On the Multiple Sources and Privacy Preservation Issues for Heterogeneous Defect Prediction

Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, Hongyu Zhang, Baowen Xu, and Shi Ying

Abstract—Heterogeneous defect prediction (HDP) refers to predicting defect-proneness of software modules in a target project using heterogeneous metric data from other projects. Existing HDP methods mainly focus on predicting target instances with single source. In practice, there exist plenty of external projects. Multiple sources can generally provide more information than a single project. Therefore, it is meaningful to investigate whether the HDP performance can be improved by employing multiple sources. However, a precondition of conducting HDP is that the external sources are available. Due to privacy concerns, most companies are not willing to share their data. To facilitate data sharing, it is essential to study how to protect the privacy of data owners before they release their data. In this paper, we study the above two issues in HDP. Specifically, to utilize multiple sources effectively, we propose a multi-source selection based manifold discriminant alignment (MSMDA) approach. To protect the privacy of data owners, a sparse representation based double obfuscation algorithm is designed and applied to HDP. Through a case study of 28 projects, our results show that MSMDA can achieve better performance than a range of baseline methods. The improvement is 3.4-15.3 percent in *g-measure* and 3.0-19.1 percent in *AUC*.

Index Terms—Heterogeneous defect prediction, multiple sources, privacy preservation, utility, source selection, manifold discriminant alignment

1 Introduction

Software defect prediction (SDP) is one of the most popular research topics in software engineering. SDP aims to predict defect-prone software modules (instances) before the defects are discovered, therefore it can be used to better prioritize software quality assurance efforts [1], [2], [3], [4], [5], [6]. In recent years, many machine learning based methods [7], [8], [9], [10], [11], [12], [13], [14] and various software metrics [15], [16], [17], [18], [19], [20], [21] have been presented and applied to SDP. For example, Kim et al. [8] used a machine learning classifier for predicting latent software bugs. Jing et al. [10] introduced dictionary learning technique into defect prediction. Recently, Lee et al. [22] presented the micro interaction metrics, which leverage developer interaction information for SDP. Existing defect prediction methods can be mainly divided into two categories according to their application scenario: within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). WPDP focuses on predicting defects of new software instances in the same project by building defect predictor using sufficient historical defect data [23], [24], [25], [26], [27]. However, in practice, such kind of historical data may be very limited for some projects [28], [29], [30], [31], which hinders the application of WPDP.

CPDP refers to building defect prediction models for software modules in a target project using historical data collected from other existing projects (i.e., source projects) [28], [29], [30]. In recent years, several CPDP methods have been developed. It has been found that defect predictors built from cross-project data can be just as effective as those built from within-project data [32], [33], [34], [35], [36], [37], [38]. Existing CPDP methods require that the instances of source and target projects have the same metrics (i.e., the metric sets should be identical between projects). However, in many cases, there could be very few common metrics between source and target projects, and it is challenging to find a source project that has the common metric sets as the target project's. For example, there are 37 metrics in many NASA datasets [39], and 61 metrics in AEEEM datasets [40]. However, there exists only one common metric between NASA and AEEEM datasets, which is Line of Code (LOC) metric. In this case, the existing CPDP methods cannot obtain satisfactory prediction results,

- *Z. Li, B. Xu, and S. Ying are with the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, China. E-mail: lzq115@163.com, bwxu@nju.edu.cn, yingshi@whu.edu.cn.*
- *X.-Y. Jing is with the the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, China and with the College of Automation, Nanjing University of Posts and Telecommunications, Nanjing 210023, China. E-mail: jingxy_2000@126.com.*
- *X. Zhu is with the School of Computer and Information Engineering, Henan University, Kaifeng 475001, China and with the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, China. E-mail: whuzxk@whu.edu.cn.*
- *H. Zhang is with the School of Electrical Engineering and Computing, University of Newcastle, Callaghan, NSW 2308, Australia. E-mail: hongyu.zhang@newcastle.edu.au.*

Manuscript received 28 Feb. 2017; revised 18 Sept. 2017; accepted 20 Nov. 2017. Date of publication 5 Dec. 2017; date of current version 24 Apr. 2019. (Corresponding author: Xiao-Yuan Jing.)
Recommended for acceptance by A. E. Hassan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2780222

TABLE 1
Terminologies Used in the Paper

Terminology	Description
SDP	software defect prediction
WPDP	within-project defect prediction
CPDP	cross-project defect prediction
HDP	heterogeneous defect prediction (CPDP using different metric sets)
Single source	a single project which provides metric data for training
Multiple sources	multiple projects which provide heterogeneous metric data for training
Source data	software instances (modules) in a source project
Target data	software instances (modules) in a target project
Training target data	the limited amount of labeled software instance (modules) from a target project
Test target data	the to-be-predicted software instances (modules) from a target project

because some informative metrics necessary for building a good prediction model may not exist in the common metrics across projects [41], [42].

Recently, heterogeneous defect prediction (HDP) models [41], [42], [43], [44] are proposed to predict defects across projects with heterogeneous metric sets (i.e., source and target projects have different metric sets). For example, Jing et al. [41] presented a transfer CCA+ method by utilizing unified metric representation and CCA-based transfer learning technique for HDP. Nam and Kim [42] employed metric selection and metric matching techniques to predict defects across projects with heterogeneous metric sets. These methods have achieved encouraging prediction results.

In this paper, we address issues associated with HDP. To clarify some specific terms used in the paper, we provide the description of terminologies in Table 1.

1.1 Motivation

Although existing heterogeneous defect prediction methods [41], [42], [43], [44] have achieved promising results, we have identified the following two issues:

(1) *The Multiple Sources Issue*. Existing HDP methods mainly focus on predicting software instances in a target project based on metric data collected from a single project. In practice, there exist plenty of external projects from other companies. Multiple source projects can generally provide more information than a single project. Intuitively, predicting software instances in a target project with multiple sources may bring better performance. However, since the heterogeneity exists not only between the source and target but also among multiple sources, it is challenging to learn a high-quality defect prediction model from multiple heterogeneous sources. Therefore, it is meaningful to investigate whether and how the performance of HDP can be improved by employing the multiple source projects.

(2) *The Privacy Preservation Issue*. A precondition to conduct HDP learning experiments with multiple source projects is that these projects can be obtained from other companies. In practice, due to the privacy concerns, most companies are not willing to share their data. In particular, some sensitive attributes (e.g., Line of Code) could exist in software defect data, from which some business-sensitive information can be inferred. For example, with Line of Code, the effort of developing corresponding project can be roughly estimated. To facilitate data sharing, it is essential to protect the privacy of data owners before they release their data. However, existing HDP methods do not consider the privacy preservation issue.

1.2 Contribution

In this paper, we aim to investigate the feasibility of using multiple source projects for HDP and provide an effective privacy preservation algorithm for data owners. The main contributions of our work can be summarized as follows:

(1) We propose a multi-source heterogeneous defect prediction approach in Section 3.2.2, named multi-source selection based manifold discriminant alignment (MSMDA). MSMDA can incrementally select distribution-similar source projects from many available sources for a given target project. By fully exploiting the label information of multiple sources and a limited amount of training target data, MSMDA can transform the target project and the well-selected multiple sources into a discriminative common subspace, where the target and sources have similar data distributions and favorable classification ability.

(2) We design a privacy preservation algorithm in Section 3.1.2, named sparse representation based double obfuscation (SRDO). With SRDO, the privacy of original data can be protected effectively, and the data utility can be well maintained at the same time.

(3) We conduct extensive and large-scale experiments on 28 public projects from five groups including NASA, SOFTLAB, ReLink, AEEEM and PROMISE to evaluate our proposed approach. Experimental results demonstrate that the proposed multi-source heterogeneous defect prediction approach can obtain better HDP prediction performance than several state-of-the-art methods, and the designed privacy preservation algorithm can provide better privacy and higher utility.

1.3 Research Questions

In this paper, we mainly address the following two research questions:

- *RQ1:* How to design a privacy preservation algorithm that can provide favorable privacy and utility for the privacy needs of HDP?
- *RQ2:* How effective is using multiple heterogeneous source projects to improve the HDP prediction performance?

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes the research methodology. The experimental setups are given in Section 4 and the corresponding experimental results are reported in Section 5. In Section 6, we provide a discussion about the proposed approach and some threats to validity. The conclusions are drawn in Section 7.

2 Related Work

In this section, we briefly review cross-project defect prediction methods, heterogeneous defect prediction methods, and privacy-preserving methods in software defect prediction.

2.1 CPDP Methods with Common Metric Sets

Cross-project defect prediction refers to predicting defect-proneness of a software instance in a target project by using prediction model trained from historical data of other source projects [28], [29], [30]. In recent years, we have witnessed a lot of interest in developing new CPDP methods. Most of existing CPDP methods focus on designing effective machine learning algorithms with strong generalization ability for building the defect predictors [45], [46], [47]. For example, Ma et al. [32] presented a CPDP method named transfer Naive Bayes, which adapts naive Bayes with weighting training data. Nam et al. [34] applied transfer component analysis (TCA) to CPDP, which is a feature-based transfer learning method. Moreover, they extended TCA to TCA+ by using the normalization techniques to preprocess data, which exhibits good performance for defect prediction. Recently, Chen et al. [36] presented a double transfer boosting method for CPDP. Canfora et al. [48] developed a multi-objective genetic algorithm for CPDP to train a logistic regression model, which adopts the cost-effectiveness multi-objective predictor. Ryu et al. [37] investigated the applicability of the class imbalance learning under CPDP setting and designed a boosting based model named value-cognitive boosting with support vector machine. Jing et al. [49] provided an improved subclass discriminant analysis based defect prediction framework for both within-project and cross-project class-imbalance learning problems. Xia et al. [38] presented a HYbrid moDel Reconstruction Approach (HYDRA) for CPDP, which includes two phases: genetic algorithm phase and ensemble learning phase. These two phases create a massive composition of classifiers. To bridge the gap between programs' semantic information and defect prediction features, Wang et al. [50] applied a powerful deep learning algorithm to learn semantic representation of programs automatically from the source code for both WPDP and CPDP. From the perspective of clustering, Zhang et al. [51] presented to apply a connectivity-based unsupervised classifier for cross-project prediction that is based on spectral clustering (SC).

Different from the methods that focus on designing effective machine learning algorithms, some CPDP methods aim to find the best suitable training data for the instances in a target [33], [52], [53]. Zimmermann et al. [28] performed 622 cross-project predictions and showed that careful selection of training data and the characteristics of data and process play important roles in successful CPDP. Turhan et al. [29] presented a nearest-neighbor filter (NN-filter) method to select training data from other projects for within-project data based on distribution similarity. Later, they introduced a mixed model [35] for CPDP, which uses both the within-project and cross-project data as filter (NN-filter) method to select training data. He et al. [30] investigated CPDP based on training data selection by carrying out three experiments using 34 data sets. They showed that the prediction results were related to the distributional attributes of datasets, which is useful for training data selection.

Additionally, there are some other CPDP methods [54], [55]. Based on the different evaluation measures, Rahman et al. [56] introduced the measure called area under the cost effectiveness curve (AUCEC) to investigate the feasibility of CPDP and drew an optimistic conclusion that CPDP is no worse than within-project prediction in terms of AUCEC. Predicting the code changes are likely to introduce defects, which is referred to change-level defect prediction. Based on this context, Kamei et al. [57] empirically evaluated the performance of Just-In-Time (JIT) cross-projects models. Zhang et al. [58] designed a universal defect prediction model to predict an individual project, which is built from an entire set including the target project and other projects.

Existing CPDP methods learn the prediction model by using the common metrics contained in the source and target projects. In practice, the size of common metric set across source and target projects may be very small. In this case, these methods cannot obtain desirable prediction results.

2.2 HDP Methods

Recently, a few HDP methods have been presented [41], [42], [43], [44]. Since the metrics except common metrics might have favorable discriminant ability, Jing et al. [41] presented a HDP method, namely CCA+, which utilizes unified metric representation (UMR) and CCA-based transfer learning technique. Specifically, the UMR consists of three types of metrics, including the common metrics of the source and target projects, source-project-specific metrics, and target-project-specific metrics. By learning a pair of projective transformations under which the correlation of the source and target project is maximized, CCA+ can make the data distribution of target be similar to that of source. Experiments on public projects indicate that CCA+ can achieve good prediction results.

At the same time, Nam and Kim [42] presented another solution for HDP. They first employed the metric selection technique to remove redundant and irrelevant metrics for source project. Then, they matched up (e.g., Kolmogorov-Smirnov test based matching, KSAnalyzer) the metrics of source and target projects based on metric similarity such as distribution or correlation. After these processes, they finally arrived at a matched source and target metric sets. With the obtained metric sets, they built HDP model to predict labels of the instances in a target. They found that about 68 percent of HDP predictions outperform or are comparable to WPDP predictions with statistical significance.

Additionally, He et al. [43] presented CPDP-IFS to address the problem of heterogeneous metric sets (Imbalanced Feature Sets) in CPDP. They used distribution characteristics vectors [30] of each instance as new metrics to enable defect prediction. Recently, Cheng et al. [44] presented a cost-sensitive correlation transfer support vector machine (CCT-SVM) method to deal with the class imbalance problem in CPDP based on CCA+ [41]. They take different misclassification costs into consideration to incorporate the SVM classifier for defective and non-defective classes.

Existing HDP methods [41], [42], [43], [44] only use a single source project to predict target project, yet they did not investigate the feasibility of using multiple source projects to conduct defect prediction. In addition, they did not consider the privacy-preserving problem for the purpose of HDP.

2.3 Privacy-Preserving Methods in Software Defect Prediction

Privacy preservation issue has been investigated in many software engineering applications, e.g., software defect

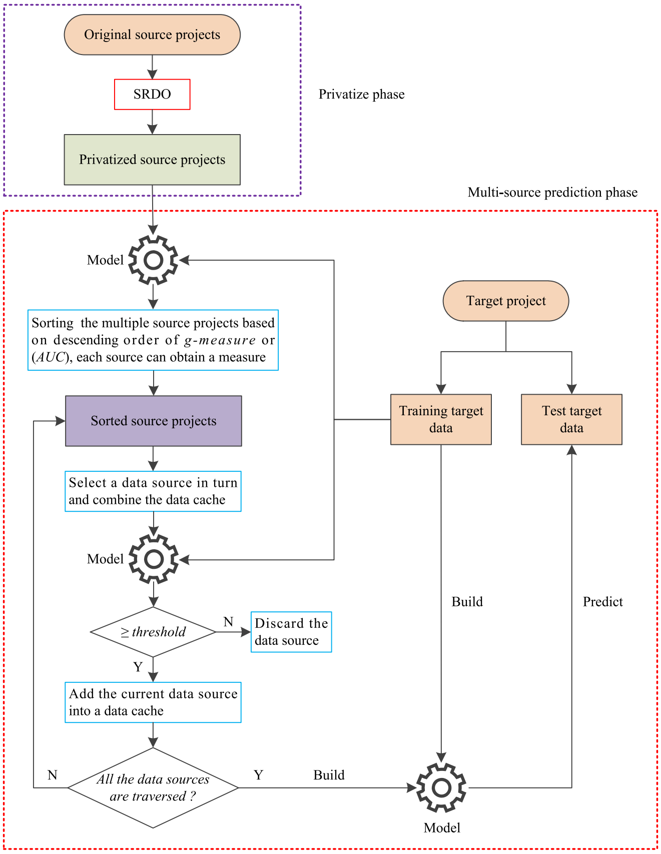

Original source projects
SRDO
Privatized source projects
Privatize phase
Multi-source prediction phase
Model
Sorting the multiple source projects based on descending order of g-measure or (AUC), each source can obtain a measure
Target project
Training target data
Test target data
Sorted source projects
Select a data source in turn and combine the data cache
Model
Build
Predict
≥ threshold
N
Discard the data source
Y
Add the current data source into a data cache
N
All the data sources are traversed ?
Y
Build
Model

Fig. 1. Overall architecture of our approach for HDP. The multiple heterogeneous source projects from other companies are first privatized. Then, a multi-source selection based manifold discriminant alignment approach is used to build HDP model. Finally, based on the trained HDP model, the test data in the target can be predicted.

prediction [59], [60], [61], software testing and debugging [62], [63], [64], [65], [66], software effort estimation [67], and so on. In this section, we briefly review the most related privacy works in software defect prediction.

