The Journal of Systems & Software 202 (2023) 111721

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

ARRAY: Adaptive triple feature-weighted transfer Naive Bayes for cross-project defect prediction ☆

Haonan Tong [a], Wei Lu [a], Weiwei Xing [a], Shihai Wang [b,*]

[a] School of Software Engineering, Beijing Jiaotong University, 100044, Beijing, China
[b] School of Reliability and Systems Engineering, Science & Technology on Reliability & Environmental Engineering Laboratory, Beihang University, 100191, Beijing, China

ARTICLE INFO

Article history:
Received 10 December 2022
Received in revised form 18 February 2023
Accepted 18 April 2023
Available online 24 April 2023

Keywords:
Cross-project defect prediction
Common metrics
Transfer learning
Feature weighting
Model adaptation

ABSTRACT

Context: Cross-project defect prediction (CPDP) aims to predict defects of target data by using prediction models trained on the source dataset. However, owing to the huge distribution difference, it is still a challenge to build high-performance CPDP models.
Objective: We propose a novel high-performance CPDP method named adaptive triple feature-weighted transfer naive Bayes (ARRAY).
Methods: ARRAY is characterized by feature weighted similarity, feature weighted instance weight, and the model adaptive adjustment. Experiments are performed on 34 defect datasets. We compare ARRAY with seven state-of-the-art CPDP methods in terms of area under ROC curve (AUC), F1, and Matthews correlation coefficient (MCC) with statistical testing methods.
Results: Experimental results show that: (1) on average, ARRAY separately improves MCC, AUC, and F1 over the baselines by at least 18.4%, 6.5%, and 4.5%; (2) ARRAY significantly performs better than each baseline on most datasets; (3) ARRAY significantly outperforms all baselines with non-negligible effect size according to post-hoc test.
Conclusion: It can be concluded that: (1) the proposed feature weighted similarity, feature weighted instance weight, and the model adaptive adjustment are very helpful for improving the performance of CPDP models; (2) ARRAY is a more promising alternative for CPDP with common metrics.

1. Introduction

Software defect prediction (SDP) plays an important role in allocating testing resources and improving testing efficiency by helping software testers pay more attention to the defective modules, which has drawn increasing attention of both academia and industrial communities (Menzies et al., 2010; Hosseini et al., 2019; Zhou et al., 2018; Yedida and Menzies, 2021; Zheng et al., 2022; B and Sadam, 2023). Most SDP approaches train prediction models on the historical defect data collected from a project and then use the trained models to predict the defects of new software modules in the same project, which are called within-project defect prediction (WPDP) (Xu et al., 2019; Zhao et al., 2019; Zhu et al., 2021). However, at the early stage of a project, since there is no sufficient historical defect data, it is difficult to perform WPDP.

To address this issue, cross-project defect prediction (CPDP) is proposed (Turhan et al., 2009; Ryu et al., 2015; Ma et al., 2012; Xia et al., 2016; Nam et al., 2013; Chen et al., 2015; Jing et al., 2015; Nam et al., 2018; MA et al., 2017; Li et al., 2018; Wang et al., 2020; Giray et al., 2023). CPDP aims to build a prediction model on the given source dataset and then uses the model on the target data collected from a new project. Existing CPDP methods can be divided into two categories (Nam et al., 2018): cross-project defect prediction with common metrics (CPDP-CM a.k.a. homogeneous defect prediction) and heterogeneous defect prediction (HDP). The former assumes that source and target datasets share the same metric set and the latter assumes that their metric sets are different. The key to build a high-performance CPDP model is to reduce the distribution difference between the source and target datasets. Many CPDP-CM methods have been proposed, such as instance-based (Turhan et al., 2009; Ryu et al., 2015; Ma et al., 2012; Xia et al., 2016; Ryu et al., 2016), feature-based (Panichella et al., 2014; Nam et al., 2013; Wang et al., 2020; Jing et al., 2017; Zou et al., 2021b), and the others (e.g., hybrid of instance and feature based method Hosseini et al., 2018; Zou et al., 2021a). Instance-based CPDP-CM methods try to reduce the distribution differences by selecting similar instances from source dataset as the training data (Turhan et al., 2009; Ryu et al., 2015) or weighting source instances (Ma et al., 2012; Chen et al., 2015; Xia et al.,

☆ Editor: Christoph Treude.
* Corresponding author.
E-mail address: wangshihai@buaa.edu.cn (S. Wang).

https://doi.org/10.1016/j.jss.2023.111721
0164-1212/

2016). Feature-based CPDP-CM methods try to find the latent common feature space by using data normalization (Panichella et al., 2014; Canfora et al., 2015) or other complex feature space transformation methods (Nam et al., 2013; Wang et al., 2020; Jing et al., 2017). Recently, some HDP methods also have been proposed, which can be divided into two main categories: feature matching (Yu et al., 2017; Nam et al., 2018) and feature space transformation (Jing et al., 2015; MA et al., 2017; Li et al., 2018, 2019a; Tong et al., 2021).

