target domain data is one of the pivotal problems of sample-based transfer learning, which determines the performance of the defect model trained by it [69]. We propose a novel strategy TL that migrates valid defect instances from historical versions we selected to train defect models for the target version. The key to TL is instance-based clustering analysis. It can first build a cluster analyzer from the defect data of the upcoming version, then divide the distribution of defect instances in the new version into several clusters, and finally output the cluster membership (i.e., membership probability) for each instance in the defect data of the historical version. The maximum membership probability of historical instances in these clusters is obtained as the membership probability of historical instances in new versions. In the historical version (namely, source domains), outlier instances are removed and valid instances are transferred to train the defect models by setting thresholds on subjection degree that is obtained by cluster analysis. The cluster analysis model we selected is EM (Expectation & Maximization) which is an unsupervised learning algorithm [76].

EM can automatically determine the number of clusters through an internal cross-validation loop and output the membership probabilities of defect instances in these clusters [76]. Although prior studies have also favored employing EM algorithms for data processing, the scenarios and objects of application are different. For example, in two works by Herbold [77] and Herbold et al. [78], they mainly focus on CPDP scenarios, while our method is CVDP scenarios. The difference between the two scenarios is the source of the training set. That is, the complexity of the redundant information contained in the training data in the two scenarios is different. Additionally, the former was to process the feature vectors for defect prediction using EM cluster analysis, and the latter was to cluster the defect instances in the training and test sets, and then train defect models on these clusters to predict potential defects in the target project. Whereas, we employed EM methods to learn the defect distribution characteristics of the targeted version migrating more defect instances from the specific historical version for the targeted version.

The flowchart and pseudocode of our TL are provided by the third part of Fig. 1 and Algorithm 2, respectively. There is a upcoming version-t with a set of defect data $D_t = \{x_{t0}, x_{t2}, \dots, x_{te}\}$. We employ K-means running 10 times to initialize EM parameters θ. The EM cluster analyzer divides D_t into several clusters $Z = \{z_1, z_2, \dots, z_r\}$ in which $D_t = \cup_1^r z_r$ where each instance $x_{(\bullet)}$ has a corresponding $z_{(\bullet)}$. If there is an historical version-*best* with a set of defect data samples $D_{best} = \{x_0, x_1, \dots, x_g\}$, the posterior probability of a historical instance is $p(z_r|x_g, \theta)$ in cluster z_r from D_t, where θ is the EM parameters. Thus, the membership probability $P(Z|x_g)$ for historical instances x_g in the new version D_t is the largest posterior probability $p(z_r|x_g, \theta)$ in cluster z_r, which can be obtained by the following formula (3).

$$P(Z|x_g) = \arg\max_{z_r}\{p(z_r|x_g, \theta)\} \tag{3}$$

In practice, the defect data in the new version cannot directly learn the optimal parameter θ for $p(z_r|x_g, \theta)$ in $P(Z|x_g)$. To obtain the $p(z_r|x_g, \theta)$, the optimal parameter $\hat{\theta}$ can be found by maximizing the log-likelihood function, as shown in formula (4).

$$\hat{\theta} = \arg\max_{\theta} lnp(D_t|\theta) \tag{4}$$

The $lnp(D_t|\theta)$ can be transformed into the sum of the likelihood estimates of the clusters, as shown in formula (5) where $q(Z)$ is any efficient probability distribution of Z and $\sum_Z q(Z) = 1$; $p(D_t, Z|\theta)$ is the joint probability of D_t and Z under parameter θ; $KL(q \parallel p)$ (Kullback–Leibler divergence) is used to measure the gap between $q(Z)$ and posterior $p(Z|D_t, \theta)$.

$$\begin{aligned} lnp(D_t|\theta) &= (\sum_Z q(Z))lnp(D_t|\theta) \\ &= \sum_Z q(Z)ln\frac{p(D_t, Z|\theta)}{q(Z)} + KL(q \parallel p) \\ &= \ell(q, \theta) + KL(q \parallel p) \end{aligned} \tag{5}$$

Thus, $\ell(q, \theta) \leq lnp(D_t|\theta)$ always holds true, and if and only if $KL(q \parallel p) = 0$, this is $q(Z) = p(Z|D_t, \theta)$, $\ell(q, \theta) = lnp(D_t|\theta)$. In other words, we found the minimum boundary value for $lnp(D_t|\theta)$. During this process, iterate continuously after initializing the parameter θ. Suppose the process is in the t_{th} iteration, $\hat{q}(Z)$ follows

$$\hat{q}(Z) = p(Z|D_t, \theta^e) \tag{6}$$

and thus,

$$\begin{aligned} \ell(\hat{q}, \theta) &= \sum_Z \hat{q}(Z)ln\frac{p(D_t, Z|\theta)}{\hat{q}(Z)} \\ &= \sum_Z \hat{q}(Z)lnp(D_t, Z|\theta) - \hat{q}(Z)ln\hat{q}(Z) \\ &= \sum_Z p(Z|D_t, \theta^e)lnp(D_t, Z|\theta) - \hat{q}(Z)ln\hat{q}(Z) \end{aligned} \tag{7}$$

We mark the item containing the parameters θ and θ^e as $\Theta(\theta, \theta^e)$ that is the expectation of log-likelihood $lnp(D_t, Z|\theta)$ to posterior distribution $p(Z|D_t, \theta^e)$, which can be obtained by the following formula (8).

$$\begin{aligned} \Theta(\theta, \theta^e) &= \sum_Z p(Z|D_t, \theta^e)lnp(D_t, Z|\theta) \\ &= E_{z|D_t, \theta^e}[lnp(D_t, Z|\theta)] \end{aligned} \tag{8}$$

Maximizing this expectation results in new and better parameter estimation θ^{e+1}, which is formula (9).

$$\begin{aligned} \theta^{e+1} &= \arg\max_{\theta} \Theta(\theta, \theta^e) \\ &= \arg\max_{\theta} E_{z|D_t, \theta^e}[lnp(D_t, Z|\theta)] \end{aligned} \tag{9}$$

The iteration is continued until the log-likelihood converges to obtain the best model parameter estimation $\hat{\theta}$. Thus, the maximum membership probability of historical instances in the new version can be obtained, which is formula (10).

$$P(Z|x_g) = \arg\max_{z_r}\{p(z_r|x_g, \hat{\theta})\} \tag{10}$$

The valid instances from D_{best} are selected to construct a new training set D'_{best} for training the defect predictor by setting thresholds ρ on membership probability, in which X'_{best} has the following objectives:

$$\begin{aligned} &\max \quad D_t \cap D'_{best} \\ &s.t. \ x_i \ni D'_{best}, \ P(Z|x_i) \geq \rho. \end{aligned} \tag{11}$$

Using the above method, we have obtained a subset of data D'_{best} that is more similar to the target data D_t than the original training set D_{best}. In Jedit, for example, the current version 4.3 and the latest historical version 4.2. The unlabeled defect data of Jedit4.3 is divided into 8 clusters by the cluster analysis, i.e., $D_{jedit4.3} = z_1 \cup z_2 \cup \dots \cup z_8$. In jedit4.2, $P(Z|x_{39}) = arg\max_{z_r}\{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0\} = 1.0$, and $P(Z|x_{70}) = arg\max_{z_r} \{0.0, 0.0, 0.0241, 0.0, 0.9759, 0.0, 0.0, 0.0\} = 0.9759$, in which x_{39} and x_{70} are defect instances of classes constructed by static indicators of code, respectively. When considering $\rho = 0.98$, x_{39} is chosen to constitute the training set $D'_{jedit4.2}$, while x_{70} is discarded.

3.4. Classifier

For the defect prediction model, we illustrate two problems: class imbalance and parameter sensitivity. As defect data are usually class imbalanced, which impairs the performance of defect models, prior studies suggest using SMOTE techniques to overcome this problem in IVDP [31]. However, in this study, we cannot ignore the problem that class rebalancing technology can cause concept drift in the training set, which can interfere with the effectiveness of our framework. As a result, we are not concerned about the impact of class imbalance on the defect prediction model in this study. Meanwhile, we employ two classifiers: Random Forest and BayesNet. The former is not sensitive to parameters [14,57], which is a popular classifier in defect prediction outperforming other classifiers, and the latter is the best base learner recommended by prior work [33]. As shown in the fourth part of Fig. 1, to predict the defect distribution in the current version, the defect model for CVDP is obtained using the filtered defect data we selected.

The ST-TLF framework we designed is a typical non-inductive migration learning. The first part, ST, makes full use of the distribution of the feature space in the source and target domains according to the pre-conditions of transfer learning. The remaining work considers the similarity of defect data between historical and manual versions, focusing only on the distribution of data in the target version. Our ST-TLF improves the accuracy of the defect model by reducing concept drift through the two strategies mentioned above.

4. Experimental setup

To evaluate our framework ST-TLF, this section describes the datasets, scenario design, research questions, statistical test, and so on. Our experiments are all run on a 3.6 GHz Intel Core i7-7700 machine with 16 GB RAM.

4.1. Benchmark dataset

In selecting the studied datasets, we identify three criterias that need to be satisfied:

(1) Publicly and commonly available defect datasets;

To improve comparability and foster replication of our experiments, we train our defect prediction models using publicly and commonly available defect datasets.