Peters and Menzies [59] presented a privacy algorithm MORPH, which is a data transformation method via perturbation to create synthetic data. With MORPH, the defect data set owners can privatize their data prior to release. However, the authors found that, sometimes, the MORPHed data exhibited worse performance than the original data. Subsequently, they presented the CLIFF+MORPH (or LACE1[1]) algorithm [60], which combines CLIFF and MORPH. CLIFF gets rid of irrelevant instances thereby increasing the distances between the remaining instances, and MORPH is able to move the remaining instances further and create new synthetic instances that do not cross class boundaries. To solve the privacy preservation problem in multi-party scenario,[2] LACE2 [61] is presented. LACE2 combines CLIFF, LeaF [68] and MORPH algorithms together, where LeaF is an online and incremental clustering technique. With LeaF, data owner incrementally add data to a private cache and contribute "interesting" data that are not similar to the current content of the private cache. However, LACE2 [61] is based on the assumptions that each data owner must provide data with the same metrics for pooling into a private cache. So LACE2 was only suitable for CPDP with the same metric sets.

3 Research Methodology

In this section, we first introduce the designed privacy-preserving algorithm, and then describe the proposed multi-source heterogeneous defect prediction approach. Fig. 1 illustrates the overall architecture of our research method. More technical details will be introduced in the following sections.

3.1 The Proposed Privacy Preservation Algorithm

The privacy and utility are two important aspects that need to be considered when designing a privacy-preserving data sharing algorithm. Here, we first review related techniques CLIFF and MORPH, and then propose a new privacy-preserving algorithm, i.e., sparse representation based double obfuscation.

3.1.1 CLIFF and MORPH

CLIFF [60] is an instance selector that return a subset of relevant instances and deletes irrelevant instances. It assumes that tables of training data can be divided into *classes*, different rows might be labeled accordingly defective or defect-free. The procedure of CLIFF is formally stated as follows:

(1) For each column of data, find the *power* of each attribute subrange; i.e., how frequently that subrange appears in one class more than any other. Computing the *power* of a subrange is based on equal frequency binning (EFB) with 10 bins to each attribute in the dataset.
(2) Finding the product of the *powers* for each row.
(3) Remove the less powerful rows of each class, keeping the most powerful rows. For example, selecting 40 percent of the top ranked rows.

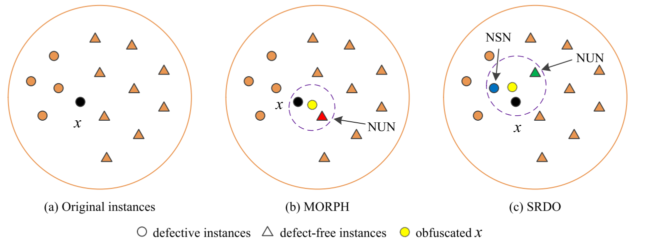
The result is a reduced dataset with fewer rows. In theory, this reduced dataset is less susceptible to privacy breaches.

MORPH [60] is an instance mutator. It changes the numeric attribute values of each row by replacing these original values with *MORPHed* values. Let $x \in data$ be the original instance to be changed, y the resulting MORPHed instance, and $z \in data$ the *nearest unlike neighbor* (NUN) of x. NUN is the nearest neighbor of x whose class label is different from $x's$ class label (i.e., between-class instance) and its distance is calculated by using the euclidean distance. Base on the above description, the MORPHed instances are created by applying following Equation to each attribute value of the instance

$$y = x \pm (x - z) * r, \tag{1}$$

where r is a random number to control the obfuscation degree for x. A small r value means the boundary is closer to the original instance, while a large r value means the boundary is farther away from the original instance. The value of random number r ranges 0.15 and 0.35. In [60] and [61], this range of values produced privatized data candidates with high privacy and accurate defect prediction. More details about CLIFF and MORPH can be found in [60].

[1] large-scale assurance confidentiality environment.
[2] The multi-party scenario means the multiple data owners privatized their data together.



(a) Original instances (b) MORPH (c) SRDO

(d) Partial CM1 defect data

x	1	2	3	4	5	6	7	8	9
75	31	24	41	17	14	14	47	18	40
32	7	7	9	8	5	2	14	6	21
23	5	0	22	14	12	0	39	1	22
59	27	22	27	15	10	12	47	17	30
16	4	2	14	2	4	2	0	1	10
74	14	23	19	11	6	6	55	15	37
Distance	87.481	89.000	75.756	100.608	108.351	110.359	**46.098**	99.393	59.942
Coefficient	0	1.021	0	0	0	0	0	0.010	**1.257**

Fig. 2. Illustration of the MORPH and SRDO algorithms. The black circle represents the original defective instance x to be changed, the yellow circle represents the resulting obfuscated instance, the red triangle denotes the selected NUN (i.e., defect-free instance, number 7) of MORPH, the green triangle (i.e., defect-free instance, number 9) and the blue circle separately denote the selected NUN and NSN of SRDO.

3.1.2 *Sparse Representation Based Double Obfuscation*

For the original instance x, MORPH only employed the NUN from between-class instances to obfuscate the data. Although the MORPHed value can ensure the privatized dataset private enough, it does not always guarantee that an instance doesn't move across the boundary between the original instance and instances of another class, which means that it may be difficult to keep the MORPHed value close enough to the original to maintain the utility of the dataset. Intuitively, for x, if the between-class and within-class instances are used simultaneously for obfuscation, it's more likely that the privatized value will not move across the boundary between the original instance and instances of another class, i.e., the data distribution can be better maintained.

As described in Section 3.1.1, MORPH adopted the euclidean distance based neighbor selector to select the NUN instances. However, one weakness of the basic euclidean distance selector is that if one of the input attributes has a relatively large range, then it can overpower the other attributes, leading to that the "true" NUN instance may be missed [69]. That is, the euclidean distance is sensitive to the noise data, while the software defect data usually contains noise [39], [70], [71] in practice. Hence, we should design a nearest neighbor selector to select the "true" neighbor instances.

Sparse representation, a recently developed technique, arouses much interest from researchers due to its effectiveness and robustness [72]. The idea of sparse representation is that an instance can be efficiently represented or coded by a sparse linear combination of a few base instances called dictionary atoms [72]. It has powerful representation ability and owns robustness to noise data. Given an instance $b \in \mathcal{R}^d$, its sparse representation can be seen as finding a coefficient vector η by solving the following optimization problem:

$$\min_{\eta \geq 0} \|b - B\eta\|_2^2 + \mu\|\eta\|_1, \tag{2}$$

where $B \in \mathcal{R}^{d \times n}$ denotes the dictionary atoms, which usually are constructed by using the training instances. μ is a small parameter which controls the sparsity of the solution. The coefficient vector η is sparse and only has a few nonzero values, where the instance corresponding to the maximal nonzero value is most similar to b. The $l1\_ls$ technique[3] can effectively solve the above optimization problem [73]. Based on this, we can employ the sparse representation technique to select the nearest neighbor.

Therefore, to maintain the distribution of privatized data more effectively, we propose a sparse representation based double obfuscation algorithm for HDP. SRDO first selects the NUN from the between-class instances and the *nearest similar neighbor* (NSN) from the within-class instances with sparse representation technique, and then uses both the selected NUN and NSN as disturbances. With SRDO, the distribution of original data can be well kept in the privatized data, such that the utility of data can be better maintained. Here, NSN is the nearest neighbor of x in the same class. Fig. 2 illustrates the steps involved in the MORPH and SRDO algorithms. In the figure, we give an example to show the difference between MORPH and SRDO. For MORPH, it only selects the NUN (the red triangle in Fig. 2b) from the between-class instances and uses the euclidean distance to find the nearest neighbors. For SRDO, it simultaneously selects the NUN (the green triangle in Fig. 2c) from the between-class instances and NSN (the blue circle in Fig. 2c) from the within-class instances as well as uses the sparse representation coefficient to find the nearest neighbors. The yellow circles in Figs. 2b and 2c separately denote the possible obfuscated instances of MORPH and SRDO.

Algorithm 1. The SRDO Algorithm

```
Input: The original projects $\{S_1, S_2, \ldots, S_K\}$;
Output: The privatized projects $\{S_1, S_2, \ldots, S_K\}$.
1: Each data owner separately uses CLIFF to select the subset of
   their data that best represents the target classes, here, keeping
   40 percent of the original data [60]. Only the data
   selected by CLIFF are used in SRDO;
2: The CLIFFed data are then obfuscated with SRDO by using
   Eq. (3);
3: The privatized data after SRDO can be added to a public
   data repository for sharing.
```

The detailed obfuscating steps of SRDO are as follows. The CLIFF technique is used to select informative instances that best predict for the target class and remove uninformative instances. Then, for each CLIFFed instance x, we separately select a NSN z_w from the within-class instances and a NUN z_b from the between-class instances by using sparse representation based the nearest neighbor selector. More specifically, to select the NSN z_w, we treat the x as a sparse linear combination of all the within-class instances (except for the x itself) by using Eq. (2), and then select the within-class instance corresponding to the maximal coefficient value as z_w. Similarly, NUN z_b is selected by treating x as a sparse linear combination of all the between-class instances by using Eq. (2). The original instance can be obfuscated as follows:

$$y = x + (x - z_w) * r_1 - sign(r_1)(x - z_b) * |r_2|, \tag{3}$$

where y is the resulting SRDOed instance, r_1 and r_2 are the random numbers to control the obfuscation degree for x,

[3] https://web.stanford.edu/~boyd/l1_ls/

$sign(\cdot)$ is the signum function, 1 if the corresponding value is greater than zero, -1 if the corresponding value is less than zero, and 0 otherwise, $|\cdot|$ is the absolute value function. Experimental results in [60] and [61] have shown that the random number r can produce privatized data candidates with high privacy and accurate defect prediction in the range of 0.15 to 0.35. Therefore, we use this range of values as our experimental parameters for the random numbers r_1 and r_2 in Eq. (3). In future work, we will investigate the optimal values for these experimental parameters. The detailed procedure of SRDO is summarized in Algorithm 1.

3.2 Multi-Source Heterogeneous Defect Prediction

To sufficiently use multiple source projects from other companies, we propose a multi-source heterogeneous defect prediction approach for HDP, namely, multi-source selection based manifold discriminant alignment.

3.2.1 Heterogeneous Domain Adaptation for HDP

In HDP, the main challenge is how to overcome the difference in data distributions between source and target projects when learning a generalized model [41], [42]. In fact, HDP can be viewed as a specific case of transfer learning, which aims to transfer the knowledge learned from a source project to a target project. Heterogeneous domain adaptation [74] is an advanced transfer learning technique [75], [76]. It aims to handle different distributions between heterogeneous source and target domains, and builds a classifier using the data in source domain that will perform well in target domain. Recently, manifold alignment (MA) based heterogeneous domain adaptation methods have achieved good results in several fields, such as text categorization [77] and remote sensing image classification [78].

In this paper, we intend to introduce the MA technique into HDP. However, there exist some shortcomings in existing MA methods. (1) They do not take the source selection problem into consideration such that they may fail to identify the right sources to use for adaptation. A key problem of transferring defect prediction knowledge from one project to another is the differences between training data and test data. Intuitively, the target project should learn from source projects with similar properties, or learn from training data with similar distributions. Since the multiple source projects are usually collected from different development environments or application fields, it is possible that there exist large data distribution differences among these source projects and a target project. Considering that the data distribution differences may bring adverse influence to the follow-up prediction, it is important to choose proper source projects for a given target project. As mentioned above, an intuitive idea for heterogeneous cross-project defect prediction is to select proper source projects that are similar to the target project. (2) Existing MA methods cannot guarantee that the discrepancy between unlabeled instances in the target project and source projects is reduced effectively. Exploiting unlabeled instances in the target project during adaptation may be beneficial for domain adaptation [74]. Therefore, we propose a novel multi-source selection based manifold discriminant alignment approach for HDP.

Algorithm 2. Pseudocode for MSMDA

```
Input: Source projects $\{S_1, S_2, \ldots, S_K\}$, Target project $T$: $\{X_t^l, X_t^u\}$, where $X_t^l$ denotes training target data and $X_t^u$ denotes test target data ;
Output: The predicted label of $X_t^u$.
for $i = 1$ to $K$ do
    Train MDA model with $\{S_i, X_t^l\}$;
    Use MDA to predict $X_t^l$;
    // evaluate performance, e.g., g-measure or AUC
    Compute the evaluation measure $\theta_i$;
end for
$\hat{\theta} = sort(\theta)$; // descending order
$\hat{S} = sort(S)$; // adjust the order of sources based on $\theta$
$threshold = \hat{\theta}(1)$;
$cache = \{\hat{S}_1\}$;
for $i = 2$ to $K$ do
    Train MDA model with $\{cache, \hat{S}_i, X_t^l\}$ ;
    Use MDA to predict $X_t^l$;
    Compute the evaluation measure $\theta_i$;
    if $\theta_i \geq threshold$ then
        $cache = cache \cup \hat{S}_i$; // add $\hat{S}_i$ into $cache$
    else
        discard $\hat{S}_i$;
    end if
end for
Use $cache$ and $X_t^l$ to train MDA model and predict $X_t^u$.
```

3.2.2 Multi-Source Selection Based Manifold Discriminant Alignment

In this section, we describe the proposed MSMDA approach. Essentially, MSMDA is an incremental optimization process by adding a data source at each iteration. The key idea of MSMDA is as follows:

(1) For a given target project, we first use each source project from multiple available sources together with a limited amount of training target data (i.e., 10 percent of labeled data from the target project) to build the MDA prediction model, and then evaluate its performance on the training target data according to the *g-measure* (or *AUC*).

(2) We can obtain a permutation by sorting the source projects based on descending order of *g-measure* to the target project;

(3) Based on the obtained permutation, we select the best *g-measure* value which corresponds to the best source project as a threshold and add this source project into a data cache.

(4) Then, we select a data source in turn together with the training target data to build the MDA prediction model and compute the *g-measure*. If the current *g-measure* value smaller than the threshold, we discard this data source. Otherwise, we add this data source into the data cache.

(5) The process is complete when all the sources have been traversed.