(2) Projects with multiple versions;

In a project with multiple historical versions, the data drift between the new version and different historical versions is diverse. The problem we need to solve is concept drift between versions when using the defect data from the historical version to predict the defects of the coming version. Namely, we do not know which of these alternative versions is the best training dataset. It is our task to choose the best version from these historical versions for the new version. Therefore, we need projects that can perform the CVDP experiment and have at least two optional historical version defect data as a training set and defect data of the new version as a test set.

(3) Each object contains at least more instances than variables.

Many studies have pointed out that learning in small samples can produce unstable results [79,80] and proposed *Events Per Variable* (EPV) to assess the potential impact of the training set, suggesting that the EPV of the training set be greater than 10 [4,81]. EPV is the ratio of the number of occurrences of the least frequently occurring class of the dependent variable (i.e., the events) to the number of independent variables used to train the model (i.e., the variables) [82]. Since defect modules are rare in software products, training sets that meet $EPV > 10$ are scarce. Tantithamthavorn et al. statistics indicate that 77 percent of 101 datasets are at a high risk of producing inaccurate and unstable results (i.e., $EPV < 10$) [4]. To release this condition, we only employ each object that contains at least more instances than the number of characteristic variables. For example, when the feature vector provided in the training set contains a variable number of 50, we only select projects containing defect instances greater than 50.

We investigated several common and unified defect datasets, ultimately selected jureczko datasets from PROMISE [46], which contain different kinds of metric sets calculated with different tools. Therefore, even if the same metric name appears in two or more different datasets, we cannot be sure they mean the same metric. To eliminate this deficiency, Ferenc et al. used the free and open-source OpenStaticAnalyzer tool that analyzes systems source code to measure 60 static code indicators (size, complexity, coupling, cohesion, and inheritance) for each project [48]. We selected ten open source projects (a total of 37 versions) that meet the experimental conditions in the improved dataset. Their details include version, scale (i.e., the number of instances contained in the dataset, where each is composed of the attributes of code for a class or file), and defect ratio (i.e., the probability of defect instances in the dataset), as shown in the Table 1 [45]. For example, there are 351 instances in ant1.6, where the defect rate is 0.262, implying that the number of defect instances in ant1.6 is $351 * 0.262 = 92$.

Table 1
Introduction to basic information about project.

Version	Scale	Ratio	Version	Scale	Ratio	Version	Scale	Ratio
ant1.3	125	0.160	xalan2.4	723	0.152	lucene2.0	194	0.469
ant1.4	178	0.225	xalan2.5	803	0.482	lucene2.2	246	0.585
ant1.5	293	0.109	xalan2.6	885	0.464	lucene2.4	339	0.599
ant1.6	351	0.262	xalan2.7	909	0.988	velocity1.4	196	0.750
ant1.7	745	0.223	poi1.5	237	0.595	velocity1.5	213	0.662
jedit3.2	272	0.331	poi2.0	314	0.118	velocity1.6	228	0.342
jedit4.0	306	0.245	poi2.5	385	0.644	xerces1.2	440	0.161
jedit4.1	312	0.253	poi3.0	442	0.636	xerces1.3	453	0.152
jedit4.2	367	0.131	synapse1.0	157	0.102	xerces2.0	546	0.725
jedit4.3	492	0.022	synapse1.1	222	0.270			
camel1.0	339	0.038	synapse1.2	256	0.336			
camel1.2	590	0.366	log4j1.0	135	0.252			
camel1.4	841	0.172	log4j1.1	109	0.339			
camel1.6	927	0.203	log4j1.2	205	0.078			

4.2. Cross version scenario design

In this work, we have constructed many experiments of CVDP from these datasets and selected 17 eligible experiments from 10 projects through the following rules.

(1) Dataset Settings

In this study, we focus on the performance improvement of CVDP, so the training and test set of the experiment must be from the historical version and the new version of the same project, respectively.

(2) Training Set Settings

One of our tasks is to match the best training set for the new version from the defect data of the historical versions. Therefore, we need the CVDP experiment that has at least two optional defect datasets from historical versions as the training set. For example, Ant is a Java-based build tool, where the training set matched for the current version Ant1.7 was derived from defect data of the historical versions (i.e., Ant1.3, Ant1.4, Ant1.5, and Ant1.6).

4.3. Research questions

We empirically evaluate our devised method by answering the following three Research Questions (RQ).

RQ1: How different classifiers impact the effectiveness of our framework ST-TLF on CVDP performance?

This question investigates whether our ST-TLF can strengthen other classifiers, not just a random forest. In the experiment, four ensemble learners are implemented by Weka to learn defect models of CVDP, including Bagging, AdaBoost, Random Forest [83], and Vote [13,84], which have all been extended for handling streaming data with concept drift [22]. AdaBoost selected pruned C4.5 decision tree as the base learner, as do Bagging, and Vote's combined classifiers are decision tree, a logistic regression model with a ridge estimator, SVM, and Naive Bayes, and their combined strategies are majority voting. And, we have implemented two Neural network algorithms to participate in the survey, involving RBF Classifier and DL4jMLP. Moreover, we consider two independent algorithms most commonly used in defect prediction: Support Vector Machine (SVM) and BayesNet [47,85].

RQ2: Is our training set matching technique ST effective?

The goal of ST is to select a training set for the version at hand from the defect data of the prior versions. To evaluate the effectiveness of our ST, we selected a training set construction scenario for CVDP as a comparison strategy, i.e., Single Prior Version (SPV), which is project data of the (accessible) latest prior release recommended by Amasaki [33]. Although the rTDS is not specially customized for CVDP, considering the similarity of functions, we introduced it as a control method [55]. Since ST mainly depends on the importance of variables, we selected five prevalent attribute evaluators in defect prediction as control methods [34,70–72]. We implement these five methods via API provided by Weka, whose interface names are GainRatioAttributeEval (GR) [86], ReliefFAttributeEval (ReF) [87], ChiSquaredAttributeEval (CS) [74], CorrelationAttributeEval(CA), and InfoGainAttributeEval(IGA), respectively. Their detailed explanations are as follows:

GR: Evaluates the worth of an attribute by measuring the gain ratio concerning the class.

ReF: Evaluates the importance of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class.

CS: Evaluates the score of a variable by computing the value of the chi-squared statistic for the class.

CA: Evaluates the weight of a factor by computing the correlation (Pearson's) between it and the class [88].

IGA: Evaluates the worth of a feature by obtaining the information gain to the class.

Based on the work efficiency of the above variable evaluators, we also evaluated the combination of the CA and IGA, called CA+IGA.

RQ3: Compared with other methods, can our ST-TLF significantly improve the performance of CVDP?

As mentioned in Section 2, we investigated the related work of CVDP, of which our most relevant method is the TSTSS proposed by Xu et al. [17], followed by Amasaki's work [33]. We make great efforts to replicate the work of Xu et al. but the results were not satisfactory. We then selected the latter as our control method. According to Amasaki's work [33], in this work, we consider the approaches demonstrated in the best scenario SPV, which are better than the baseline approach they designed. The descriptions of these control methods are as follows:

P15-NB: The original Peters15 method with basic classifier NaiveBayes, where Peters15 extracts instances based on a multi-party data sharing algorithm called LACE2 [51].

P15-RF: The original Peters15 method with basic classifier random forest, in which the number of classification trees is 25 and the number of iterations is five [51].

P15-LR: The original Peters15 method with basic classifier Logistic [51].

K15-RF: The Kawata15 selects instances using DBSCAN, whose base classifier is random forest [54].

K15-LR: The original Kawata15 with Logistic [54].

P14-LR: Panichella14 is combining defect classifiers of different algorithms trained with results of classification of training data, which is known as CODEP, where Logistic regression (CODEP-LR) was used for combination [89].

P14-NB: The BayesNet (CODEP-BN) was employed for combination [89].

T09-NB: Turhan09 first transforms the metric data with the logarithm, then applies a relevancy filter to the available training data based on the k-Nearest Neighbor (NN) algorithm, whose basic classifier is NaiveBayes [90].

T09-RF: The original Turhan09 utilizes random forest as a base classifier [90].

W08-RF: Watanabe08 standardizes metrics values with averages of those from training data and testing data to improve the performance of CPDP defect models, in which random forest is the best base classifier [91].

We replicated ten defect models mentioned above by benchmark CrossPare implemented by Herbold et al. [16]. In this study, we used the same parameters as in previous studies [16,33]. Noting that among the algorithms mentioned above, P15-NB is the best in SVP scenarios, which is the control method we mainly focus on [33]. Besides, we employ two transfer learning approaches of CPDP as control methods: a two-phase transfer learning model (**TPTL**) [53] and a collective transfer learning for defect prediction (**CTDP**) [41]. When replicating CTDP, considering the limitations of cross-version defect data, we only employ the transfer component analysis (TAC) and N1 strategies for data processing, where N1 is the Min–Max normalization [41]. Also, we consider the random forest without instance transfer strategy as the most basic method.