(6) Finally, we use selected sources in the data cache to conduct defect prediction.

Algorithm 2 provides pseudocode for the proposed MSMDA approach. The core component of MSMDA is the

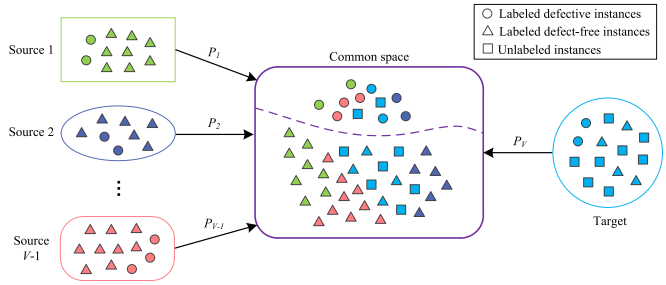


Fig. 3. Illustration of the MDA algorithm. Different colors represent the instances in different projects. "○" represents the labeled defective instances, "△" the labeled defect-free instances and "□" the unlabeled instances.

MDA algorithm. The rest of this section offers further detail on MDA.

(1) Inputs for MDA. Assuming that there are V heterogeneous input domains (here, a domain corresponding to a project), each contains c classes (in this paper, $c = 2$, that is defective class and defect-free class). The project X_v can be viewed as a matrix of size $m_v \times n_v$. Let $M = \sum_v m_v$ and $N = \sum_v n_v$ be the total metrics and total instances of all projects respectively. Specifically, *the first $V-1$ domains represent the source projects and the Vth domain represents the target project*. Each source contains a data set $X_s = x_s^i|_{i=1}^{n_s}$ and a label set $Y_s = y_s^i|_{i=1}^{n_s}$, $s = 1, 2, \ldots, V-1$, where x_s^i denotes the ith instance in X_s, y_s^i is the corresponding label and n_s is the number of instances in X_s. The target consists of a small amount of labeled instances $X_t^l = x_t^i|_{i=1}^{n_l}$ (the corresponding label set is denoted by $Y_t^l = y_t^i|_{i=1}^{n_l}$), and plenty of unlabeled instances $X_t^u = x_t^i|_{i=n_l+1}^{n_l+n_u}$, where x_t^i denotes the ith instance in X_t, n_l and n_u are the number of labeled and unlabeled instances in the target respectively. $X_t = X_t^l \cup X_t^u$, and $n_t = n_l + n_u$. Instance in source X_s can be represented as $x_s^i = [a_s^{i1}; a_s^{i2}; \ldots; a_s^{im_s}]$ and instance in target X_t can be represented as $x_t^i = [a_t^{i1}; a_t^{i2}; \ldots; a_t^{im_t}]$. Here, $a_s^{ij}(a_t^{ij})$ represents the value of the jth metric in $x_s^i(x_t^i)$, m_s and m_t denote the size of metrics for the source and target, respectively. Considering the large difference in values of different metrics, we first employ the z-score normalization (i.e., zero mean and unit standard deviation) to preprocess data, which is similar to the N2 normalization [34].

The goal of MDA is to learn V projection matrices (i.e., $P_v \in m_v \times d, v = 1, 2, \ldots, V$) to map the V input domains to a shared d-dimensional subspace, where (1) the distributions of data between each source and target domains are close to each other, (2) the local geometrical structures of each domain is preserved, (3) the instances from the same class across the input domains are located as close together as possible and (4) the instances from different classes are well separated from each other. Fig. 3 illustrates the schematic diagram of MDA.

(2) Detailed Procedure. MDA contains the following three terms: *source-target discrepancy reducing term, locality preserving term, class discriminant term*.

Source-Target Discrepancy Reducing Term. This term is used to make the data distributions of each source and target domains be close to each other. We define this term as follows:

$$F_{red} = \sum_{a=1}^{V-1} \sum_{i=1}^{n_a} \sum_{j=1}^{n_b} \left\| P_a^T x_a^i - P_b^T x_b^j \right\|_2^2, \tag{4}$$

where $b = V$, $P_a \in \mathcal{R}^{m_a \times d}$ and $P_b \in \mathcal{R}^{m_b \times d}$ denote the projections on source and target domains.

Locality Preserving Term. This term is used to preserve the geometrical structure of each domain. It is defined as follows:

$$F_{loc} = \frac{1}{2} \sum_{a=1}^{V} \sum_{i=1}^{n_a} \sum_{j=1}^{n_a} \left\| P_a^T x_a^i - P_a^T x_a^j \right\|^2 W_l^a(i,j), \tag{5}$$

where $W_l^a \in \mathcal{R}^{n_a \times n_a}$ denotes the affinity matrix for the ath domain which is built using a simple KNN graph. Entries of the matrices are $W_l^a(i,j) = 1$ if instances x_i and x_j from domain v are neighbors in the KNN graph of that domain and 0 otherwise. Here, we select 10 neighbors based on some defect prediction papers [29], [33], [35]. This term ensures that neighboring instances in the original domains are mapped close to each other in the projected common subspace. The geometry is preserved solely within each graph

$$\begin{gathered} \min_{P_v, v=1,2,\ldots,V} F_{red} + \alpha F_{loc} + \beta F_{disc} \\ s.t.\ P^T P = I, \end{gathered} \tag{6}$$

where I denotes an identity matrix. α and β are parameters to balance the terms F_{red}, F_{loc} and F_{disc}.

(3) Optimization. Although V variables $(P_1, \ldots, P_V)$ need to be optimized in Eq. (6), they can be obtained together after some algebraic transformations. Specifically, let $P = [P_1; P_2; \ldots P_V]$, $\phi_v = [O_{m_v \times m_1}, \ldots, I_{m_v}, O_{m_v \times m_{v+1}}, \ldots, O_{m_v \times m_V}]$, where I_m is a $m \times m$ identity matrix, and $O_{m \times r}$ is a $m \times r$ zero matrix. Then we have $p_v = \phi_v P$. By substituting p_v into Eq. (6), we can rewrite the objective function as

$$\begin{gathered} \min_P tr\left(P^T(\Lambda_r + \alpha\Lambda_l + \beta\Lambda_d)P\right) \\ s.t.\ \ P^T P = I \end{gathered} , \tag{7}$$

where $\Lambda_r = \sum_{a=1}^{V-1} \sum_{i=1}^{n_a} \sum_{j=1}^{n_b} (\phi_a^T x_a^i - \phi_b^T x_b^j)(\phi_a^T x_a^i - \phi_t^T x_b^j)^T$, $\Lambda_l = \frac{1}{2} \sum_{a=1}^{V} \sum_{i=1}^{n_a} \sum_{j=1}^{n_a} \left(\phi_a^T x_a^i - \phi_a^T x_a^j\right)\left(\phi_a^T x_a^i - \phi_a^T x_a^j\right)^T \mathrm{W}_l^a(i,j)$, and $\Lambda_d = \frac{1}{2} \sum_{a=1}^{V} \sum_{b=1}^{V} \sum_{i=1}^{n_a} \sum_{j=1}^{n_b} T(W_s^{a,b}(i,j) - \gamma W_d^{a,b}(i,j))$, $= \phi_a^T x_a^i - \phi_b^T x_b^j$. $tr(\cdot)$ denotes the trace of a matrix.

By constructing the Lagrange function of Eq. (7) and setting the derivative of P to zero, we can get

$$(\Lambda_r + \alpha\Lambda_l + \beta\Lambda_d)P = \lambda P, \tag{8}$$

where λ is the Lagrange multiplier. Hence, the optimal solution of Eq. (8) can be obtained with eigenvalue decomposition technique [79] and P is determined by the d smallest eigenvectors.

After obtained P, we can find the projected data of each domain to the common subspace by simple matrix multiplication $P_v^T X_v, v = 1, 2, \ldots, V$. Specifically, the projected data of source and target can be represented as follows:

$$\begin{cases} D_s = P_s^T X_s,\ s = 1, 2, \ldots, V-1 \\ D_t^l = P_t^T X_t^l,\ t = V \\ D_t^u = P_t^T X_t^u,\ t = V. \end{cases} \tag{9}$$

Based on the projected data, we can conduct defect prediction to use logistic regression (LR) classifier, which has been widely used in defect prediction studies [28], [34], [38], [42], [80]. The LR is implemented in LIBLINEAR [81] (an award-winning library for large linear classification). For LIBLINEAR execution, we use the options "-S 0" (i.e., logistic regression) and "-B 1" (i.e., no bias term added) as used by Nam et al. [34]. Algorithm 3 provides the detail optimization process of MDA.

Algorithm 3. The MDA Algorithm

```
Input: Source data $X_s$ and corresponding class label $Y_s$, $s = 1, 2, \ldots, V-1$; training target data $X_t^l$ and corresponding class label $Y_t^l$; test target data $X_t^u$, $t = V$; $V$ is the number of input projects;
Output: The predicted label $\hat{Y}_t^u$ of $X_t^u$.
Employ the z-score normalization to preprocess $X_s$, $X_t^l$ and $X_t^u$;
Construct terms $\Lambda_h$, $\Lambda_l$ and $\Lambda_d$;
Construct the objective function by using Eq. (7);
Obtain the projected transformation $P$ by using Eq. (8);
Obtain the projected features $D_s$, $D_t^l$ and $D_t^u$ by using Eq. (9);
$\hat{Y}_t^u \longleftarrow Classifier(D_s, D_t^l, D_t^u, Y_s, Y_t^l)$, i.e., combine $D_s$ and $D_t^l$ to predict $D_t^u$.
```

Class Discriminant Term. To ensure that instances from the same class get close together, while instances from different classes move away from each other, we define the *class discriminant term* as follows:

$$F_{disc} = \frac{1}{2}\sum_{a=1}^{V}\sum_{b=1}^{V}\left(\sum_{(i,j)\in s}\left\|p_a^T x_a^i - p_b^T x_b^j\right\|^2 W_s^{a,b}(i,j) - \gamma\sum_{(i,j)\in d}\left\|p_a^T x_a^i - p_b^T x_b^j\right\|^2 W_d^{a,b}(i,j)\right), \tag{10}$$

where $W_s^{a,b}$ and $W_d^{a,b}$ denote the similarity and dissimilarity graph matrix between the ath and bth domains, respectively. $W_s^{a,b}(i,j) = 1$, if x_a^i and x_b^j are from the same class and 0 otherwise. $W_d^{a,b}(i,j) = 1$, if x_a^i and x_b^j are from the different classes and 0 otherwise (including unlabeled data in the target). Note that the term F_{disc} considers relations of labeled instances in multiple different domains.

In order to simultaneously achieve the above three goals in the projected common subspace, we combine Eqs. (4), (5) and (10), and define the objective function of MDA as follows:

4 Experiments

4.1 Data Set

In experiments, we employ 28 publicly available and commonly used projects from five different groups including NASA[4] [39], SOFTLAB[4] [29], ReLink[5] [82], AEEEM[6] [40] and PROMISE[4] [45] as the experimental data. Table 2 shows the details about the datasets we used.

NASA dataset is publicly available and commonly used for defect prediction [7], [39]. Each dataset in NASA represents a software system or sub-system, which contains the corresponding defect-marking data and various static code metrics. Static code metrics of NASA datasets include size, readability, complexity and etc., which are closely related to software quality. Here, we use only five projects including CM1, MW1, PC1, PC3 and PC4 from NASA dataset, since these five projects have 37 same metrics.

TABLE 2
Details of Project Used in Experiment

Group	Project	# of metrics	# of total instances	# of defective instances	% of defectiveinstances
NASA	CM1	37	327	42	12.84%
	MW1	37	253	27	10.67%
	PC1	37	705	61	8.65%
	PC3	37	1,077	134	12.44%
	PC4	37	1,458	178	12.21%
SOFTLAB	AR1	29	121	9	7.44%
	AR3	29	63	8	12.70%
	AR4	29	107	20	18.69%
	AR5	29	36	8	22.22%
	AR6	29	101	15	14.85%
ReLink	Apache	26	194	98	50.52%
	Safe	26	56	22	39.29%
	ZXing	26	399	118	29.57%
AEEEM	EQ	61	324	129	39.81%
	JDT	61	997	206	20.66%
	LC	61	691	64	9.26%
	ML	61	1,862	245	13.16%
	PDE	61	1,497	209	13.96%
PROMISE	ant1.3	20	125	20	16.00%
	arc	20	234	27	11.54%
	camel1.0	20	339	13	3.83%
	poi1.5	20	237	141	59.49%
	redaktor	20	176	27	15.34%
	skarbonka	20	45	9	20.00%
	tomcat	20	858	77	8.97%
	velocity1.4	20	196	147	75.00%
	xalan2.4	20	723	110	15.21%
	xerces1.2	20	440	71	16.14%

Turkish software dataset (SOFTLAB) consists of AR1, AR3, AR4, AR5 and AR6 projects. There exist 28 common metrics between SOFTLAB and NASA, which are both obtained from PROMISE repository. Although the defect data of these two groups are from the same repository, these datasets are very different from each other.

ReLink was collected by Wu et al. [82] and the defect information in ReLink has been manually verified and corrected. ReLink has 26 complexity metrics, which are widely used in defect prediction [82]. Among three used datasets, the number of instances ranges from 56 to 399, while the number of attributes is fixed to 26.

The AEEEM data set was collected by D'Ambros et al. [40]. AEEEM consists of 61 metrics: 17 source code metrics, 5 previous-defect metrics, 5 entropy-of-change metrics, 17 entropy-of-source-code metrics, and 17 churn-of-source code metrics [40].

The defect datasets in the fifth group are originally collected by Jureczko and Madeyski [45] from the online PROMISE data repository, which consists of several open source projects. These datasets are also used to study the privacy issue for defect prediction [60]. The PROMISE datasets have 20 metrics in total, which contains McCabe's cyclomatic metrics, CK metrics and other OO metrics.

[4] http://openscience.us/repo/
[5] http://www.cse.ust.hk/~scc/ReLink.htm
[6] http://bug.inf.usi.ch/

4.2 Evaluation Measures

In experiments, to evaluate the privacy of our SRDO algorithm, we employ the privatization measure: *IPR*. To evaluate the performance of our multi-source heterogeneous defect prediction approach, we employ two widely used measures: *g-measure* and *AUC*.

4.2.1 Privacy Evaluation

To measure privacy, we use the Increased Privacy Ratio (*IPR*) [60] as the percent of data not found to evaluate the privacy ability of an algorithm. We first apply EFB to the sensitive attribute to create subrangs of values, represented by a set $S = \{s_1, s_2, \ldots, s_{|S|}\}$. Then, we assume that attackers have access to privatized data T' of an original dataset T, and some background knowledge of non-sensitive values for a specific target in T. We refer to the background knowledge as a query. Given queries $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$, the *IPR* can be defined as follows:

$$100 * IPR(T^*) = 1 - \frac{1}{|Q|}\sum_{i=0}^{|Q|} Breach(S, G_i^*), \qquad (11)$$

where, $Breach(S, G_i^*) = \begin{cases} 1 & if\ s_{\max}(G_i) = s_{\max}(G_i'), \\ 0 & otherwise. \end{cases}$ G_i^* is a group of rows from any dataset which matches q_i. G_i is the group from the original dataset and G_i' is the group from the private data candidate which matches q_i. $s_{\max}(G_i)$ is the most common sensitive attribute value in the set s. For example, $G_i' = \{[0{-}1], (2{-}4], (2{-}4]\}$ denotes the results of ith query returned from the privatized dataset T', then $s_{\max}(G_i') = \{(2{-}4]\}$.

The higher *IPRs* the better a privacy algorithm, so a good privacy algorithm will have *IPRs* closer to 100 percent while a poor privacy algorithm will have *IPRs* closer to 0 percent.

A *query generator* is used to generate queries based on what the attacker may know about a target in the original dataset. In order to maintain some "realism" to the attacks, a selected sensitive attribute and the class attribute are not used as part of query generation. Therefore, these attribute values (sensitive and class) are unchanged in the privatized dataset.

Assume the query size is 1, which is used to measure the number of attribute subrange pairs. The process of the generator to create a query is as follows:

(1) Given a set of attributes B (i.e., the metric set of defect dataset except the sensitive and class attributes) and all their possible subranges (applying EFB with 10 bins to create subranges for the original data).

(2) Randomly select an attribute b from B, for example, the attribute b may be has three possible subranges $\{[0{-}1], (1{-}3], (3{-}6]\}$.

(3) Randomly select a subrange from all possible subranges of b, for example, $\{(1{-}3]\}$.

In the end, the query we generate is $b = \{(1{-}3]\}$. Each query must also satisfy: (1) they must not be the same as another query, (2) they must return at least one instance from the original data set.

4.2.2 Prediction Performance Evaluation

We employ two commonly used comprehensive measures to assess the performance of the defect predictors: *g-measure* [33], [36], [58], [60], [61] and *AUC* [37], [42], [51], [83], [84], [85].

TABLE 3
Four Kinds of Defect Prediction Results

	Actual defective	Actual defect-free
Predict defective	TP	FP
Predict defect-free	FN	TN

These measures can be defined by using true positive (*TP*), false negative (*FN*), false positive (*FP*) and true negative (*TN*) in Table 3. Here, *TP*, *FN*, *FP* and *TN* are the number of defective instances that are predicted as defective, the number of defective instances that are predicted as defect-free, the number of defect-free instances that are predicted as defective, and the number of defect-free instances that are predicted as defect-free, respectively.

Probability of detection (*Pd*) or *recall* is defined as *TP/(TP+FN)* which denotes the ratio of the number of defective instances that are correctly classified as defective to the total number of defective instances. Probability of false alarm (*Pf*) is defined as *FP/(FP+TN)* which denotes the ratio of the number of defect-free instances that are wrongly classified as defective to the total number of defect-free instances. Like *f-measure*, the *g-measure* [60] is harmonic mean of *Pd* and *1-Pf*, which is defined as *2*Pd*(1-Pf)/(Pd+(1-Pf))*. Here, the *1-Pf* represents *specificity* [86].

AUC is the area under the receiver operating characteristic curve. This curve is plotted in a two-dimensional space with *Pf* as *x*-coordinate and *Pd* as *y*-coordinate. The *AUC* is known as a useful measure for comparing different models and is widely used because *AUC* is unaffected by class imbalance as well as being independent from the prediction threshold. The higher *AUC* represents better prediction performance and the *AUC* of 0.5 means the performance of a random predictor [56]. Lessmann et al. [83] and Ghotra et al. [84] suggest to use the *AUC* for better comparability. Hence, we also select the *AUC* measure as our performance measure.

All the above evaluation measures range from 0 to 1. Obviously, an ideal defect prediction model should hold high values of *g-measure* and *AUC*. As observed by Menzies et al. [87], *precision* is highly unstable performance indicator when datasets contain a low percentage of defects. From Table 2, we see that the percentage of defects is low in most cases. As a result, other measures such as *precision*, *accuracy* and *f-measure* were not used in this paper.

4.3 Experimental Settings

4.3.1 Experiment Protocol

In the Single-Source Scenario. For each trial, we select one project from 28 projects as the target, and each project from other four groups is used as the source in turn. For example, when CM1 in NASA of Table 2 acts as the target, there exists 23 (28-5) cross-project prediction combinations. Since we mainly focus on prediction across projects with heterogeneous metric sets, we did not conduct defect prediction across projects in the same group where projects have the same metric sets. In total, we have 600 possible prediction combinations from these 28 projects of five groups. *In the multi-source scenario*: for each trial, one project from 28 projects is selected as the target, and all projects from the other four groups are used as the source projects together. The same process is followed when the projects are privatized. Note that the test target data is not privatized.

4.3.2 Evaluation Settings

For testing privacy, assume that a project has m_i attributes (metrics) with the jth attribute has n_j distinct subranges. If the query size is k, there will be $C_{m_i}^{k} * (n_j)^k$ possible queries. In this study, the used query sizes include 1, 2 and 4. It is unrealistic and unnecessary to enumerate all possible queries for all possible query sizes, especially when the number of Quasi-identifiers (QIDs)[7] attributes is very large. For example, the project CM1 in NASA of Table 2 has 36 QIDs attributes (all attributes minus the sensitive attribute), if each QIDs attribute has 10 distinct subranges and the query size is 4, then the generator will create $C_{36}^{4} * 10^4 = 589,050,000$ queries. We randomly select 1000 unique queries from the total number of possible queries for each query size in our experiments. If the number of queries generated is less than 1000, we use the actual generated queries. Similar to the works [60] and [61], we use the *Lines of Code* metric as the sensitive attribute in this study.

For testing utility, we conduct HDP experiments for all the projects of Table 2 in their original state and after they have been privatized. *For the original data*, we repeat the above study 10 times, each time using 10 percent available instances in a target project as training target data and the remaining 90 percent instances of the target project used for testing. *For the privatized data*, we run SRDO 10 times since there involves a degree of randomness in SRDO algorithm for obfuscating data. Therefore, we repeat the above study 100 (10×10) times in the privatized scenario, each time performs the same process as their original state. Then, we report the median results for each target project across the multiple runs.

4.3.3 Baselines

To evaluate the privacy of SRDO, we compare it with CLIFF +MORPH [60], which is a state-of-the-art privacy-preserving algorithm for CPDP.

Based on the privatized datasets, we conduct HDP experiments to assess our multi-source heterogeneous defect prediction approach (MSMDA). Hence, we compare MSMDA with prior methods including: WPDP, NN-filter [29], TCA+ [34], CPDP-IFS [43], CCA+ [41], HDP by KSAnalyzer (HDP-KS, KSAnalyzer led to the best prediction performance in the paper) [42], SC [51], MultiSourceTrAdaBoost [88] and HYDRA [38]. Note that we do not compare the CCT-SVM [44] method. CCT-SVM is just tailored for the SVM classifier, and it cannot be applied to other classifiers directly.

NN-filter and TCA+ are two representative CPDP methods. The idea of NN-filter is to select suitable training data from other projects close to within-project data. TCA+ combines data normalization techniques and a feature-based transfer learning method (i.e., transfer component analysis, TCA) to CPDP.

CPDP-IFS, CCA+ and HDP-KS are three HDP methods. CPDP-IFS uses distribution characteristics vectors of each instance as new metrics to enable defect prediction. The goal of CCA+ is to maximize the correlation of the source and target data, and make the data distributions of source and target similar. HDP-KS aims to match the selected source and target metrics based on the metric similarity.

SC is a spectral clustering method, which applies connectivity-based unsupervised classifier for defect prediction.

MultiSourceTrAdaBoost is an instance transfer method which uses multiple sources for object recognition and detection. It considers each source individually in combination with the target and retains only the single best combination after each boosting round. Intrinsically, HYDRA is a task-based boosting technique for instance transfer and it includes genetic algorithm and ensemble learning two phases. Different from the above methods [38], [88], our MSMDA can be referred to as feature representation based transfer approach [75]. The intuitive idea is to learn a "good" feature representation for the target domain, and the knowledge used to transfer across domains is encoded into the learned feature representation. Additionally, TCA+ and CCA+ also belong to this case.

4.3.4 Parameter Settings

For MSMDA, there are three parameters α, β and γ. α is used to control the effect of locality preserving term, β is used to control the effect of class discriminant term, and γ is used to balance the effect of instances from the same class and that from different classes.

Based on the source and a limited amount of labeled training target data (by default, 10 percent of labeled data), we separately search the following parameter space $[0.01, 0.05, 0.1, 0.5, 1, 5, 10]$ for α and β, $[0.5, 1, 2]$ for γ. Specifically, we first randomly divide the 10 percent labeled training target data into two halves: the first half with 5 percent and the second half with another 5 percent. Then we use the first half and the source together as the training set while the second half as the test set. In a reverse way, we use the second half and the source together as the training set while the first half as the test set. This process is similar to a two-fold cross validation. Finally, we can achieve the optimal parameter values according to the overall best *g-measure* result across all target projects.

As observed by Menzies et al. [7], the defect prediction data has low dimensionality nature. Gao et al. [89] compared various feature selection techniques, they found that the performance of the defect prediction models either improved or remained unchanged when over 85 percent of the software metrics were eliminated. Furthermore, Nam and Kim also [42] selected the top 15 percent of metrics in their HDP study. In this paper, for the projected dimension d in MSMDA, we set it to $0.15 * d_t$. Here, d_t is the number of metrics of target project.

For the other compared methods, we follow the default parameter settings used in their papers.

4.4 Experimental Design

These experiments are designed to address the following two research questions.

RQ1: How to design a privacy preservation algorithm that can provide favorable privacy and utility for the privacy needs of HDP?

[7] Attribute whose values alone or together with others can potentially identify an instance.

CLIFF+MORPH [60] has achieved good privacy-preserving results and prediction performance for CPDP. In order to address RQ1, we compare SRDO with CLIFF+MORPH for the privacy needs of HDP. For evaluating the privacy of each algorithm, we separately calculate the *IPR* for the privatized data produced by CLIFF+MORPH and SRDO. In terms of evaluating the utility, we first use MSMDA to build HDP models on the original data and privatized data produced by CLIFF+MORPH and SRDO, respectively. Then, we observe the *g-measure* and *AUC* values of the learned models and report on the median performance.

RQ2: How effective is using multiple heterogeneous source projects to improve the HDP prediction performance?

In order to address RQ2, we conduct HDP experiments under two settings: using a single source project and using multiple source projects. Accordingly, we split RQ2 into two following sub-questions.

(1) RQ2.1: *Does MSMDA provide better HDP performance than existing representative methods that use a single source project?*

To answer RQ2.1, we compare MSMDA to the conventional WPDP method, the typical CPDP methods NN-filter and TCA+, the HDP methods CPDP-IFS, CCA+ and HDP-KS, the spectral clustering based method SC. Additionally, we also compare the results of MSMDA using multiple sources and single source. Here, we refer to MSMDA using a single source project as *sMDA*.

(2) RQ2.2: *Does MSMDA provide better HDP performance than existing representative methods that use multiple source projects?*

To answer RQ2.2, we compare MSMDA with two representative multi-source methods, including MultiSourceTrAdaBoost and HYDRA.

For NN-filter, TCA+, MultiSourceTrAdaBoost and HYDRA, we conduct HDP experiments using common metrics shared by source and target projects, because these methods are not designed originally for HDP. For MultiSourceTrAdaBoost and HYDRA, since they need a small amount of labeled instances from the target, we take all instances from the source and 10 percent of labeled instances from the target to predict the remaining 90 percent of instances in the target. For other compared methods, to ensure that we use the same test set to evaluate all methods for a fair comparison, we remove the same 10 percent of instances in the target and predict the same remaining 90 percent of instances in the target. We implement NN-filter, TCA+, CPDP-IFS, HDP-KS, SC, MultiSourceTrAdaBoost and HYDRA based on MATLAB programming by following the settings of corresponding papers. To be fair, we choose logistic regression classifier for all compared methods (except SC which is a clustering-based method) and apply z-score normalization to all of training and test data before running these methods.

We use the above methods to separately build prediction model and evaluate the performance of these methods on the 28 heterogeneous projects from five groups. To check whether our approach can obtain better prediction performance with the compared methods are statistic significant, for each target, we employ the non-parametric Wilcoxon rank-sum test (also called the Mann-Whitney U test[8]) [90] at a confidence level of 95 percent on 100 random running results which corresponds to the detailed *g-measure* and *AUC* values. This statistic test has been used in some defect studies [2], [29], [35], [37]. As recommended by Menzies et al. [2], Wilcoxon rank-sum test does not demand that the two compared populations are of the same size and it can avoid the need of Bonferroni-Dunn test to counteract the results of multiple comparisons [91]. In this paper, the compared methods WPDP and SC only have 10 random running results, as compared with other methods and MSMDA, they have different size of populations. So, we choose the non-parametric Wilcoxon rank-sum test. Then, we report the win/tie/loss (w/t/l) results of our approach against each compared method, similar to prior studies in [30], [32], [36], [38]. “Win” means that the results of our approach are significantly better than those of baselines at a confidence level of 95 percent, “tie” means ”equal” (no statistical significance), and otherwise “lose”. By using the w/t/l evaluation, we can investigate the number of projects in which our approach can outperform the compared methods. Note that the total number of projects is 28.

TABLE 4
Mappings of Cliff’s Delta Values to Effectiveness Levels

Cliff’s Delta ($\lvert\delta\rvert$)	Effectiveness Levels
$\lvert\delta\rvert < 0.147$	Negligible (N)
$0.147 \leq \lvert\delta\rvert < 0.33$	Small (S)
$0.33 \leq \lvert\delta\rvert < 0.474$	Medium (M)
$0.474 \leq \lvert\delta\rvert$	Large (L)

Besides, to measure the degree of differences in the results between our approach and the baselines, we compute Cliff’s delta (δ), which is a non-parametric effect size test [92], [93]. In this context, δ is a measure of how often one the values in one method are larger than the values in a second method. All possible values of δ are in the closed interval $[-1, 1]$, where -1 or 1 indicates that all values in one method are smaller or larger than those of the other method, and 0 indicates that the measure in the two methods is completely overlapping. The mappings between different δ values and their effectiveness levels are shown in Table 4.

5 Experimental Results

5.1 Results for RQ1

Fig. 4 displays the *IPR*, *g-measure* and *AUC* values of each privatized project. For each chart, we plot *IPR* on the x-axis, *g-measure* and *AUC* values on the y-axis respectively. The *g-measure* and *AUC* values are based on the proposed MSMDA approach and the *IPR* is based on queries of size 1. In Fig. 4, the *horizontal* lines show the *g-measure* and *AUC* values of the original project and the *vertical* line shows *IPR*=80%, respectively.