4.4. Evaluation indicators

When predicting defective classes in a target version, a predictor may succeed (True Positive, TP) or fail (False Positive, FP) to predict a defect class, truly (True Negative, TN) or wrongly (False Negative, FN) identify a clean class [53]. Based on these four possible results, in this study, we use three comprehensive measures (i.e., F-measure, MCC, and AUC) to evaluate the performance of the models in the experiment [92–94]. F-measure is a trade-off between recall and precision which is defined as

$$F_\beta = \frac{(\beta^2+1)Precision \times Recall}{\beta^2 Precision + Recall} \tag{12}$$

The β is a bias parameter to measure the relative importance of recall and precision. In this work, we choose $\beta = 2$ (i.e., F2) that emphasizes more on the importance of recall, following the previous studies [17,70]. AUC (Area Under Curve) is a performance indicator to measure the quality of a learner by the relationship between TP and FP at different thresholds [93]. The closer the AUC is to 1, the stronger the model predicts; when it is equal to 0.5, the model loses its predictive value. Matthews correlation coefficient (MCC) is a fairly uniform index that considers four possible results [92,95]. Essentially, MCC is the correlation coefficient between the actual situation and the predicted results. The range of values is [−1,1], where +1 means perfect prediction; 0 means no better than a random prediction; and −1 means complete inconsistency between prediction and observation. The formula for calculating MCC is as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{13}$$

Meanwhile, we calculate the threshold-dependent performance measures (i.e., F-2 and MCC) using the default threshold parameter value of 0.5. The AUC is a threshold-independent performance indicator to measure a classifier's ability under varying thresholds, which is computed by measuring the area under the curve that plots the TP rate against the FP rate.

4.5. Parameter configuration

When implementing this framework, the configuration of two parameters needs to be concerned. The first parameter is the threshold parameter k in Eq. (2). This parameter is correlated with the feature dimension in the dataset, where it is required to be less than the feature dimension. Because the importance of feature variables is objective in mature datasets, the threshold k within a reasonable range according to the dimension of data characteristics has little influence on the accuracy of training set selection. In this work, the thresholds k of the ST, CA+IGA, and independent feature evaluators are 50, 50, and 20, respectively. Another parameter is the threshold ρ in Eq. (11). In this work, we determine the ρ based on the membership probability of the historical instance in the new version. More specifically, we set the ρ value from $D_\rho = \{0.0, 0.80, 0.85, 0.90, 0.95, 0.98\}$ to determine whether to transfer the historical instance for each cross-version pair. Note that we consider the 0 migration scenario (i.e., no instances removed) due to the similarity of defect data distribution between versions. According to the above description, the final training set constructed by transferring valid historical instances may be the original training set we selected.

4.6. Statistical test

To analyze and test the performance of these algorithms, Scott–Knott Effect Size Difference (ESD) test is selected in this paper [4,70, 96]. We used the sk_esd function implemented in the ScottKnott ESD R package to compare the identification performance of the approaches we examined. Additionally, to measure the effect size of test results, we compute the $Cohen's\,d$ effect size using the check Difference function implemented in the ScottKnott ESD R package. The d is derived from

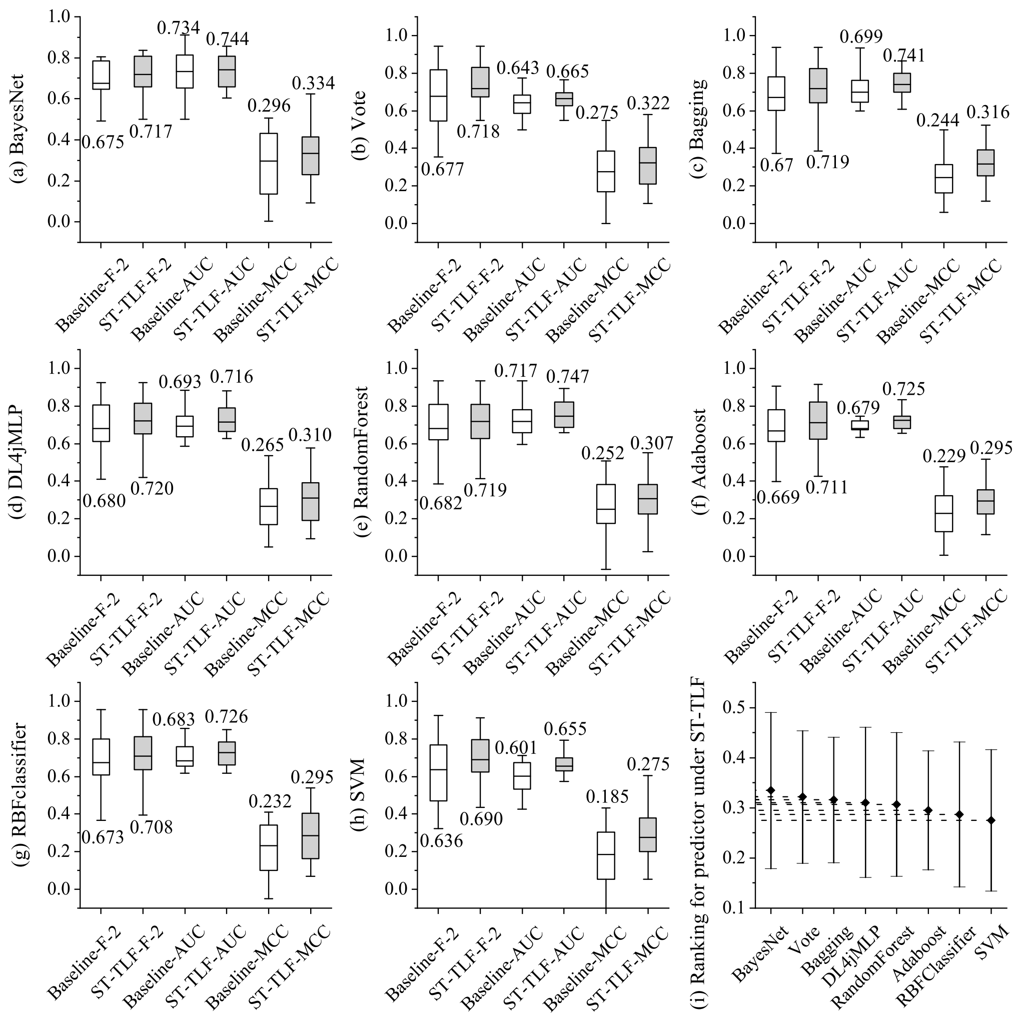

Fig. 2. The comparison results of 8 defect classifiers under our framework ST-TLF.

the distribution of the measured data, that is, the two means divided by the standard deviation of the two groups of subjects (i.e., $d = \frac{\bar{x_1} - \bar{x_2}}{s.d.}$). The magnitude is accessed using the thresholds that are provided by Cohen [97] as follows: negligible ($d \leq 0.2$), small ($0.2 < d \leq 0.5$), medium ($0.5 < d \leq 0.8$), and large ($d > 0.8$). If the $Cohen's\,d$ is larger, it indicates that the significance difference between the data is more obvious. Finally, according to the test results, the performance rankings of all the algorithms across all the experimental objects are obtained.

5. Analysis of experimental results

This section presents the experimental results and analyses according to different research questions.

5.1. Results for RQ1

Approach. In baseline experiments, we recommend the latest historical version of the defect data as a training set, which is the most popular method in previous work. To record the results of our ST, we consider two composite indicators AUC and MCC. Following previous studies [17,25,94], we chose the latter. We, therefore, only record the results corresponding to the best MCC value among the multiple sets of results for each cross-version pair.

Results. Fig. 2 not only provides the results for each defect model on 17 CVDP experiments using box plots, i.e., (a)–(h), but also presents the ranking results of these learners using the vertical bar chart, i.e., (i). The results of different algorithms are partitioned by blocks, each of which reveals the results of three indicators paired by the baseline and our framework ST-TLF in white and gray, respectively. The bottom and top of the boxes indicate the first and third quartiles respectively, while the solid horizontal line in a box indicates the median value in each performance distribution, and the black diamond in a box indicates the mean value in each performance distribution.

The following results can be observed from Fig. 2. Our framework ST-TLF with various classifiers can achieve satisfactory performance.

The F2 can reflect the accuracy of the classifier. The accuracy of all classifiers under our framework has been improved compared with the baseline. For instance, the F2 of BayesNet is improved by 6.2% under our framework, as shown in Fig. 2(a). The average F2 of the defect predictors we examined is elevated by 5.2% to 7.8% under our ST-TLF, of which the lower is the RBF classifier and the higher is the SVM classifier, as shown in Fig. 2(h) and (g) respectively.

When considering AUC and MCC, the ST-TLF framework improves the generalization ability of the defect model. The random forest has increased average AUC and MCC under our ST-TLF by 4.2% and 22%, respectively, as shown in Fig. 2(e). In the defect prediction models we examined, some defect model models (i.e., Bagging, Adaboost, SVM, and RBF Classifier) and the remaining models (i.e., BayesNet, Vote, and DL4jMLP) achieved higher and lower improvements than random forest, respectively. For instance, Adaboost, one of the most popular predictors in defect prediction, has obtained encouraging results under the framework ST-TLF, as shown in Fig. 2(f). The average AUC and MCC of the baseline experiments for Adaboost were 0.679 and

Table 2
Dataset matching results of 17 experiments for cross-version defect prediction.