From Fig. 4, we can see that the privatized projects produced by CLIFF+MORPH and SRDO appearing in the *upper right* of these lines, which means that both data are private enough (i.e., the *IPR* over 80 percent) and good results can be achieved with both privatized data. The reasons that

[8] https://en.wikipedia.org/wiki/Mann-Whitney_U_test

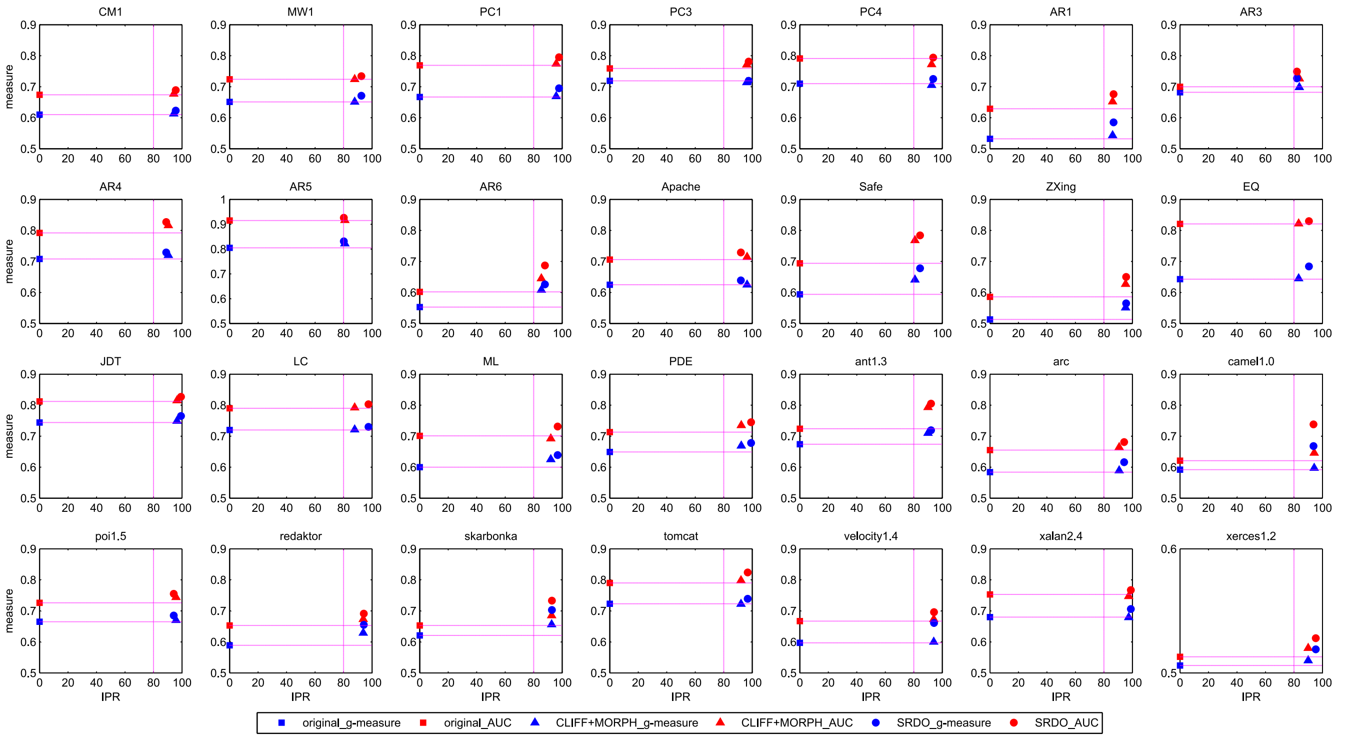


Fig. 4. The *g-measure* versus *IPR* and *AUC* versus *IPR* with query size 1 for each target project. The *horizontal* lines show the *g-measure* and *AUC* values of the original project and the *vertical* line shows *IPR* with value 80 percent, respectively. Note that points above and to the right of these lines are private enough (*IPR* over 80 percent) and performs as good as or better than the original data.

CLIFF+MORPH and SRDO have enough privacy include the following two aspects: (1) CLIFF preserves the privacy of the instances it deletes since these instances are no longer available; (2) MORPH or SRDO increases the privacy of all mutated instances since their original data distorted. In terms of utility, due to the instance selection done by CLIFF, as claimed by Peters and Menziers [60], most *g-measure* and *AUC* values are higher than those of the original projects.

With regard to the *IPR* results between CLIFF+MORPH and SRDO, we find that most of the privatized projects produced by SRDO have achieved comparable *IPR* values to CLIFF+MORPH. We also find that most of the *g-measure* and *AUC* values of SRDO are higher than CLIFF+MORPH. This is because SRDO simultaneously uses NUN and NSN as disturbances, and employs sparse representation based the nearest neighbor selector that is more likely to select the closest instance for the original instance. So the utility of the dataset can be further maintained. In conclusion, SRDO achieves comparable privacy and yields better *g-meausre* and *AUC* values in utility than CLIFF+MORPH in most of the cases under the HDP setting.

The *IPR* results for query size 2 and 4 are shown in Fig. 7 of Section 6.1. This summary of results also hold true when measuring *IPRs* for query sizes 2 and 4 (see Fig. 7).

5.2 Results for RQ2

5.2.1 Results for RQ2.1

Table 5 shows the median *g-measure* and *AUC* results for each target project between our approach and the compared methods that use a single source project. In Table 5, "All" refers to the median values across 28 target projects in terms of *g-measure* and *AUC*, and the bold font represents the best values for each target project. "API (%)" refers to the *average performance improvement* of our approach over the baselines. From Table 5, we can observe that MSMDA obtains the highest *g-measure* and *AUC* values in most cases as compared with WPDP using 10 percent training data, NN-filter, TCA+, CPDP-IFS, CCA+, HDP-KS, SC and sMDA. With respect to the performance across 28 target projects (i.e., "All" in the last row of each table), MSMDA improves the *g-measure* values by 8.1, 11.5, 9.8, 15.3, 8.2, 9.1, 4.5 and 3.4 percent, and the *AUC* values by 14.9, 19.1, 16.9, 21.2, 10.7, 8.7, 11.3 and 3.0 percent over WPDP with 10 percent training data, NN-filter, TCA+, CPDP-IFS, CCA+, HDP-KS, SC and sMDA, respectively. *Thus, we can see that the improvement is 3.4-15.3 percent in terms of g-measure and 3.0-19.1 percent in terms of AUC.*

The overall API of *g-measure* and *AUC* across 28 target projects are 8.8 and 13.2 percent, respectively. With regard to the API for each target project, we find that all the projects can achieve different degrees of improvement in terms of *g-measure* (except for the Safe project) and *AUC*. Specifically, the API of *g-measure* and *AUC* are from -0.2 to 37.4 percent and from 2.7 to 28.0 percent, respectively.

Possible reasons that MSMDA achieves better results are as follows. *Compared with NN-filter, TCA+, CPDP-IFS, CCA+ and HDP-KS*: (1) These methods only utilize single source to predict the target project, while MSMDA takes advantage of multiple source projects. Generally, multiple source projects can provide more useful information than a single one. With these useful information contained in multiple sources, the learned prediction model can own better prediction capability in the target project. (2) They don't consider utilizing the label information of the source data, which is available in practice. Different from these methods, MSMDA utilizes the useful source label information to learn projection matrices, such that instances can own favorable

TABLE 5
Our Approach Compared with WPDP Using 10 Percent Training Data, and with NN-Filter, TCA+, CPDP-IFS, CCA+, HDP-KS and SC That Use a Single Source Project for Each Target

Target	*g-measure*										*AUC*									
	WPDP	NN-filter	TCA+	CPDP-IFS	CCA+	HDP-KS	SC	sMDA	MS MDA	API	WPDP	NN-filter	TCA+	CPDP-IFS	CCA+	HDP-KS	SC	sMDA	MS MDA	API
CM1	59.9	59.6	60.6	56.0	60.4	59.3	61.5	60.8	**62.0**	3.8	62.3	61.8	64.7	59.4	66.1	67.6	64.2	67.3	**68.6**	7.1
MW1	64.4	64.5	63.4	62.7	66.1	65.6	**67.4**	65.6	66.8	2.9	64.4	64.8	68.9	64.2	69.7	70.6	67.8	70.7	**73.1**	8.2
PC1	63.7	60.7	64.0	58.8	64.1	64.2	64.2	66.2	**69.2**	9.6	64.7	61.7	64.9	61.4	71.6	72.6	65.0	75.0	**79.2**	18.6
PC3	66.8	59.3	63.0	56.9	58.8	64.2	67.7	70.9	**71.6**	13.4	66.9	61.1	64.7	59.9	62.9	73.3	68.1	77.3	**77.8**	17.3
PC4	**73.2**	61.2	67.9	58.1	62.6	63.5	66.9	71.6	72.2	10.6	76.3	61.9	68.2	62.2	67.8	70.3	67.0	78.2	**79.1**	15.4
AR1	47.9	56.0	56.1	56.3	60.1	59.3	**64.1**	56.1	58.2	2.8	49.4	56.5	57.5	58.8	66.8	**71.9**	65.1	60.4	67.3	12.0
AR3	43.9	71.3	**74.4**	71.7	74.0	73.4	73.8	69.3	72.4	8.0	46.9	71.6	76.0	72.9	81.6	**81.9**	76.5	68.4	74.6	6.6
AR4	68.4	70.8	73.2	63.4	70.4	66.6	73.0	71.4	**73.4**	5.6	68.9	71.3	76.5	66.6	78.0	78.6	73.2	80.4	**82.4**	11.5
AR5	81.7	74.9	80.6	73.1	79.8	80.6	**84.4**	81.0	82.8	4.3	81.9	75.2	80.9	74.6	85.6	88.6	84.9	91.4	**92.3**	11.9
AR6	58.2	54.2	49.7	60.6	60.7	56.1	61.2	59.4	**62.3**	8.8	60.1	56.8	56.2	62.8	63.6	61.7	61.0	65.7	**68.4**	12.4
Apache	63.4	58.2	62.6	52.8	62.5	58.6	62.6	62.8	**63.6**	5.6	64.8	64.4	68.0	60.1	70.7	68.0	66.2	71.2	**72.6**	9.2
Safe	66.2	64.5	69.4	64.8	**72.1**	67.2	71.8	66.0	67.5	-0.2	67.6	67.3	71.6	67.5	76.2	74.2	72.5	76.7	**78.1**	9.2
ZXing	50.6	51.4	52.5	52.2	55.7	54.6	55.7	55.8	**56.2**	5.1	53.5	57.5	58.6	57.3	61.1	63.4	59.5	62.0	**64.7**	9.7
EQ	67.5	56.2	59.3	56.8	67.3	61.6	67.7	67.8	**68.1**	8.7	69.2	61.8	65.1	63.7	73.8	77.3	69.6	82.0	**82.7**	18.6
JDT	74.1	63.6	68.8	64.2	63.2	65.4	75.9	73.3	**76.2**	11.7	75.3	66.1	71.9	67.1	64.8	72.2	76.5	78.6	**82.4**	15.7
LC	68.1	54.7	55.2	61.3	62.0	58.2	71.4	71.9	**72.7**	16.9	69.5	58.7	61.2	64.3	64.5	64.4	72.1	79.3	**80.0**	20.8
ML	63.5	58.9	62.0	57.3	55.8	58.2	60.6	62.3	**63.6**	6.5	66.2	61.3	64.8	59.8	57.5	63.3	62.7	71.5	**72.8**	15.3
PDE	61.5	63.1	65.4	60.4	59.1	63.3	66.8	65.1	**67.5**	7.2	63.8	64.0	66.3	62.6	60.4	68.8	67.2	72.0	**74.2**	13.4
ant1.3	66.8	70.1	69.9	69.8	61.9	65.8	66.4	70.2	**71.6**	6.1	66.8	70.4	70.0	70.8	64.4	73.7	69.1	78.1	**80.2**	14.2
arc	60.0	59.1	60.4	60.2	60.3	60.3	54.8	60.8	**61.3**	3.2	62.4	61.9	62.7	62.9	**70.4**	66.9	55.1	67.1	67.8	7.0
camel1.0	49.2	60.7	56.3	60.7	61.1	61.0	54.4	63.0	**66.5**	14.8	56.2	61.9	58.9	67.1	64.8	65.5	56.4	68.2	**73.5**	18.5
poi1.5	64.5	53.7	58.4	57.8	60.4	60.4	66.8	67.1	**68.2**	12.1	65.4	58.1	63.7	62.8	61.2	69.9	67.6	74.0	**75.2**	15.7
redaktor	59.6	48.0	47.3	50.3	59.5	57.1	40.2	64.0	**65.2**	25.1	59.8	50.7	50.8	54.2	60.9	61.1	37.4	67.7	**68.8**	28.0
skarbonka	61.7	60.6	59.0	62.1	64.0	62.5	**73.5**	68.2	70.0	10.0	61.7	62.0	60.9	64.5	62.5	70.1	**79.7**	71.3	73.0	10.5
tomcat	67.7	74.1	**76.2**	67.8	58.5	68.1	71.7	72.9	73.6	6.3	68.5	74.5	76.3	69.4	61.7	74.3	73.1	80.3	**82.1**	14.3
velocity1.4	64.2	37.6	31.7	41.2	60.5	52.3	55.8	64.6	**65.8**	37.4	64.8	48.6	45.5	47.3	63.7	62.4	59.3	67.3	**69.3**	23.5
xalan2.4	64.4	68.0	67.8	64.0	58.0	62.0	67.6	68.8	**70.3**	8.4	65.7	69.5	72.0	65.9	60.1	68.7	68.4	75.7	**76.4**	12.4
xerces1.2	50.5	40.6	38.9	41.8	44.7	46.4	43.7	49.3	**52.0**	17.8	**54.4**	53.4	54.1	50.6	53.0	51.2	44.4	50.3	52.6	2.7
All	62.0	60.1	61.0	58.1	61.9	61.4	64.1	64.8	**67.0**	8.8	65.1	62.8	64.0	61.7	67.6	68.8	67.2	72.6	**74.8**	13.2

The last row shows the median g-measures *and* AUC *across 28 target projects. "API (%)" denotes the average performance improvement of our approach compared to the baseline methods. The best prediction values are in bold font.*

discriminant ability under the learned projection. In addition, for NN-filter and TCA+, they only adopt the common metrics shared by the source and target projects, which will limit their performance, because some informative metrics necessary for building a good prediction model may not be in the common metrics across projects [41], [42]. *Compared with the WPDP using* 10 *percent training data*: Generally, good prediction models can be built when sufficient historical data is available for WPDP [28], [29], [30]. However, in practice, it is difficult to collect sufficient historical data for new projects. In this case, the performance of WPDP is usually limited. Different from WPDP, our approach takes advantage of information contained in multiple source projects. *Compared with SC*: SC belongs to unsupervised clustering classifiers, who usually underperform supervised ones in terms of their predictive power [51].

TABLE 6
Win/Tie/Loss Results of Our Approach Against Each Compared Method in Terms of *g-Measure* and *AUC* That Use a Single Source Project

MSMDA versus Baselines	w/t/l					
	g-measure			*AUC*		
MSMDA versus WPDP	23	4	1	25	2	1
MSMDA versus NN-filter	22	5	1	25	2	1
MSMDA versus TCA+	21	4	3	23	3	2
MSMDA versus CPDP-IFS	25	3	0	27	1	0
MSMDA versus CCA+	20	5	3	22	4	2
MSMDA versus HDP-KS	21	5	2	20	6	2
MSMDA versus SC	16	6	6	24	2	2
MSMDA versus sMDA	19	9	0	20	8	0

Here, 'versus' is short for 'versus'.

Table 6 shows the win/tie/loss results of our approach against each compared method in terms of the *g-measure* and *AUC* values. We can see that MSMDA can statistically significantly improve the performance of baselines in most cases. For example, compared with HDP-KS, MSMDA obtains statistically significant improvements for 21 (20) projects out of 28 in terms of *g-measure* (*AUC*) values.

To give a clearer comparison of the prediction results, we count the number of projects in each effectiveness level according to Table 4, and report the results in Table 7. From Table 7, we can observe that MSMDA produces significant improvements in most cases as compared with related methods. Taking the results of MSMDA versus HDP-KS as an example, MSMDA achieves non-negligible differences for 23 out of 28 projects (22 out of 28 projects) in terms of the *g-measure* (*AUC*) values.

In general, we can draw the conclusion that MSMDA can provide higher HDP performance than existing representative methods that use a single source project.

TABLE 7
The Effectiveness Results of Our Approach Against Each Compared Method in Terms of *g-Measure* and *AUC* That Use a Single Source Project

MSMDA versus Baselines	N/S/M/L							
	g-measure				*AUC*			
MSMDA versus WPDP	4	2	9	13	3	3	1	21
MSMDA versus NN-filter	5	3	9	11	3	2	1	22
MSMDA versus TCA+	6	5	7	10	4	2	2	20
MSMDA versus CPDP-IFS	2	3	5	18	2	0	2	24
MSMDA versus CCA+	6	4	8	10	5	4	3	16
MSMDA versus HDP-KS	5	5	7	11	6	3	5	14
MSMDA versus SC	11	3	5	9	3	1	1	23
MSMDA versus sMDA	8	5	9	6	7	4	10	7

5.2.2 Results for RQ2.2

Table 8 shows the *g-measure* and *AUC* results for each target project between our approach and the compared methods that use multiple source projects. The median *g-measure* and *AUC* results across 28 target projects are also reported in the last row of the table (denoted as "All") and the best values of each target project are in bold font. As shown in Table 8, MSMDA achieves the best prediction values in most projects as compared with MultiSourceTrAdaBoost and HYDRA in terms of *g-measure* and *AUC*. In particular, compared with MultiSourceTrAdaBoost and HYDRA, MSMDA improves the median *g-measure* values by 5.7 and 5.4 percent, and the median *AUC* values by 3.5 and 2.3 percent, respectively. Although MultiSourceTrAdaBoost and HYDRA utilize multiple source projects for prediction, they cannot achieve satisfactory prediction performance. A possible reason is that they only use the common metrics shared by source and target projects for the learning of prediction models. The overall API of *g-measure* and *AUC* across 28 target projects are 5.5 and 2.9 percent, respectively. With regard to the API for each target project, we find that most projects can achieve the performance improvement. Specifically, the API of *g-measure* and *AUC* are from -3.1 to 81.3 percent and from -6.1 to 37.1 percent, respectively. However, the results of a few projects do not get improved. Possible reason could be that MSMDA could not select the "good" data sources when comparing to the multi-source baseline methods. Since there could exist large distribution differences between source and target projects, the improper data sources may bring negative influence to the learned MSMDA model, causing MSMDA to underperform. In the future, we plan to develop a better multi-source selection method, which can improve the prediction performance further.

TABLE 8
Our Approach Compared with MultiSourceTrAdaBoost (MSTAB) and HYDRA That Use Multiple Source Projects for Each Target

Target	*g-measure*				*AUC*			
	MS TAB	HY DRA	MS MDA	API	MS TAB	HY DRA	MS MDA	API
CM1	59.1	60.1	**62.0**	4.0	68.3	**70.1**	68.6	-0.9
MW1	65.0	65.3	**66.8**	2.5	71.4	72.3	**73.1**	1.7
PC1	67.6	66.5	**69.2**	3.2	77.7	77.3	**79.2**	2.2
PC3	70.9	69.8	**71.6**	1.8	76.9	77.0	**77.8**	1.1
PC4	71.6	69.0	**72.2**	2.7	**81.9**	76.9	79.1	-0.3
AR1	47.4	49.1	**58.2**	20.7	51.5	**74.0**	67.3	10.8
AR3	39.0	40.9	**72.4**	81.3	49.9	**82.2**	74.6	20.1
AR4	72.3	55.4	**73.4**	17.0	80.2	**83.1**	82.4	1.0
AR5	80.6	71.6	**82.8**	9.2	**93.1**	92.6	92.3	-0.6
AR6	55.3	36.9	62.3	40.8	63.9	68.2	**68.4**	3.7
Apache	62.3	**69.4**	63.6	-3.1	**77.3**	**77.3**	72.6	-6.1
Safe	67.2	**70.8**	67.5	-2.1	80.7	**82.4**	78.1	-4.2
ZXing	53.3	**59.7**	56.2	-0.2	64.4	64.6	**64.7**	0.3
EQ	65.7	67.0	**68.1**	2.7	79.4	78.7	**82.7**	4.6
JDT	69.7	72.0	**76.2**	7.6	78.0	78.4	**82.4**	5.4
LC	60.1	59.0	**72.7**	22.1	66.4	66.7	**80.0**	20.2
ML	62.8	**63.7**	63.6	0.6	69.0	68.4	**72.8**	6.0
PDE	65.9	66.6	**67.5**	1.9	71.9	71.5	**74.2**	3.5
ant1.3	**72.3**	71.5	71.6	-0.4	**81.6**	**81.6**	80.2	-1.7
arc	57.7	49.3	**61.3**	15.3	67.0	67.1	**67.8**	1.1
camel1.0	60.8	59.4	**66.5**	10.7	66.4	61.1	**73.5**	15.5
poi1.5	58.4	66.2	**68.2**	9.9	72.2	72.2	**75.2**	4.2
redaktor	47.3	37.6	**65.2**	55.6	50.4	50.0	**68.8**	37.1
skarbonka	59.0	58.6	**70.0**	19.1	72.7	72.8	**73.0**	0.3
tomcat	**76.9**	74.2	73.6	-2.6	81.8	81.8	**82.1**	0.4
velocity1.4	51.1	42.6	**65.8**	41.6	64.3	59.5	**69.3**	12.1
xalan2.4	**72.1**	71.5	70.3	-2.1	80.1	**80.3**	76.4	-4.7
xerces1.2	39.3	41.5	**52.0**	28.8	49.8	48.7	**52.6**	6.8
All	63.4	63.6	**67.0**	5.5	72.3	73.1	**74.8**	2.9

"API (%)" denotes the average performance improvement of our approach compared to the baseline methods. The best prediction values are in bold font.

TABLE 9
Win/Tie/Loss Results of Our Approach Against Each Compared Method in Terms of *g-Measure* and *AUC* Values That Use Multiple Source Projects

MSMDA versus Baselines	w/t/l					
	g-measure			*AUC*		
MSMDA versus MSTAB	20	5	3	16	7	5
MSMDA versus HYDRA	19	4	5	14	7	7

Tables 9 and 10 show the win/tie/loss and the effectiveness results of MSMDA against MultiSourceTrAdaBoost and HYDRA in terms of *g-measure* and *AUC* values, respectively. From Table 9, we can see that MSMDA achieves statistically significant performance improvements in most of the cases as compared with MultiSourceTrAdaBoost and HYDRA. From Table 10, we can observe that MSMDA obtains non-negligible effectiveness results in most cases as compared with MultiSourceTrAdaBoost and HYDRA. Therefore, we can conclude that MSMDA can provide higher HDP performance than related state-of-the-art methods that use multiple source projects.

5.2.3 Scott-Knott Effect Size Difference Test

In the above experiments, we use the Wilcoxon rank-sum statistic test and the effect size test separately for the evaluation.

TABLE 10
The Effectiveness Results of Our Approach Against Each Compared Method in Terms of *g-Measure* and *AUC* Values That Use Multiple Source Projects

MSMDA versus Baselines	N/S/M/L							
	g-measure				*AUC*			
MSMDA versus MSTAB	7	2	8	11	9	4	5	10
MSMDA versus HYDRA	7	3	9	9	11	3	6	8

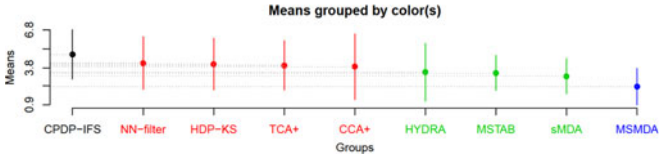


Fig. 5. Scott-Knott ESD test for all methods in terms of *g-measure*. The lower the rank value is, the better the performance is.

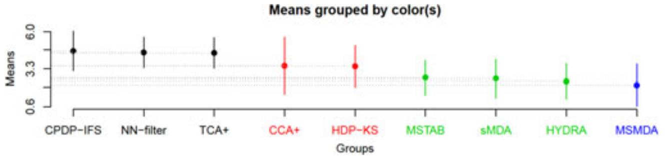


Fig. 6. Scott-Knott ESD test for all methods in terms of *AUC*. The lower the rank value is, the better the performance is.

Recent studies [85] and [6] suggest using the Scott-Knott effect size difference (ESD) test[9] to rank multiple different learners and classifiers. This test is a mean comparison approach that leverages hierarchical cluster analysis to partition the set of treatment means (e.g., means of variable importance scores, means of model performance) into statistically distinct groups ($\alpha = 0.05$) with non-negligible difference. The Scott-Knott ESD test is an alternative approach of the Scott-Knott test [84], [94] that considers the magnitude of the difference (i.e., effect size) of treatment means within a group and between groups. In this paper, we use the Scott-Knott ESD test that is provided by the *ScottKnottESD R* package [95].

Similar to prior work [6], we conduct a double Scott-Knott ESD test to compare our approach and baselines. We first apply a Scott-Knott ESD test on the mean results from the 100 random trails that are performed for each project individually. After conducting the first set of Scott-Knott ESD tests, we have a list of ranks for each method (i.e., one rank from each project). Based on these lists of ranks, we then apply the second run of Scott-Knott ESD test to them. After that, the test produces a ranking of methods across all the projects. To avoid potential comparison bias, we do not test WPDP and SC since they have only 10 random runs as described above. Figs. 5 and 6 show the Scott-Knott ESD test of our approach and the baselines across 28 projects on *g-measure* and *AUC*, respectively. We can observe that MSMDA ranks first in these two measures. The Scott-Knott ESD test results confirm that MSMDA overall performs better than the baseline methods.

By Investigating RQ2.1 and RQ2.2, we can Answer RQ2 as Follows. The utilization of well-selected multiple source projects is helpful for improving the HDP performance. The improvement is considerable in most cases.

6 Discussion

6.1 Privacy Testing with Different Query Sizes

To further evaluate the privacy of CLIFF+MORPH and SRDO, we conduct experiments with different query sizes (2 and 4), and report the corresponding *IPR* results. Fig. 7 displays the *IPR* results of CLIFF+MORPH and SRDO with query sizes 2 and 4 for each target project. We can see that the *IPR* values of each target project are all larger than 80 percent for both query sizes 2 and 4, which means that the *IPR* results of CLIFF+MORPH and SRDO are private enough. We also observe that SRDO obtains similar *IPR* results than CLIFF+MORPH in most cases for both query sizes 2 and 4, which means that SRDO can yield comparable privacy than CLIFF+MORPH. Hence, we believe SRDO is a good alternative option for preserving privacy in a scenario where data is shared for the purpose of HDP.

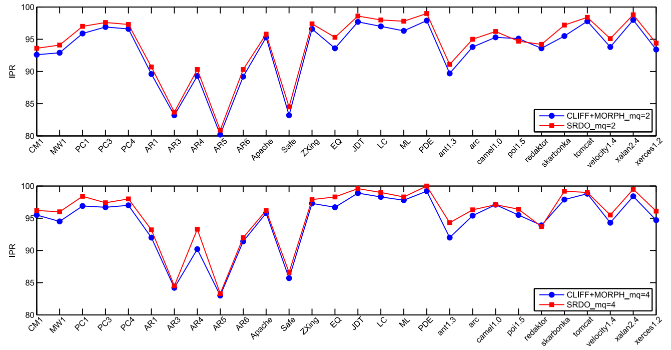


Fig. 7. The *IPR* results with query sizes 2 and 4 on 28 projects of CLIFF +MORPH and SRDO.

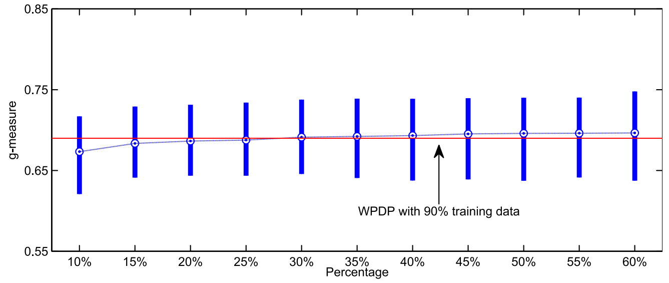


Fig. 8. Boxplot of the *g-measure* values for different percentages of training target data.

6.2 Effect of Different Percentages of Training Target Data

In this experiment, we evaluate the effect of different percentages of training target data to the performance of MSMDA. Specifically, the used percentage of labeled training target data ranges from 10 to 60 percent with step length 5 percent. For example, 15 percent of the training target data together with source project data are used for building the prediction model, the remaining 85 percent of instances in the target are used for testing. Additionally, we investigate how many percentages of labeled training target data should be employed together with source projects to achieve comparable prediction performance with WPDP under sufficient historical data. For WPDP under sufficient historical data, we randomly select 90 percent of labeled instances from the target to train defect predictors, and predict the label of the remaining 10 percent instances in that target.

To graphically visualize the prediction results across multiple runs, we use the compact boxplot to display the median, first and third quartiles of the *g-measure* and *AUC* values across 28 target projects similar to Menzies et al. [2] and D'Ambros et al. [40]. The bar indicates the first-third quartile range and circle denotes the median. The minimal and maximal values are not displayed.

[9] https://github.com/klainfo/ScottKnottESD

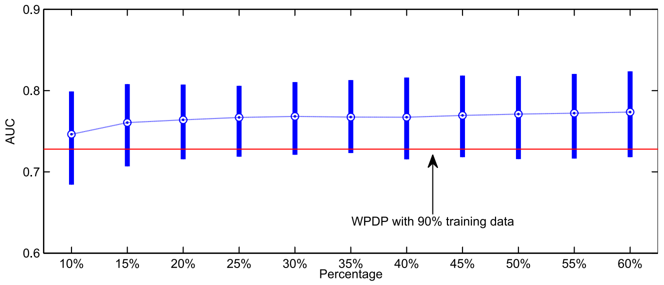


Fig. 9. Boxplot of the *AUC* values for different percentages of training target data.

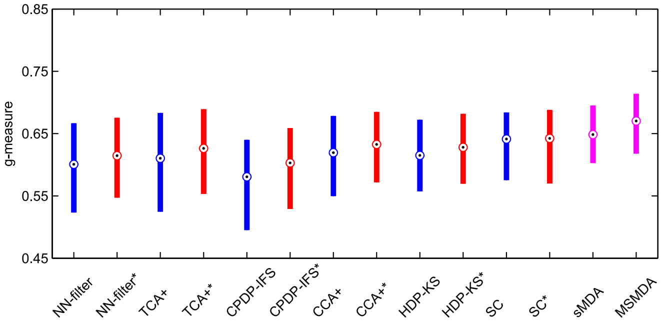


Fig. 10. Boxplot of the *g-measure* values for compared methods with and without training target data.