NO.	Test	Train	MCC	SPV	rTDS	GR	Ref	CS	CA	IGA	CA+IGA	ST
1	ant1.5	**ant1.3**	**0.183**			⋆		⋆	⋆	⋆	⋆	⋆
		ant1.4	−0.068	⋆	⋆		⋆					
2	ant1.6	**ant1.3**	**0.321**			⋆		⋆	⋆	⋆		
		ant1.4	0.104				⋆					
		ant1.5	0.213	⋆	⋆						⋆	⋆
3	ant1.7	ant1.3	0.257				⋆					
		ant1.4	0.089									
		ant1.5	0.246									
		ant1.6	**0.427**	⋆	⋆	⋆		⋆	⋆	⋆	⋆	⋆
4	jedit4.1	jedit3.2	0.471									
		jedit4.0	**0.507**	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
5	jedit4.2	jedit3.2	0.397									
		jedit4.0	**0.456**		⋆							
		jedit4.1	0.443	⋆		⋆	⋆	⋆	⋆	⋆	⋆	⋆
6	jedit4.3	jedit3.2	0.049									
		jedit4.0	0.143			⋆						
		jedit4.1	0.147				⋆			⋆		
		jedit4.2	**0.176**	⋆	⋆			⋆	⋆		⋆	⋆
7	camel1.4	camel1.0	0.025									
		camel1.2	**0.393**	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
8	camel1.6	camel1.0	0.066			⋆					⋆	⋆
		camel1.2	**0.288**									
		camel1.4	0.269	⋆	⋆		⋆	⋆	⋆	⋆		
9	xalan2.6	xalan2.4	0.263									
		xalan2.5	**0.381**	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
10	xalan2.7	xalan2.4	0.042					⋆	⋆			
		xalan2.5	**0.113**							⋆	⋆	⋆
		xalan2.6	0.099	⋆	⋆	⋆						
11	poi2.5	**poi1.5**	**0.505**			⋆		⋆	⋆	⋆	⋆	⋆
		poi2.0	0.064	⋆	⋆		⋆					
12	poi3.0	**poi1.5**	**0.350**						⋆		⋆	⋆
		poi2.0	0.100			⋆		⋆		⋆		
		poi2.5	0.339	⋆	⋆		⋆					
13	synapse1.2	**synapse1.0**	**0.283**						⋆	⋆	⋆	⋆
		synapse1.1	0.252	⋆	⋆	⋆	⋆	⋆				
14	log4j1.2	**log4j1.0**	**0.115**		⋆	⋆					⋆	⋆
		log4j1.1	0.064	⋆			⋆	⋆	⋆	⋆		
15	lucene2.4	**lucene2.0**	**0.274**		⋆		⋆	⋆	⋆	⋆	⋆	⋆
		lucene2.2	0.211	⋆		⋆						
16	velocity1.6	velocity1.4	−0.078			⋆						
		velocity1.5	**0.329**	⋆	⋆		⋆	⋆	⋆	⋆	⋆	⋆
17	xerces2.0	xerces-1.2	−0.146					⋆	⋆	⋆	⋆	
		xerces-1.3	**0.190**	⋆	⋆	⋆	⋆					⋆
Accuracy				0.412	0.412	0.529	0.353	0.529	0.647	0.647	0.765	0.824
AVG.MCC				0.252	0.260	0.242	0.235	0.266	0.287	0.270	0.271	0.291

0.229, respectively, while those obtained for random forest within our framework ST-TLF were 0.725 and 0.295, respectively. Namely, by comparing the baseline, the average improvement of AUC and MCC is 6.8% and 22.4%, respectively. These results show that our method can enhance multiple learners to varying degrees, not just a random forest.

Fig. 2 (i) provides the ranking results of these learners under our ST-TLF, which are obtained by the ScottKnott ESD using MCC results from 17 experiments. Although SVM is the last ranked learner, its improvement under ST-TLF is more significant than other classifiers. As shown in Fig. 2(h), the average MCC of SVM is improved by 49.74%, in which the baseline experiment is 0.185 and the target experiment is 0.275. BayesNet is the first ranking defect predictor under the framework ST-TLF, which improved its average MCC by 12.8%. As we all know, previous studies have not only researched the validity of random forest for CVDP but also demonstrated its insensitivity to parameter adjustment. To foster the replication and ensure the comparability of our experiments, we choose BayesNet and random forest as our basic classifiers for the following experiments.

Answer to RQ1: Overall, our framework ST-TLF with different classifiers can work well in CVDP, rather than customized basic algorithms.

5.2. Results for RQ2

Approach. The essence of the research question is to verify the significance of the first part of our framework for CVDP. To examine the accuracy of ST, we present the MCC of the defect models trained on the defect data of all historical versions as an evaluation metric.

Results. Table 2 provides the results of dataset matching with 17 experiments for nine dataset matching approaches on CVDP, the last part of which presents accuracy and the average MCC, in which '⋆' indicates the mark of defect data selected by a specific method. The version number and MCC of the optimal training set for the current version in each experiment are marked in bold, as shown in Table 2.

According to the analysis of the results provided by each experiment, our ST is more efficient than other methods. The SPV, namely the empirically recommended method, has worked well in 7 out of 17 experiments, this is, the accuracy of SPV is only 41.2%, suggesting that it is not an optimal choice for the cross-version training set. Compared with this, our ST has achieved good results in 14 out of 17 experiments where two of the remaining three experiments (i.e., No. 2 and 5) have the same results as the SPV. In other words, ST works as well or better than the SPV in 16 of the 17 experiments. For example, in experiment No. 1, the MCC of the defect model obtained by the last version recommended by SPV as a training set is 0.068, while that of the ST is 0.183, meaning that the comprehensive performance of the defect model has increased by 369%. Experiment No. 9 has obtained similar results as experiment No. 1. Like SPV, the accuracy of other methods, including rTDS, is lower than that of our ST.

When considering the performance of defect models under the ST, MCC can provide a reference for us. The ST we proposed improves the performance of the defect prediction model by selecting a training set, while SPV, the most commonly used method in previous studies, was the worst, as did rDTS. Compared with SPV, the comprehensive performance of the defect model obtained through the training set recommended by ST is improved by 15.5%. The other approaches (e.g., GR, ReF) are to build a training set matching strategy based on the feature evaluator. The results of the training set recommended by these strategies are different. For example, the MCC of the defect model obtained from the CS recommended training set is 0.292 and that of the ReF is 0.245. This result is due to the single variable assessor taking into account one relevant factor between characteristics and response variables. For example, CS relies primarily on the chi-square statistic for each feature, and GR evaluates the value of an attribute by measuring the gain ratio relative to the category. Relatively, our ST and CA + IGA have better work efficiency, of which our ST is superior. For example, our method ST improves the MCC of the defect model by about 7.4% than CA + IGA by selecting a training set.

Our ST has another advantage: avoiding strong concept drift between training and test sets. There are many forms of concept drift, e.g., covariate drift, mean drift, and probability shift [19]. Class imbalance drift is one type of probability shift, which is also one of the basic data provided by defect datasets. Combining the defect ratio in Table 1, it is found that our ST can effectively avoid strong class imbalance drift. As everyone knows, the poor results in CVDP experiments are strongly related to class imbalance drift between versions. For example, in experiment No. 9, the defect ratios of the training set Poi2.0 and test set Poi2.5 are 0.118 and 0.644, respectively. And another benchmark experiment (i.e., Poi1.5 to Poi2.5), the class imbalance drift between the training set and the test set is not significant, and the defect ratio was 0.595 and 0.644, respectively. Their MCC is 0.064 and 0.505, respectively. Similar situations are found in the training and test sets of experiments No. 1, No. 10, and No. 13. This is one of the unavoidable problems in CVDP, which has been ignored in previous studies. In experiment No. 1, prior studies recommended ant1.4 as the training set for the ant1.5 learning defect prediction model, and the class imbalance drift problem destroys the reliability of the defect model. Through the similarity of abstract feature space, our ST recommends ant1.3 as the training set to learn defect models for ant1.5. ST has a similar performance in other experiments, including experiments No. 9, 10, and 16. The above experimental results verify that our ST can avoid strong class imbalance drift between two subjects. The principle of our ST to avoid strong class imbalance drift is to map information from a complex feature space to a comparable space, and then identify the defect data with the minimum concept drift as the target version.

The SPV is a typical empirical recommendation method, which lacks flexibility in combating class imbalance drift in CVDP. The rTDS uses the descriptive statistics of variables to select the defect data for the current version, but the general statistical description is not sensitive to the response variables. Other dataset matching methods based on the importance of feature variables introduce a single correlation factor that does not fully account for irregular changes in the feature space. And, our ST makes up for the shortcomings of the above-mentioned methods, which can avoid class imbalance drift between datasets in CVDP, improving the performance of defect prediction models.

Answer to RQ2: In conclusion, ST is more effective to select the training dataset for the current version for enhancing CVDP performance. Also, it can surmount the class imbalance drift between two subjects.

5.3. Results for RQ3

Approach. To answer this question, we first executed our ST-TLF, control approaches, and baseline method on 17 CVDP experiments, then implemented a statistical test at the significance level of 0.05 to reveal the effectiveness of our framework. When we perform ST-TLF, according to previous practical experience from the software engineering community, we hired two predictors that are not sensitive to a parameter as base classifiers: random forest and BayesNet, which are called ST-TLF1 and ST-TLF2, respectively. In this work, we selected two evaluation indicators, covering AUC and MCC.

Results. Fig. 3, which consists of two parts (i.e., Figs. 3.1 and 3.2), presents the AUC and MCC values for each evaluated experiment across the one baseline method, 12 control approaches, and ST-TLF framework. The results of the different experiments are partitioned by the block, where AUC and MCC in each block are represented by a diamond-marked connection diagram and independent histogram, respectively. To improve the readability of the results, we presented the results for the baseline in each experiment separately with dotted lines.

Fig. 4 visualizes the ranking results of the Scott–Knott ESD test for 15 methods under two performance indicators, where the horizontal dotted line represents their mean. And the dispersion degree of the results for a specific algorithm in multiple experiments relative to the mean value is displayed with a black vertical solid line to explain the generalization performance of the learning algorithm in multiple experiments.

The following results can be observed in Figs. 3 and 4:

First, as shown in Fig. 3, the MCC of ST-TLF1 is greater than and equal to that of random forest in 12 experiments and 4 experiments, respectively, excluding experiment No. 7. In experiment No. 11, for example, the MCC of our ST-TLF1 and RF are 0.505 and 0.064, respectively, indicating that ST-TLF improves 689.06% over baseline RF. When considering the control method, P15-NB is the best method recommended in Amasaki's research work [33]. The ST-TLF2 acquires better MCC than P15-NB in 10 CVDP experiments, especially Experiments Nos. 1, 2, and 11. The performance of our ST-TLF under AUC is similar to MCC.

Second, from Fig. 4, we find that ST-TLF1 and ST-TLF2 achieve the highest and second-best average AUC across the 17 experiments, with values of 0.744 and 0.747, respectively. For ST-TLF1, the average AUC increased by 4.2% and 3.9% compared with RF and P15-NB, respectively. Also, ST-TLF2 and ST-TLF1 obtain the best and next-best average MCC among all methods across the 17 experiments, where their average MCC values are 0.307 and 0.334, respectively. About ST-TLF2, the average AUC and MCC improved by 3.24% and 18.84% compared with P15-NB, respectively.

Third, as shown in Fig. 4, ST-TLF rankings are among the top in all models we examined in the Scott–Knott ESD test of all the evaluation metrics. The Cohen's d of AUC between ST-TLF1 and RF is 0.315 and its magnitude is S, namely, the ST-TLF1 outperforms RF with the Small magnitude of the effect size. The kernel classifiers of P15-NB and ST-TLF2 are similar. The results of the statistical test show that magnitude of the effect size between P15-NB and RF is negligible, and ST-TLF2 obtained a Small effect size compared to RF with a Cohen's d value of 0.295, illustrating ST-TLF2 exceeds baseline RF and comparative method P15-NB.

Fourth, the comprehensive indicator MCC reveals a more significant difference in these algorithms than the threshold-independent performance metric AUC. When comparing RF under the MCC, ST-TLF1 and

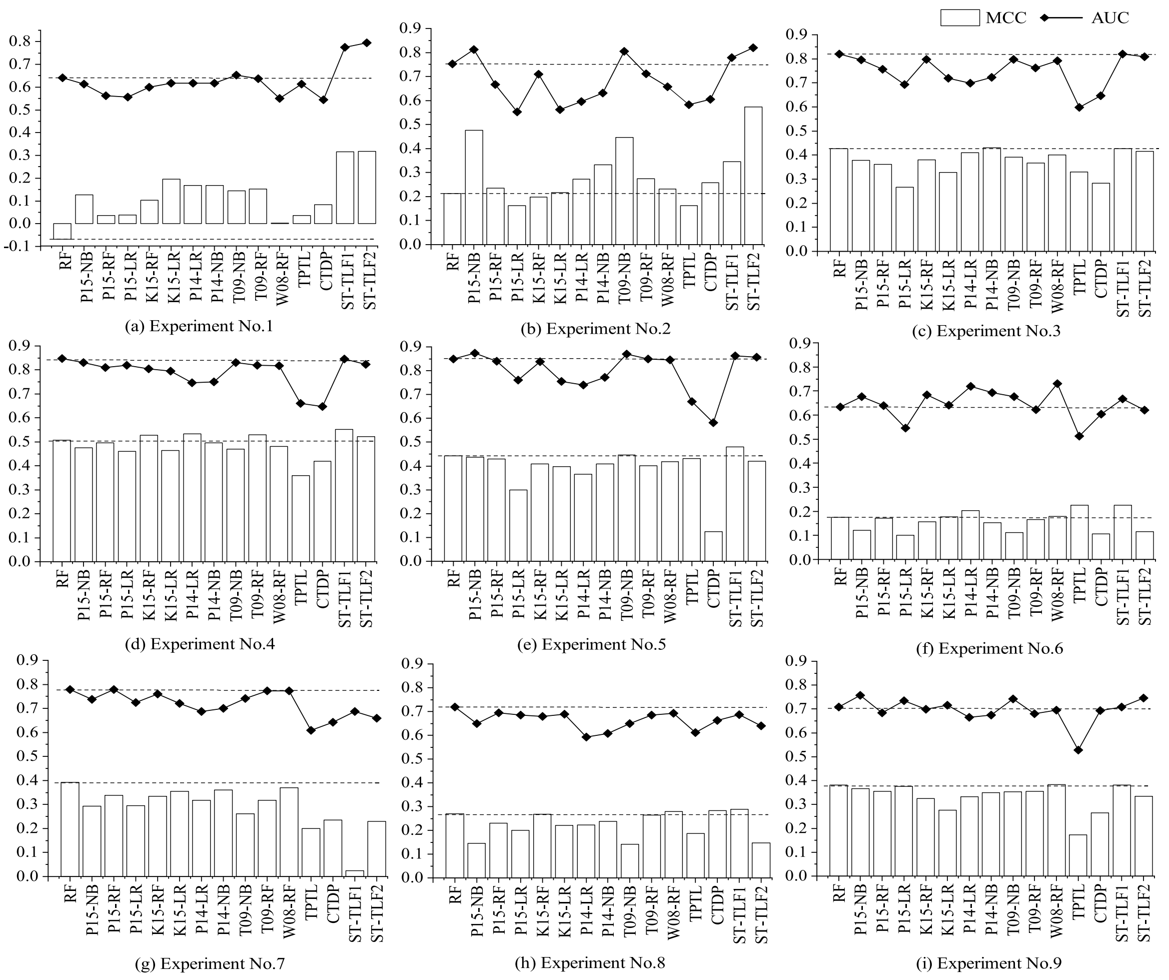

Fig. 3.1. Contrasting results between our ST-TLF, control methods, and baseline in 17 experiments.

ST-TLF2 obtain Small and Medium effect sizes, respectively, where their Cohen's d values are 0.365 and 0.526, respectively. Considering the improved efficiency of the defect model, although P15-NB and T09-NB also improve the performance of the baseline method, they do not exceed our method. The underlying classifiers for P15-RF and ST-TLF2 are RF. ST-TLF1 and P15-RF obtained higher and lower MCC than RF, with values of 0.307 and 0.236, respectively. In the statistical test, P14-RF negatively treats the performance of the defect model, ranking 11 after the baseline experiment, whereas our ST-TLF1 significantly improves it. More exactly, our method improved by 27.1% over P15-RF.

RF is the baseline method we selected, which has been shown to own outstanding accuracy for defect prediction in many studies. Among the methods we examined, multiple methods did not effectively improve the performance of RF, e.g., K15-RF, W08-RF, and P15-RF, while our method greatly improved its accuracy. The principles of ST-TLF and these comparison methods are different. P15-RF is to discard instances that do not belong to any group through cluster analysis on mixed defect data of source and target domain, and build the training set for the training defect model with the remaining instances in the source domain. Besides, the data processing cores of TPTL and CTDP are TCA, which minimizes the difference between domains by reducing the dimension of the source domain and the target domain. This data processing method may result in the loss of important information for defect prediction in the target domain. Other contrasting approaches have also built data filter methods on mixed data. These methods pay the same attention to the defect data of the source and target domain. When the scale of the source data is much larger than the target domain, some key knowledge of the target domain will be diluted or lost. Our method fully keeps a watchful eye on the specificity of the target domain, surmounting the shortcomings of the control method. ST-TLF reduces the concept drift between versions through two processes: first, selecting the best training set for the current version through the similarity of the abstract feature space between the source domain and target domain; then, filtering instances in the selected training set through clustering analysis of the defect data for the current version. Therefore, the elaborately training set built by ST-TLF is more refined and targeted. Also, ST-TLF is more flexible control the training set according to the practice conditions (i.e., specificity of the version at hand), which is a factor that is not taken into account by other control methods.

Answer to RQ3: In summary, ST-TLF is more effective for enhancing CVDP performance than other control methods and baseline.

6. Discussion

In a project with multiple versions, there is data drift between versions. To solve this problem, our study provides a defect prediction framework for CVDP, whose predictors can be potentially redeveloped. Our experimental results show that the instance transfer strategy improves the reliability of the defect prediction model compared with the original results. Improving CVDP performance is challenging, our results can provide a reference for researchers in practice. Besides, considering the similarity between CVDP and CPDP, we infer that this method may effectively improve the performance of defect models in CPDP, which is the focus of our future work.