Figs. 8 and 9 separately show the compact boxplots of *g-measure* and *AUC* values across the 28 projects with different percentages of labeled training target data. From these figures, we can observe that the median *g-measure* and *AUC* of MSMDA increase when the percentage of labeled training target data increases from 10 to 20 percent, and the results are relatively stable when the percentage of labeled training target data is in the range of [20%, 60%]. This demonstrates that MSMDA almost has sufficient training data from the target project at about 20 percent. Adding more training data from the target project has little effect on the performance of MSMDA.

In Figs. 8 and 9, the red lines separately represent the median *g-measure* and *AUC* of WPDP using 90 percent within-project data. Their corresponding values are 0.692 and 0.728, respectively. We can see that, by using only 10 percent of labeled target data, MSMDA has obtained similar results as WPDP (using 90 percent training data). Our finding is consistent with previous CPDP studies [35], [36], [38]. These studies indicate that their models trained by using limited labeled instances from a target can produce comparable result to WPDP with sufficient training data. Therefore, in practice, it is worthwhile to use MSMDA for companies to conduct early defect prediction when there doesn't exist sufficient historical defect data.

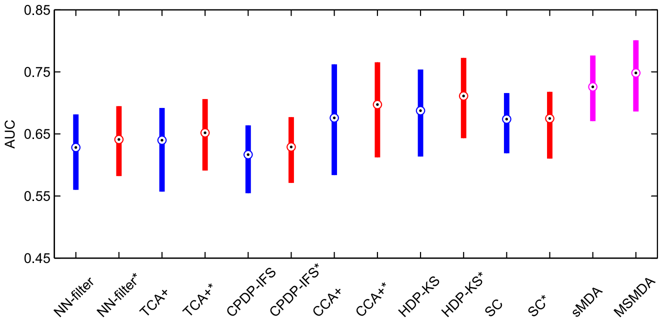


Fig. 11. Boxplot of the *AUC* values for compared methods with and without labeled training target data.

TABLE 11
Median *g-Measure* and *AUC* Values of Our Approach Compared with NN-filter*, TCA+*, CPDP-IFS*, CCA+*, HDP-KS* and SC*

Method	*g-measure*	*AUC*
NN-filter*	0.615	0.641
TCA+*	0.626	0.652
CPDP-IFS*	0.601	0.628
CCA+*	0.633	0.697
HDP-KS*	0.628	0.711
SC*	0.642	0.675
sMDA	0.648	0.726
MSMDA	0.670	0.748

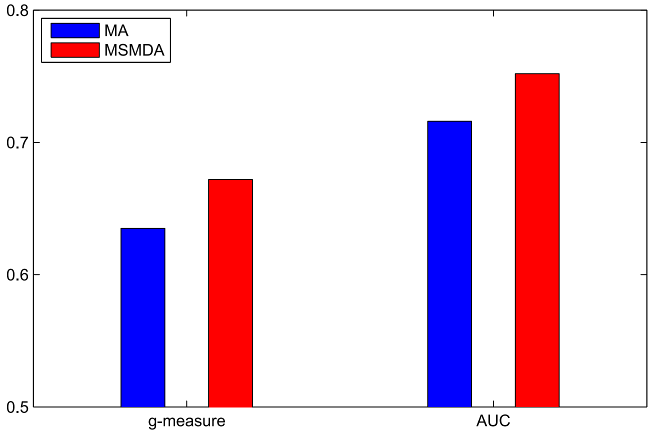


Fig. 12. The prediction results of MA and MSMDA across 28 target projects.

6.3 Effect of Training Target Data for Baselines

In experiments, MSMDA utilizes the 10 percent training target data, while the compared methods NN-filter, TCA+, CPDP-IFS, CCA+, HDP-KS and SC don't make use of this part of data. To investigate the effect of the 10 percent training target data to the performance of the compared methods, we divide instances belonging to each of the target project into two sets: 10 and 90 percent. Then the same 10 percent of labeled instances in the target are used for training by all these methods and our approach, and the remaining 90 percent instances are used for testing. That is, we incorporate the 10 percent of labeled instances into the training sets for these compared methods. We call NN-filter, TCA+, CPDP-IFS, CCA+, HDP-KS and SC using the 10 percent of labeled instances in a target as NN-filter*, TCA+*, CPDP-IFS*, CCA+*, HDP-KS* and SC* (here, SC* denotes to cluster all the instances in the target), respectively.

Figs. 10 and 11 show the compact boxplots of median *g-measure* and *AUC* for our approach and these compared methods. Table 11 shows the detailed median *g-measure* and *AUC* values of NN-filter*, TCA+*, CPDP-IFS*, CCA+*, HDP-KS* and SC*. As can be seen from Figs. 10 and 11 as well as Table 11, both the median *g-measure* and *AUC* values of NN-filter*, TCA+*, CPDP-IFS*, CCA+* and HDP-KS* are slightly increased, while the SC* is almost the same. However, their *g-measure* and *AUC* values are still lower than those of MSMDA.

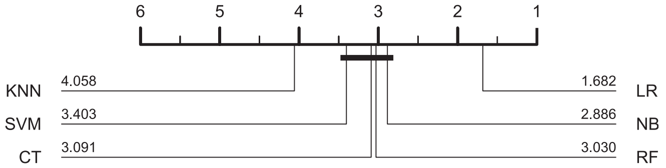


Fig. 13. Comparison of all classifiers against each other with Nemenyi's post-hoc test in *g-measure*. Groups of classifiers that are not significantly different (at p=0.05) are connected.

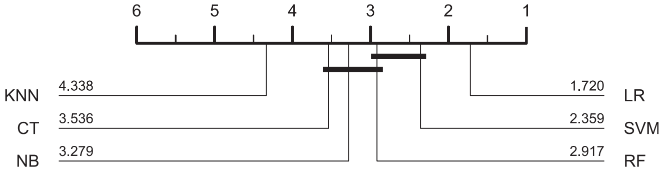


Fig. 14. Comparison of all classifiers against each other with Nemenyi's post-hoc test in *AUC*. Groups of classifiers that are not significantly different (at p=0.05) are connected.

6.4 Effect of Multi-Source Selection Based Manifold Discriminant Alignment

In this experiment, we evaluate the effect of proposed MSMDA approach. To this end, we compare the results of MSMDA and the related manifold alignment method [77]. Fig. 12 shows the median *g-measure* and *AUC* values of MSMDA and MA across the 28 target projects. We can find that our MSMDA approach achieves higher results than MA in terms of both the *g-measure* and *AUC* values.

The reasons are that: (1) MA does not select the proper sources for the target. In practice, there exist large distribution differences between some sources and the target project, which will bring negative influence to the learned prediction model. However, MSMDA aims to select the sources that are similar to the target, such that MSMDA can learn more beneficial knowledge for target. (2) MA does not consider reducing the data discrepancy between each source project and unlabeled instances in the target explicitly, while MSMDA designs a source-target discrepancy reducing term (F_{red}) in Eq. (4), which is used to reduce the distribution differences between each source and target projects.

6.5 Effect of Different Classifiers

In this section, we evaluate the effect of different classifiers to the prediction performance of MSMDA. To this end, we compare the performance of MSMDA with default LR classifier and other five classifiers including k nearest neighbor (KNN), naive Bayes (NB), support vector machine (SVM), classification tree (CT) and random forest (RF). The implementations of these five classification techniques are provided by Matlab Statistics and Machine Learning Toolbox with default parameter settings.[10]

To statistically analyze the experimental results using different classifiers, we conduct the non-parametric Friedman test with the Nemenyi's post-hoc test at a confidence level of 95 percent when comparing multiple classifiers over the 28 projects [91]. This test has been used in the defect prediction studies [40], [80], [83]. The Friedman test compares whether the average ranks are statistically significant. The Nemenyi's post-hoc test aims to check if the performance of each pair classifiers is significantly different. The statistical test is performed on the detailed results (100 random running). Then we report the visual representation results. Figs. 13 and 14 graphically visualize the comparison results of the Nemenyi's post-hoc test after Friedman test on *g-measure* and *AUC*, respectively. Classifiers that are not statistically significant are connected. From these figures, we can observe MSMDA generally achieves the best prediction results with LR classifier as compared with other classifiers in terms of *g-measure* and *AUC* values.

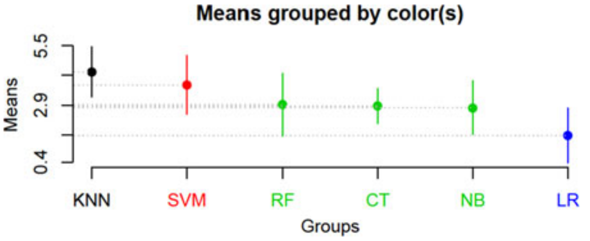


Fig. 15. Comparison of all classifiers against each other with Scott-Knott ESD test in terms of *g-measure*. The lower the rank value is, the better the performance is.

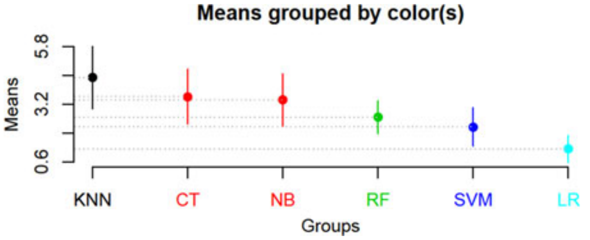


Fig. 16. Comparison of all classifiers against each other with Scott-Knott ESD test in terms of *AUC*. The lower the rank value is, the better the performance is.

Prior works [6], [84], [85] point out that Nemenyi's post-hoc test produces overlapping groups of classification techniques. It implies that there exists no statistically significant difference among the defect prediction models built using multiple different classification techniques. To overcome the confounding issue of overlapping groups, we use the Scott-Knott ESD test [6], [85], [95] to rank multiple different classifiers. Similar to Section 5.2.3, we perform a double Scott-Knott ESD test. Figs. 15 and 16 show the comparison results of the Scott-Knott ESD test on *g-measure* and *AUC* across 28 target projects with six different classifiers. The results demonstrate that LR statistically significantly performs better than the other classifiers. Thus, the use of different choices of the statistical test produces different ranking of classification techniques.

6.6 Practical Guidelines for HDP

Prediction of software defects works well within the same project as long as sufficient historical defect data is available for training the prediction model [28], [29]. Therefore, for a target project, if there is sufficient historical data, the software engineers can employ the WPDP methods to conduct defect prediction. However, there might not be enough historical data for a new software project or a new company. In practice, there usually exists plenty of data from other

[10] Version R2014a, http://mathworks.com/help/stats/index.html

companies or organizations. If these external data has the same metric sets with the target project, software engineers can conduct CPDP by using the recently proposed CPDP methods [29], [32], [34], [36], [37], [38]. The above CPDP methods may not be feasible when the metric sets of external data and target are heterogeneous. In this case, the recently proposed HDP methods [41], [42] can be employed.

Before software engineers use these CPDP or HDP methods, they need to collect the external data. However, due to privacy concerns of data owners, it's not an easy task to obtain defect data from data owners. To facilitate data sharing, the data owners can adopt our privacy-preserving algorithm SRDO to protect the privacy before they release their data.

In the early phases of software development, there usually exists a limited amount of historical data in projects. If the limited amount of historical data can be used for training together with multiple external projects, the learned prediction model will have more favorable prediction ability. Therefore, the software engineers can use our MSMDA approach to build effective prediction models in the early phase. When enough historical data is collected, the software engineers can also employ the WPDP methods to learn the defect predictors.

In Section 5, we conduct extensive and large-scale experiments on 28 projects with heterogeneous metric sets from five groups. Experimental results show that MSMDA outperforms the typical CPDP methods (NN-filter and TCA+), the distribution characteristic based method CPDP-IFS, the state-of-the-art HDP methods (CCA+ and HDP-KS), the spectral clustering method SC, the multi-source instance transfer method MultiSourceTrAdaBoost, the hybrid model reconstruction CPDP method HYDRA and WPDP. The Wilcoxon ranksum statistical test, the Cliff's delta effect size test and the Scott-Knott ESD test also validate this conclusion. Therefore, we suggest utilization of MSMDA at the early stages of software development activities.

6.7 Threats to Validity

Followings are several potential threats to the validity with respect to our empirical study.

(1) Generalizability bias. Prior work [96] points out that researchers should experiment with a broader selection of datasets and metrics in order to maximize external validity. Thus, we chose 28 projects from five groups that are widely used in papers published in top software engineering venues [7], [29], [34], [40], [41], [42], [60], [82]. These projects come from both proprietary (NASA and SOFTLAB) and open-source (ReLink, AEEEM and PROMISE) data sets. Therefore, our findings might not be generalizable to other closed software projects. In the future, we will plan to reduce this bias by conducting further experiments on more defect data from open source and commercial systems.

(2) Comparison bias. This paper focuses on the software defect prediction. In terms of the privacy-preserving algorithm, we compare SRDO against CLIFF+MORPH, which is a well-known privatization method for CPDP. As to more other privacy methods in the software engineering field, experiments might need to be done to evaluate our approach and this is a subject for future work. With respect to the related compared methods for HDP, we carefully implement the NN-filter, TCA+, CPDP-IFS, HDP-KS, SC, MultiSourceTrAdaBoost and HYDRA methods by following the corresponding papers. However, our implementation may not be exactly the same as the original methods, leading to that there could be a bias in the comparison between our approach and these methods. Recent works [97] and [85] point out that classification techniques with default parameter settings have a large impact on the performance of defect prediction models, which usually leads to suboptimal prediction results. Thus, the parameter settings of the classification techniques should be carefully tuned. Based on the findings of [97] and [85], there is a bias in the comparison between different classifiers with the default parameter settings. In future work, we plan to tune the parameters of the classifiers to further improve the defect prediction performance.

(3) Sampling bias. Since the privacy-preserving algorithm SRDO has a randomness to create different obfuscation instances, we mitigate this potential bias with 10 runs of the experiment for SRDO. In addition, to construct a training target data, we randomly select 10 percent of the instances in a target project, and we mitigate this potential bias with 10 runs of the experiment.

(4) Evaluation bias. This paper uses one measure of privacy (i.e., *IPR*), which is the same as prior works [60], [61]. Other privacy measures used in software engineering are not reported. Regarding the privatized project, we measure its utility empirically with defect prediction. We employ two comprehensive measures *g-measure* and *AUC*, both of them have been widely used to evaluate the effectiveness of defect prediction [33], [36], [37], [42], [51], [58], [60], [61], [83], [84], [85]. Measuring privacy and utility with other measures is left for future work.

7 Conclusion

Heterogeneous defect prediction is very promising as it permits potentially all heterogeneous data of software projects to be used for defect prediction on new projects or projects lacking in sufficient defect data. To support HDP, we provide a new privatization algorithm named sparse representation based data obfuscation (SRDO). With SRDO, data from other software development organizations can be privatized, which facilitates data sharing. To make full use of the information contained in multiple source projects, we propose a multi-source heterogeneous defect prediction approach, i.e., MSMDA. Extensive and large-scale experiments are conducted on the 28 projects from five groups. The non-parametric Wilcoxon ranksum statistical test, Cliff's delta effect size test and Scott-Knott ESD test are employed for the evaluation. Experimental results show the effectiveness of the privatization algorithm SRDO and the multi-source heterogeneous defect prediction approach MSMDA. Compared to a range of baseline methods, MSMDA can achieve the improvement with 3.4-15.3 percent *g-measure* and 3.0-19.1 percent *AUC*. Furthermore, SRDO achieves comparable privacy and better utility values than the CLIFF +MORPH method. The improved defect prediction performance with our approach is beneficial to allocating limited human and time resources effectively in software quality assurance activities.