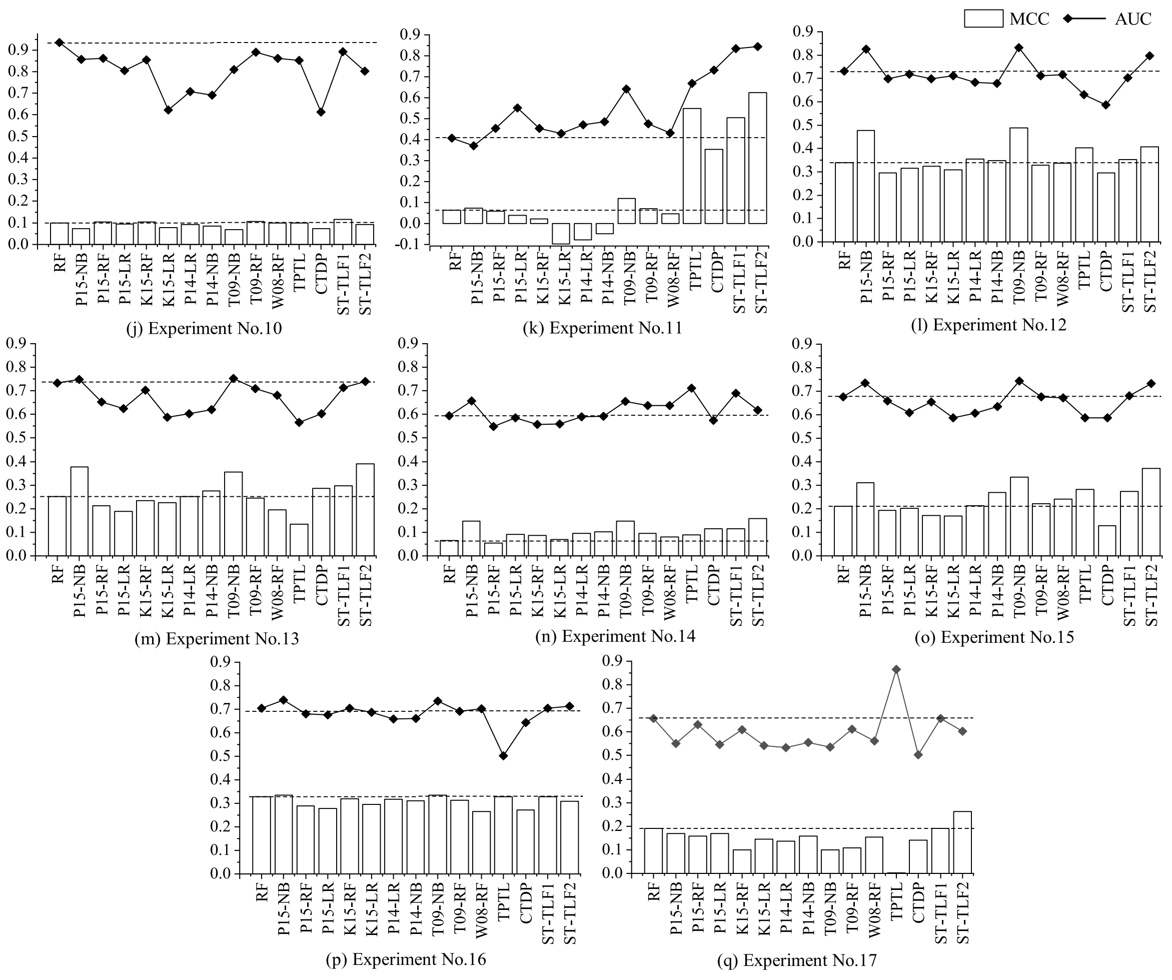

Fig. 3.2. Contrasting results between our ST-TLF, control methods, and baseline in 17 experiments.

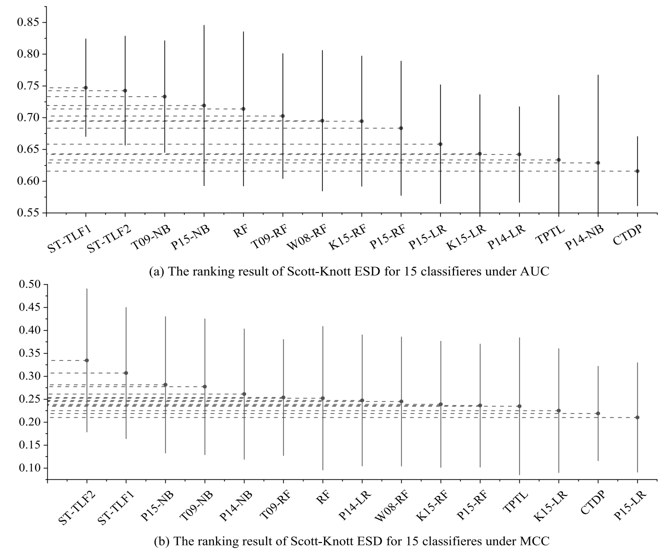

Fig. 4. The ranking result of Scott–Knott ESD for 15 classifiers under two performance indicators.

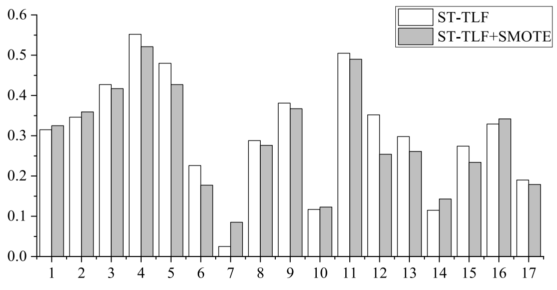

Fig. 5. Comparison results of SMOTE under our ST-TLF in 17 experiments.

Class imbalance is a familiar phenomenon in defect prediction, including CVDP [31]. Defect models are trained on class imbalanced datasets, which are highly susceptible to producing inaccurate prediction models [98]. We have shown its negative impact on CVDP performance through RQ2 in Section 5.2. Prior work raised concerns regarding the class imbalance in IVDP, and suggested SMOTE technique is beneficial for improving the reliability of the defect model [31]. Whether this conclusion applies to CVDP has not been proved. In this study, we examined the performance of SMOTE under ST-TLF in CVDP, as shown in Fig. 5. The results of 17 CVDP experiments show that the SMOTE technique has promoted the defect model in only six experiments, but not in the remaining experiments, indicating that SMOTE is not widespread to improve the accuracy of CVDP. In the future, we will continue to investigate the effect of class rebalancing technology on CVDP defect prediction.

7. Threats to validity

We now identify some potential threats to the validity in this paper, covering external validity, internal validity, and construct validity.

7.1. Internal validity

In this study, we performed some comparative experiments with defect classifiers. We have not adjusted the parameters of these models. For RQ2, we implemented eight classifiers. When performing CVDP experiments, we used the default parameters provided by Weka. Since the objective of this investigation is to test the generalization ability of our framework to multiple classifiers, adjusting the parameters for the classifier may reduce the reliability of our survey results. For RQ3, we replicated 12 defect models as control methods by benchmark CrossPare implemented by Herbold et al. [16]. These approaches are recommended by Amasaki, where they show better performance in CVDP than in the baseline [33]. Considering these models, we used parameters provided in previous studies to guarantee the comprehensibility of our contrast experiments. Also, we used random forest which is insensitive to parameters as the base classifier of our framework [96]. If the parameters for the defect model are adjusted in this investigation, the experimental results will change minimally.

We employed the Jureczko dataset [46] in the experiment, where the defect prediction features provided by Ferenc et al. [48]. The defect features consist of 60 different kinds, including source code and code duplication metrics. If other features (process metrics and semantic information) are applied, better or worse prediction results may be obtained, which have not been validated.

7.2. External validity

Our approach study relies on one defect prediction scenario (i.e., CVDP). However, there are a variety of defect prediction scenarios in the literature (e.g., CPDP and heterogeneous defect prediction), and the effectiveness of our method is unknown in these scenarios. In the future, we will work to achieve the effectiveness of our approach in CPDP.

The approach presented in this paper has only been validated by the Jureczko dataset [46,48]. The project for this dataset uses object-oriented encoding language (JAVA), and there are no other encoding languages (C, C++, Python, and so on). Replication studies on other projects that developed with C, C++, or Python may prove fruitful.

There are many research toolkits employed to implement machine learning, e.g., Weka [65], R [66], MATLAB [67], Scikit-learn [68]. In this study, we constructed baseline and control methods using Weka. When considering reproducing the study with other toolkits, results that slightly deviate from the current conclusions may be obtained.

Also, Forrest from the PROMISE library was a small-scale software product, and the number of defect instances (ratios) of the three versions were 6 (16.67%), 29 (17.24%), and 32 (6.25%), respectively. But in the experiment, we abandoned it because it did not meet condition 3. Actually, we performed complementary experiments for Forrest. Our framework ST-TLF and some control methods we replicated (e.g., Pater15) did not successfully predict potential defects in the target version. This may be the overfitting problem caused by small samples in machine learning, which is a typical few-shot learning problem [99]. Therefore, we do not ensure the generalization ability of our ST-TLF framework in small software products like Forrest. In future work, we will try our best to solve this problem.

7.3. Construct validity

In this study, two factors threaten the structural validity of our ST-TLF, involving class imbalance and performance evaluation. To evaluate the validity of ST-TLF, we selected three composite indicators, avoiding the structural threat of the study results. Class imbalance can reduce the accuracy and reliability of the defect model in CVDP, as shown in the result analysis. To objectively explain the performance of the method we provided, we have not taken relevant strategies to address the problem in CVDP. However, in Section 6, we initially discussed the impact of SMOTE on our framework through experiments, the results of which are not ideal. In future work, we will continue to focus on this issue.

8. Summary and prospect

Although CVDP is a practical scenario compared to IVDP and CPDP, some sticky problems may lead to the decreased accuracy and reliability of the defect models, where we are mainly concerned with concept drift in CVDP. To solve concept drift, we propose a novel transfer framework that consists of dataset matching techniques and instance-based transfer learning, which is called ST-TLF. For evaluating the performance of ST-TLF, we used the random forest and BayesNet as the base classifiers to investigate three research problems, involving the accuracy of matching training set, the generalization of ST-TLF in different classifiers, and the performance of ST-TLF in CVDP. The experimental results on ten open source projects (a total of 37 versions) from public datasets show that our ST-TLF can accurately match the best training set for the current version and effectively solve the problem of concept drift, improving the performance of CVDP. Also, in the comparative experiment, we selected 12 control methods, of which P15-NB is the proposed best defect model. Our framework combined with the underlying classifier performs better than P15-NB in CVDP.

Class imbalance is a common problem in defect prediction. In this study, we found that class imbalance drift between versions can impair the reliability of defect models in CVDP. Although the ST we proposed can avoid class imbalance drift by matching the best training for the current version, it does not fundamentally solve this problem. Prior work raised concerns regarding class imbalance and proposed solutions, e.g., SMOTE. In the discussion, we used the SMOTE method under the ST-TLF framework, whose results on 16 CVDP experiments show that the SMOTE does not significantly improve the performance of defect prediction in the CVDP. In future work, we will continue to explore concept drift to improve the accuracy of defect prediction in CVDP.

CRediT authorship contribution statement

Yanyang Zhao: Conceptualization, Data curation, Formal analysis, Methodology, Software, Validation, Investigation, Writing – original draft. **Yawen Wang:** Writing – review & editing, Funding acquisition, Project administration. **Yuwei Zhang:** Methodology, Validation, Writing – review & editing. **Dalin Zhang:** Methodology, Validation, Writing – review & editing. **Yunzhan Gong:** Software, Validation, Investigation. **Dahai Jin:** Software, Validation, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. U1736110) and supported by Guangxi Key Laboratory of Cryptography and Information Security, China (No. GCIS202103).

References

[1] M. Rodriguez, M. Piattini, C. Ebert, Software verification and validation technologies and tools, IEEE Softw. 36 (2) (2019) 13–24.
[2] H. Dezfuli, A. Benjamin, C. Everett, G. Maggio, R. Williams, NASA Risk Management Handbook, 2011.
[3] E. Tom, A. Aurum, R. Vidgen, An exploration of technical debt, J. Syst. Softw. 86 (6) (2013) 1498–1516.
[4] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Trans. Softw. Eng. 43 (1) (2017) 1–18.
[5] F. Wu, X.-Y. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, Y. Sun, Cross-project and within-project semisupervised software defect prediction: A unified approach, IEEE Trans. Reliab. 67 (2) (2018) 581–597.

[6] H.K. Dam, T. Tran, T. Pham, S.W. Ng, J. Grundy, A. Ghose, Automatic feature learning for predicting vulnerable software components, IEEE Trans. Softw. Eng. 47 (1) (2021) 67–85.

[7] E.V.d.P. Sobrinho, A. De Lucia, M.d.A. Maia, A systematic literature review on bad smells?5 w's: Which, when, what, who, where, IEEE Trans. Softw. Eng. 47 (1) (2021) 17–66.

[8] V. Abramova, F. Pires, J. Bernardino, Open source vs proprietary project management tools, in: A. Rocha, A.M. Correia, H. Adeli, L.P. Reis, M. Mendonça Teixeira (Eds.), New Advances in Information Systems and Technologies, Springer International Publishing, Cham, 2016, pp. 331–340.

[9] J.D. Blischak, E.R. Davenport, G. Wilson, A quick introduction to version control with git and GitHub, PLoS Comput. Biol. 12 (1) (2016).

[10] D. Russell, N. Patel, Increasing software engineering efficiency through defect tracking integration, in: 2006 International Conference on Software Engineering Advances (ICSEA'06), 2006, p. 5.

[11] S. Hosseini, B. Turhan, D. Gunarathna, A systematic literature review and meta-analysis on cross project defect prediction, IEEE Trans. Software Eng. 45 (2) (2019) 111–147.

[12] N. Li, M.J. Shepperd, Y. Guo, A systematic review of unsupervised learning techniques for software defect prediction, Inf. Softw. Technol. 122 (2020) 106287.

[13] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (6) (2012) 1276–1304.

[14] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, The impact of automated parameter optimization on defect prediction models, IEEE Trans. Software Eng. 45 (7) (2019) 683–711.

[15] J. Jiarpakdee, C. Tantithamthavorn, A.E. Hassan, The impact of correlated metrics on the interpretation of defect models, IEEE Trans. Software Eng. 47 (2) (2021) 320–331.

[16] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Trans. Software Eng. 44 (9) (2018) 811–833.

[17] Z. Xu, S. Li, X. Luo, J. Liu, T. Zhang, Y. Tang, J. Xu, P. Yuan, J. Keung, TSTSS: A two-stage training subset selection framework for cross version defect prediction, J. Syst. Softw. 154 (2019) 59–78.

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

[19] B. Turhan, On the dataset shift problem in software engineering prediction models, Empir. Softw. Eng. 17 (1) (2012) 62–74.

[20] F. Dong, J. Lu, K. Li, G. Zhang, Concept drift region identification via competence-based discrepancy distribution estimation, in: 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), 2017, pp. 1–7.

[21] M.A. Kabir, J.W. Keung, K.E. Bennin, M. Zhang, A drift propensity detection technique to improve the performance for cross-version software defect prediction, in: 44th IEEE Annual Computers, Software, and Applications Conference, 2020, pp. 882–891.

[22] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE Trans. Knowl. Data Eng. 31 (12) (2019) 2346–2363.

[23] S. Jayatilleke, R. Lai, A systematic review of requirements change management, Inf. Softw. Technol. 93 (2018) 163–185.

[24] J. Al Dallal, A. Abdin, Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review, IEEE Trans. Softw. Eng. 44 (1) (2018) 44–69.

[25] Z. Mahmood, D. Bowes, P.C.R. Lane, T. Hall, What is the impact of imbalance on software defect prediction performance? in: Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, in: PROMISE '15, Association for Computing Machinery, New York, NY, USA, 2015.

[26] K.E. Bennin, J. Keung, P. Phannachitta, A. Monden, S. Mensah, [Journal first] MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 2018, p. 699.

[27] E.A. Felix, S.P. Lee, Systematic literature review of preprocessing techniques for imbalanced data, IET Softw. 13 (6) (2019) 479–496.

[28] F. Thabtah, S. Hammoud, F. Kamalov, A. Gonsalves, Data imbalance in classification: Experimental evaluation, Inform. Sci. 513 (2020) 429–441.

[29] S. Amasaki, On applicability of cross-project defect prediction method for multi-versions projects, in: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, in: PROMISE, Association for Computing Machinery, New York, NY, USA, 2017, pp. 93–96.

[30] Z. Xu, S. Li, Y. Tang, X. Luo, T. Zhang, J. Liu, J. Xu, Cross version defect prediction with representative data via sparse subset selection, in: Proceedings of the 26th Conference on Program Comprehension, in: ICPC '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 132–143.

[31] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, IEEE Trans. Softw. Eng. 46 (11) (2020) 1200–1219.

[32] S. Amasaki, Cross-version defect prediction using cross-project defect prediction approaches: Does it work? in: Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, in: PROMISE'18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 32–41.

[33] S. Amasaki, Cross-version defect prediction: use historical data, cross-project data, or both? Empir. Softw. Eng. 25 (2) (2020) 1573–1595.

[34] J. Nam, S. Kim, Heterogeneous defect prediction, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 508–519, http://dx.doi.org/10.1145/2786805.2786814.

[35] J. Wang, Y. Chen, W. Feng, H. Yu, M. Huang, Q. Yang, Transfer learning with dynamic distribution adaptation, ACM Trans. Intell. Syst. Technol. 11 (1) (2020) 6:1–6:25.

[36] W. Pan, Q. Yang, W. Cai, Y. Chen, Q. Zhang, X. Peng, Z. Ming, Transfer to rank for heterogeneous one-class collaborative filtering, ACM Trans. Inf. Syst. 37 (1) (2019) 10:1–10:20.

[37] Y. Wei, Y. Zhang, J. Huang, Q. Yang, Transfer Learning via Learning to Transfer, in: 2018 Proceedings of the 35th International Conference on Machine Learning (PMLR), Vol. 80, 2018, pp. 5085–5094.

[38] M. Edmonds, X. Ma, S. Qi, Y. Zhu, H. Lu, S. Zhu, Theory-based causal transfer: Integrating instance-level induction and abstract-level structure learning, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, 2020, AAAI Press, 2020, pp. 1283–1291.

[39] Q. Chen, B. Xue, M. Zhang, Instance based transfer learning for genetic programming for symbolic regression, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 3006–3013.

[40] J. Nam, S.J. Pan, S. Kim, Transfer defect learning, in: 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 382–391, http://dx.doi.org/10.1109/ICSE.2013.6606584.

[41] J. Chen, K. Hu, Y. Yang, Y. Liu, Q. Xuan, Collective transfer learning for defect prediction, Neurocomputing 416 (2020) 103–116, http://dx.doi.org/10.1016/j.neucom.2018.12.091.

[42] Z. Xu, S. Pang, T. Zhang, X. Luo, J. Liu, Y. Tang, X. Yu, L. Xue, Cross project defect prediction via balanced distribution adaptation based transfer learning, J. Comput. Sci. Technol. 34 (5) (2019) 1039–1062, http://dx.doi.org/10.1007/s11390-019-1959-z.

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

[44] S. Shukla, T. Radhakrishnan, K. Muthukumaran, L.B.M. Neti, Multi-objective cross-version defect prediction, Soft Comput. 22 (6) (2018) 1959–1980.

[45] Y. Zhao, Y. Wang, D. Zhang, Y. Gong, Eliminating the high false-positive rate in defect prediction through Bayesnet with adjustable weight, Expert Syst. n/a (n/a) e12977, http://dx.doi.org/10.1111/exsy.12977.

[46] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, in: PROMISE '10, Association for Computing Machinery, New York, NY, USA, 2010.

[47] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.