In the future, we plan to evaluate our approach with data from more software projects. Additionally, in software defect datasets, there intrinsically exist class imbalance problem [49], [98], [99], [100], [101], which could affect the prediction performance of learned models.

We provide the datasets and the source code of the proposed approach that are used to conduct this study at https://sites.google.com/site/mssmdav/.

ACKNOWLEDGMENTS

The authors would like to thank the editors and anonymous reviewers for their constructive comments and suggestions. This work was supported by the NSFC-Key Project of General Technology Fundamental Research United Fund under Grant No. U1736211, the National Key Research and Development Program of China under Grant No. 2017YFB0202001, the National Nature Science Foundation of China under Grant Nos. 61373038, 61672392, 61472178, 61672208, U1404618, 41571417, the Science and Technology Program in Henan province under Grant No.1721102410064, the Science and Technique Development Program of Henan under Grant No.172102210186, the Province-School-Region Project of Henan University under Grant No.2016S11, and Research Foundation of Henan University No.2015YBZR024.

REFERENCES

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic literature review on fault prediction performance in software engineering,” *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov./Dec. 2012.

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

[3] A. Tosun, A. Bener, B. Turhan, and T. Menzies, “Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry,” *Inf. Softw. Technol.*, vol. 52, no. 11, pp. 1242–1257, 2010.

[4] T. Menzies, et al., “Local versus global lessons for defect prediction and effort estimation,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 822–834, Jun. 2013.

[5] M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: The use of machine learning in software defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, Jun. 2014.

[6] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, Jan. 2017.

[7] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[8] S. Kim, E. J. Whitehead, and Y. Zhang, “Classifying software changes: Clean or buggy?” *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, Mar./Apr. 2008.

[9] K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support vector machines,” *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.

[10] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, “Dictionary learning based software defect prediction,” in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 414–423.

[11] X.-Y. Jing, Z.-W. Zhang, S. Ying, F. Wang, and Y.-P. Zhu, “Software defect prediction based on collaborative representation classification,” in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 632–633.

[12] T. Wang, Z. Zhang, X.-Y. Jing, and L. Zhang, “Multiple kernel ensemble learning for software defect prediction,” *Automated Softw. Eng.*, vol. 23, no. 4, pp. 569–590, 2016.

[13] Z. Zhang, X.-Y. Jing, and T. Wang, “Label propagation based semi-supervised learning for software defect prediction,” *Automated Softw. Eng.*, vol. 24, no. 1, pp. 47–69, 2017.

[14] X. S. Si, C. H. Hu, and Z. J. Zhou, “Fault prediction model based on evidential reasoning approach,” *Sci. China Inf. Sci.*, vol. 53, no. 10, pp. 2032–2046, 2010.

[15] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 284–292.

[16] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in *Proc. 30th Int. Conf. Softw. Eng.*, 2008, pp. 531–540.

[17] A. E. Hassan, “Predicting faults using the complexity of code changes,” in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 78–88.

[18] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, “Change bursts as defect predictors,” in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng.*, 2010, pp. 309–318.

[19] T. Jiang, L. Tan, and S. Kim, “Personalized defect prediction,” in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2013, pp. 279–289.

[20] L. Chen, et al., “Empirical analysis of network measures for predicting high severity software faults,” *Sci. China Inf. Sci.*, vol. 59, no. 12, pp. 1–18, 2016.

[21] Y. M. Zhou, et al., “An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems,” *Sci. China Inf. Sci.*, vol. 55, no. 12, pp. 2800–2815, 2012.

[22] T. Lee, J. Nam, D. Han, S. Kim, and H. In, “Developer micro interaction metrics for software defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1015–1035, Nov. 2016.

[23] L. C. Briand, W. L. Melo, and J. Wust, “Assessing the applicability of fault-proneness models across object-oriented software projects,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 706–720, Jul. 2002.

[24] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, “Predicting faults from cached history,” in *Proc. 29th Int. Conf. Softw. Eng.*, 2007, pp. 489–498.

[25] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, “Reducing features to improve code change-based bug prediction,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 552–569, Apr. 2013.

[26] Y. Kamei, et al., “A large-scale empirical study of just-in-time quality assurance,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.

[27] T. Wang, Z. Zhang, X.-Y. Jing, and Y. Liu, “Nonnegative sparse-based semiboost for software defect prediction,” *Softw. Testing Verification Rel.*, vol. 26, no. 7, pp. 498–515, 2016.

[28] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: A large scale experiment on data versus domain versus process,” in *Proc. 7th Joint Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2009, pp. 91–100.

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

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

[31] M. Li, H. Zhang, R. Wu, and Z. H. Zhou, “Sample-based software defect prediction with active and semi-supervised learning,” *Automated Softw. Eng.*, vol. 19, no. 2, pp. 201–230, 2012.

[32] Y. Ma, G. Luo, X. Zeng, and A. Chen, “Transfer learning for cross-company software defect prediction,” *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, 2012.

[33] F. Peters, T. Menzies, and A. Marcus, “Better cross company defect prediction,” in *Proc. 10th Work. Conf. Mining Softw. Repositories*, 2013, pp. 409–418.

[34] J. Nam, S. J. Pan, and S. Kim, “Transfer defect learning,” in *Proc. 35th Int. Conf. Softw. Eng.*, 2013, pp. 382–391.

[35] B. Turhan, A. T. Mısırlı, and A. Bener, “Empirical evaluation of the effects of mixed project data on learning defect predictors,” *Inf. Softw. Technol.*, vol. 55, no. 6, pp. 1101–1118, 2013.

[36] L. Chen, B. Fang, Z. Shang, and Y. Tang, “Negative samples reduction in cross-company software defects prediction,” *Inf. Softw. Technol.*, vol. 62, pp. 67–77, 2015.

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

[38] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "HYDRA: Massively compositional model for cross-project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 977–998, Oct. 2016.

[39] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

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

[41] X.-Y. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," in *Proc. 10th Joint Meet. Found. Softw. Eng.*, 2015, pp. 496–507.

[42] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proc. 10th Joint Meet. Found. Softw. Eng.*, 2015, pp. 508–519.

[43] P. He, B. Li, and Y. Ma, "Towards cross-project defect prediction with imbalanced feature sets," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1411.4228

[44] M. Cheng, G. Wu, M. Jiang, H. Wan, G. You, and M. Yuan, "Heterogeneous defect prediction via exploiting correlation subspace," in *Proc. 28th Int. Conf. Softw. Eng. Knowl. Eng.*, 2016, pp. 171–176.

[45] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, 2010, pp. 1–10.

[46] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'Union fait la force," in *Proc. Softw. Evolution Week - IEEE Conf. Softw. Maintenance Reengineering Reverse Eng.*, 2014, pp. 164–173.

[47] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Softw. Quality J.*, vol. 25, no. 1, pp. 235–272, 2017.

[48] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Defect prediction as a multiobjective optimization problem," *Softw. Testing Verification Rel.*, vol. 25, no. 4, pp. 426–459, 2015.

[49] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 321–339, Apr. 2017.

[50] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 297–308.

[51] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 309–320.

[52] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2013, pp. 45–54.

[53] S. Herbold, "Training data selection for cross-project defect prediction," in *Proc. 9th Int. Conf. Predictive Models Softw. Eng.*, 2013, pp. 6–15.

[54] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng.*, 2008, pp. 19–24.

[55] X. Xia, D. Lo, S. McIntosh, E. Shihab, and A. E. Hassan, "Cross-project build co-change prediction," in *Proc. 22nd IEEE Int. Conf. Softw. Anal. Evolution Reengineering*, 2015, pp. 311–320.

[56] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, 2012, pp. 1–11.

[57] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2072–2106, 2016.

[58] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 1–39, 2016.

[59] F. Peters and T. Menzies, "Privacy and utility for defect prediction: Experiments with MORPH," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 189–199.

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

[61] F. Peters, T. Menzies, and L. Layman, "LACE2: Better privacy-preserving data sharing for cross project defect prediction," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 801–811.

[62] K. Taneja, M. Grechanik, R. Ghani, and T. Xie, "Testing software in age of data privacy: A balancing act," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 201–211.

[63] B. Li, M. Grechanik, and D. Poshyvanyk, "Sanitizing and minimizing databases for software application test outsourcing," in *Proc. IEEE 7th Int. Conf. Softw. Testing Verification Validation*, 2014, pp. 233–242.

[64] M. Castro, M. Costa, and J.-P. Martin, "Better bug reporting with better privacy," in *Proc. 13th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2008, pp. 319–328.

[65] J. Clause and A. Orso, "Camouflage: Automated anonymization of field data," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 21–30.

[66] A. Budi, D. Lo, L. Jiang, and Lucia, "kb-anonymity: A model for anonymized behaviour-preserving test and debugging data," in *Proc. 32nd ACM SIGPLAN Conf. Program. Language Des. Implementation*, 2011, pp. 447–457.

[67] F. Qi, X.-Y. Jing, X. Zhu, F. Wu, and L. Cheng, "Privacy preserving via interval covering based subclass division and manifold learning based bi-directional obfuscation for effort estimation," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 75–86.

[68] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: Wiley, 2012.

[69] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *J. Artif. Intell. Res.*, vol. 6, no. 1, pp. 1–34, 2000.

[70] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 481–490.

[71] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng.*, 2015, pp. 812–823.

[72] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.

[73] S. J. Kim, K. Koh, M. Lustig, and S. Boyd, "An interior-point method for large-scale L1-regularized least squares," *IEEE J. Sel. Topics Signal Process.*, vol. 1, no. 4, pp. 606–617, Dec. 2007.

[74] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, "Visual domain adaptation: A survey of recent advances," *IEEE Signal Process. Mag.*, vol. 32, no. 3, pp. 53–69, May 2015.

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

[76] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, no. 1, pp. 1–40, 2016.

[77] C. Wang and S. Mahadevan, "Heterogeneous domain adaptation using manifold alignment," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1541–1546.

[78] D. Tuia, M. Volpi, M. Trolliet, and G. Camps-Valls, "Semisupervised manifold alignment of multimodal remote sensing images," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 12, pp. 7708–7720, Dec. 2014.

[79] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.

[80] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2015, pp. 1–12.

[81] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, no. 9, pp. 1871–1874, 2008.

[82] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: Recovering links between bugs and changes," in *Proc. 19th ACM SIGSOFT Symp. Found. Softw. Eng. 13th Eur. Softw. Eng. Conf.*, 2011, pp. 15–25.

[83] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul./Aug. 2008.

[84] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng.*, 2015, pp. 789–800.

[85] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 321–332.

[86] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Softw. Eng.*, vol. 13, no. 5, pp. 561–595, 2008.

[87] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors"'," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 637–640, Sep. 2007.

[88] Y. Yao and G. Doretto, "Boosting for transfer learning with multiple sources," in *Proc. 23rd IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 1855–1862.

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

[90] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*. Hoboken, NJ, USA: Wiley, 1999.

[91] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.

[92] N. Cliff, *Ordinal Methods for Behavioral Data Analysis*. East Sussex, U.K.: Psychology Press, 2014.

[93] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," in *Proc. 11th Joint Meet. Found. Softw. Eng.*, 2017, pp. 72–83.

[94] Y. Yang, et al., "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 157–168.

[95] C. Tantithamthavorn, "ScottKnottESD: The Scott-Knott effect size difference (ESD) test (version-2.02)," 2017. [Online]. Available: https://cran.r-project.org/web/packages/ScottKnottESD/index.html

[96] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Comments on "researcher bias: The use of machine learning in software defect prediction"," *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1092–1094, Nov. 2016.

[97] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Inf. Softw. Technol.*, vol. 76, pp. 135–146, 2016.

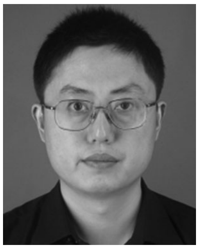
[98] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng.*, 2015, pp. 99–108.

[99] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[100] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.

[101] H. Zhang and X. Zhang, "Comments on "data mining static code attributes to learn defect predictors"," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 635–637, Sep. 2007.

Zhiqiang Li is working toward the PhD degree in the State Key Laboratory of Software Engineering, School of Computer, Wuhan University. His research interests include software engineering and machine learning.



Xiao-Yuan Jing is now a professor in the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, and in the College of Automation, Nanjing University of Posts and Telecommunications, China. He has published more than 70 papers in the international journals and conferences like the *IEEE Transactions on Software Engineering* (TSE), the *Automated Software Engineering* (ASE), the *IEEE Transactions on Image Processing* (TIP), Int. Conf. Software Engineering (ICSE), ACM Symp. Foundations of Software Engineering (FSE), Int. Conf. on Automated Software Engineering (ASE), Int. Joint Conf. Artificial Intelligence (IJCAI), and AAAI Conf. Artificial Intelligence (AAAI). His main research interests include software engineering, program analysis, machine learning, and artificial intelligence.

Xiaoke Zhu received the PhD degree in pattern recognition and intelligence system from the Wuhan University, Wuhan, China, in 2017. He is currently an associate professor in the School of Computer and Information Engineering, Henan University, China. His current research interests include person re-identification, image classification, and software engineering.

Hongyu Zhang received the PhD degree from National University of Singapore, in 2003. He is an associate professor with the University of Newcastle, Australia. Previously, he was a lead researcher at Microsoft Research Asia, an associate professor with Tsinghua University, China, and a lecturer with RMIT University, Australia. His research is in the area of software engineering, in particular, software analytics, testing, maintenance, metrics, and reuse. The main theme of his research is to improve software quality and productivity by mining and analyzing software data. He has published more than 100 Research Papers in international journals and conferences, including the *IEEE Transactions on Software Engineering*, the *ACM Transactions on Software Engineering and Methodology*, ICSE, FSE, ASE, ISSTA, POPL, AAAI, ICSM, ICDM, and USENIX ATC. He received two ACM Distinguished Paper awards. He has also served as a program committee member for many software engineering conferences.

Baowen Xu is now a professor and dean in the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, and a professor in the Department of Computer Science and Technology, Nanjing University. He has published more than 200 papers in the international journals and conferences like the *ACM Transactions on Software Engineering and Methodology* (TOSEM), the *IEEE Transactions on Software Engineering* (TSE), the *Journal of Artificial Intelligence Research*, Int. Conf. on Automated Software Engineering (ASE) and Int. Joint Conf. Artificial Intelligence (IJCAI). His main research interests include programming languages, software testing, software maintenance, and software metrics.

Shi Ying is now a professor with Wuhan University, where he is vice dean in the Computer School and he is also the deputy director of the State Key Laboratory of Software Engineering. He has authored or co-authored more than 100 referred papers in the area of software engineering. His main research interests include service-oriented software engineering, Semantic Web service, and trustworthy software.