[48] R. Ferenc, Z. Toth, G. Ladanyi, I. Siket, T. Gyimothy, A public unified bug dataset for java and its assessment regarding metrics and bug prediction, Softw. Qual. J. (2020).

[49] S.J. Pan, Transfer learning, in: C.C. Aggarwal (Ed.), Data Classification: Algorithms and Applications, CRC Press, 2014, pp. 537–570.

[50] Q. Yang, Y. Zhang, W. Dai, S.J. Pan, Introduction, in: Transfer Learning, Cambridge University Press, 2020, pp. 3–22, http://dx.doi.org/10.1017/9781139061773.003.

[51] F. Peters, T. Menzies, L. Layman, LACE2: Better privacy-preserving data sharing for cross project defect prediction, in: Proceedings of the 37th International Conference on Software Engineering - Volume 1, in: ICSE '15, IEEE Press, 2015, pp. 801–811.

[52] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inf. Softw. Technol. 54 (3) (2012) 248–256, http://dx.doi.org/10.1016/j.infsof.2011.09.007.

[53] C. Liu, D. Yang, X. Xia, M. Yan, X. Zhang, A two-phase transfer learning model for cross-project defect prediction, Inf. Softw. Technol. 107 (2019) 125–136, http://dx.doi.org/10.1016/j.infsof.2018.11.005.

[54] K. Kawata, S. Amasaki, T. Yokogawa, Improving relevancy filter methods for cross-project defect prediction, in: 2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence, 2015, pp. 2–7.

[55] P. He, B. Li, D. Zhang, Y. Ma, Simplification of training data for cross-project defect prediction, 2014, CoRR abs/1405.0773. arXiv:1405.0773.

[56] Y. Li, J. Su, X. Yang, Multi-objective vs. Single-objective approaches for software defect prediction, in: Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences, in: ICMSS 2018, Association for Computing Machinery, 2018, pp. 122–127.

[57] X. Yang, W. Wen, Ridge and lasso regression models for cross-version defect prediction, IEEE Trans. Reliab. 67 (3) (2018) 885–896.

[58] G.H. Golub, H.G. Wahba, Generalized cross-validation as a method for choosing a good ridge parameter, Technometrics 21 (2) (1979) 215–223.

[59] A.E. Hoerl, R.W. Kannard, K.F. Baldwin, Ridge regression: Some simulations, Commun. Stat. 4 (2) (1975) 105–123.

[60] J. Lawless, W. P, A simulation study of ridge and other regression estimators, Comm. Statist. Theory Methods 5 (1976) 307–323.

[61] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Comparing the effectiveness of several modeling methods for fault prediction, Empirical Softw. Engg. 15 (3) (2010) 277–295.

[62] Z. Xu, J. Liu, X. Luo, T. Zhang, Cross-version defect prediction via hybrid active learning with kernel principal component analysis, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018, pp. 209–220.

[63] X. Yang, K. Tang, X. Yao, A learning-to-rank approach to software defect prediction, IEEE Trans. Reliab. 64 (1) (2015) 234–246.

[64] X. Yang, X. Li, W. Wen, J. Su, An investigation of ensemble approaches to cross-version defect prediction, in: A. Perkusich (Ed.), The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019, KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019, pp. 437–556.

[65] M. Hall, E. Frank, G. Holmes, B. Pfahringer, I.H. Witten, The WEKA data mining software: An update, ACM SIGKDD Explor. Newsl. 11 (1) (2008) 10–18.

[66] R. Core Team, R: A language and environment for statistical computing, R Found. Stat. Comput. Vienna, Austria 14 (2009) 12–21.

[67] T. Mathworks, MATLAB, 1997, http://Www.Mathworks.Com/Products/Matlab/.

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[69] S.J. Pan, Y. Qiang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.

[70] Y. Zhang, D. Jin, Y. Xing, Y. Gong, Automated defect identification via path analysis-based features with transfer learning, J. Syst. Softw. 166 (2020) 110585.

[71] J. Nam, W. Fu, S. Kim, T. Menzies, L. Tan, Heterogeneous defect prediction, IEEE Trans. Softw. Eng. 44 (9) (2018) 874–896, http://dx.doi.org/10.1109/TSE.2017.2720603.

[72] H. Chen, X. Jing, Z. Li, D. Wu, Y. Peng, Z. Huang, An empirical study on heterogeneous defect prediction approaches, IEEE Trans. Softw. Eng. 47 (12) (2021) 2803–2822, http://dx.doi.org/10.1109/TSE.2020.2968520.

[73] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley, New-York, 2012.

[74] H. Liu, R. Setiono, Chi2: feature selection and discretization of numeric attributes, in: Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence, 1995, pp. 388–391.

[75] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (null) (2003) 1157–1182.

[76] K.D.J. Nagendra, J.V.R. Murthy, N.B. Venkateswarlu, Fast expectation maximization clustering algorithm, Int. J. Comput. Intell. Res. 8 (2) (2012) 71–94.

[77] S. Herbold, A. Trautsch, J. Grabowski, Global vs. Local models for cross-project defect prediction, Empirical Softw. Engg. 22 (4) (2017) 1866–1902, http://dx.doi.org/10.1007/s10664-016-9468-y.

[78] S. Herbold, Training data selection for cross-project defect prediction, in: Proceedings of the 9th International Conference on Predictive Models in Software Engineering, in: PROMISE '13, Association for Computing Machinery, New York, NY, USA, 2013, http://dx.doi.org/10.1145/2499393.2499395.

[79] Y. Jiang, J. Lin, B. Cukic, T. Menzies, Variance analysis in software fault prediction models, in: 2009 20th International Symposium on Software Reliability Engineering, 2009, pp. 99–108, http://dx.doi.org/10.1109/ISSRE.2009.13.

[80] T. Mende, Replication of defect prediction studies: Problems, pitfalls and recommendations, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, in: PROMISE '10, Association for Computing Machinery, New York, NY, USA, 2010, http://dx.doi.org/10.1145/1868328.1868336.

[81] P. Peduzzi, J. Concato, E. Kemper, T.R. Holford, A.R. Feinstein, A simulation study of the number of events per variable in logistic regression analysis, J. Clin. Epidemiol. 49 (12) (1996) 1373–1379, http://dx.doi.org/10.1016/S0895-4356(96)00236-3.

[82] P.C. Austin, E.W. Steyerberg, Events per variable (EPV) and the relative performance of different strategies for estimating the out-of-sample validity of logistic regression models, Stat. Methods Med. Res. 26 (2) (2017) 796–808, http://dx.doi.org/10.1177/0962280214558972.

[83] L. Breiman, Random forest, Mach. Learn. 45 (2001) 5–32.

[84] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Third ed., in: The Morgan Kaufmann Series in Data Management Systems (Third edition), Morgan Kaufmann, Boston, 2011, pp. 587–605.

[85] Z. Li, X. Jing, X. Zhu, Progress on approaches to software defect prediction, IET Softw. 12 (3) (2018) 161–175.

[86] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

[87] I. Kononenko, Estimating attributes: Analysis and extensions of RELIEF, in: Proceedings of the European Conference on Machine Learning on Machine Learning, in: ECML-94, Springer-Verlag, Berlin, Heidelberg, 1994, pp. 171–182.

[88] Nahler, Gerhard, Pearson correlation coefficient, Springer Vienna, 2009, p. 132, http://dx.doi.org/10.1007/978-3-211-89836-9, (Chapter 1025).

[89] A. Panichella, R. Oliveto, A.D. Lucia, Cross-project defect prediction models: L'Union fait la force, in: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), IEEE Computer Society, Los Alamitos, CA, USA, 2014, pp. 164–173.

[90] B. Turhan, T. Menzies, A.B. Bener, J.D. Stefano, On the relative value of cross-company and within-company data for defect prediction, Empir. Softw. Eng. 14 (5) (2009) 540–578.

[91] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a fault prediction model to allow inter languagereuse, in: Proceedings International Conference on Software Engineering, 2008, pp. 19–24.

[92] J. Yao, M. Shepperd, Assessing software defection prediction performance: Why using the matthews correlation coefficient matters, in: Proceedings of the Evaluation and Assessment in Software Engineering, in: EASE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 120–129.

[93] P.A. Flach, The geometry of ROC space: Understanding machine learning metrics through ROC isometrics, in: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, in: ICML'03, AAAI Press, 2003, pp. 194–201.

[94] J. Yao, M. Shepperd, The impact of using biased performance metrics on software defect prediction research, Inf. Softw. Technol. 139 (C) (2021).

[95] A. Luque, A. Carrasco, A. Mart?, A. de las Heras, The impact of class imbalance in classification performance metrics based on the binary confusion matrix, Pattern Recognit. 91 (2019) 216–231.

[96] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, Automated parameter optimization of classification techniques for defect prediction models, in: Proceedings of the 38th International Conference on Software Engineering, in: ICSE '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 321–332.

[97] J. Cohen, A power primer, Psychol. Bull. 112 (1) (1992) 155–159, http://dx.doi.org/10.1037//0033-2909.112.1.155.

[98] K.E. Bennin, J.W. Keung, A. Monden, On the relative value of data resampling approaches for software defect prediction, Empirical Softw. Engg. 24 (2) (2019) 602–636.

[99] Y. Wang, Q. Yao, J.T. Kwok, L.M. Ni, Generalizing from a few examples: A survey on few-shot learning, ACM Comput. Surv. 53 (3) (2020) 63:1–63:34, http://dx.doi.org/10.1145/3386252.