However, it is still a challenge to build a high-performance CPDP models owing to the huge distribution difference between source and target datatsets. We find that previous CPDP models have two major problems: (1) there is a large calculation bias of instance weight which plays an important role to reduce the distribution difference between source and target datasets, and (2) the robustness of CPDP model is insufficient, which refers to the ability to keep high-performance and stability of a defect prediction model even in extreme situations (e.g., the defect ratio of source dataset is extremely high or the size of source dataset is very small).

To bridge these gaps, we propose a novel CPDP-CM method named adaptive triple feature weighted transfer naive Bayes (ARRAY) based on feature weighted similarity, feature weighted instance weight, transfer learning, and model adaptive adjustment. Extensive experiments are performed on 34 different defect datasets and the performance is evaluated in terms of F1, MCC, and AUC. We compare ARRAY with seven state-of-the-art CPDP methods including HDP-KS (Nam et al., 2018), FMT (Yu et al., 2017), CFIW-TNB (Zou et al., 2021a), ManualDown (Zhou et al., 2018), DMDAJFR (Zou et al., 2021b), and one most famous deep semantic feature CPDP method i.e., DBN-CP (Wang et al., 2020). The following two research questions are investigated:

RQ1: How much improvement does ARRAY obtain compared with previous cross-project defect prediction models?

The experiment results show that, on average, ARRAY improves the performance over all baselines by 18.4%∼177.8%, 6.2%∼29.8%, and 4.5%∼533.4% in terms of MCC, AUC, and F1, respectively.

RQ2: Does ARRAY significantly outperform the deep semantic based cross-project defect prediction model?

The experiment results show that ARRAY significantly outperforms the deep semantic CPDP model (i.e., DBN-CP) on almost all datasets. On average, ARRAY improves the performance over DBN-CP by 122.1%, 16.5%, and 29.2% in terms of MCC, AUC, and F1, respectively.

The main contributions of this paper are presented as follows:

(1) To effectively reduce the calculation bias of instance weight, we propose the concept of feature weighted similarity. To the best our knowledge, we are the first to propose this concept for CPDP.

(2) To reduce the distribution difference between source and target datasets, we propose the concept of feature weighted instance weight based on feature importance and above feature weighted similarity.

(3) To achieve high robustness, we further propose the model self-adaptive adjustment strategy and the corresponding implement method, which can automatically adjust the model building strategy according to the characteristic of software defect datasets.

(4) A novel high-performance CPDP-CM approach named ARRAY is proposed, which combines unsupervised CPDP and supervised CPDP for CPDP. Extensive experiments are performed to demonstrate the superiority of ARRAY.

The rest of this paper is organized as follows: Section 2 briefly reviews the related work. Section 3 introduces the details of ARRAY. Sections 4 and 5 separately present experimental design and results. Section 6 further discusses our research. The potential threats to validity are shown in Section 7. Finally, this study is concluded in Section 8.

2. Related work

In this section, we briefly review the existing cross-project defect prediction methods from the following two aspects: (1) cross-project defect prediction with common metrics (CPDP-CM), and (2) and heterogeneous defect prediction (HDP).

2.1. Cross-project defect prediction with common metrics

CPDP-CM (a.k.a. homogeneous defect prediction) methods are the most widely researched CPDP methods, which assume that the source dataset uses the same metrics as that in the target dataset. Many CPDP-CM methods have been proposed.

Some CPDP-CM methods attempt to select source instances similar to target data from the source dataset as the training data (Turhan et al., 2009; He et al., 2012; Bhat and Farooq, 2022). Turhan et al. (2009) proposed to k-nearest neighbor method to select training instances from the source data based on Euclidean distance. He et al. (2012) used the distributional characteristics (median, mean, variance, standard deviation, etc.) to select training data. Bhat and Farooq (Bhat and Farooq, 2022) used Hamming distance to compute the similarity and then selected the similar source instances as the training dataset.

Some CPDP-CM methods try to reduce the distribution difference between the source and target data by weighting source instances according to their similarity to the target data (Ma et al., 2012; Chen et al., 2015; Xia et al., 2016; Ryu et al., 2016). Ma et al. (2012) proposed transfer naive Bayes (TNB) method for CPDP. For each source instance, its similarity is calculated based on the distribution of target data and the weight is computed based on data gravitation. Chen et al. (2015) assumed that target dataset has a small part of labeled instances which can be used during model training and proposed a double transfer boosting method. Xia et al. (2016) developed a CPDP method named hybrid model reconstruction approach which adjusts the weights of source instances during each round of training. Ryu et al. (2016) proposed a value-cognitive boosting with support vector machine approach which uses the similar method to TNB to calculate the similarity of source instances.

Some CPDP-CM approaches aim to minimize the distribution differences between the source and target datasets to find the latent common feature space by using feature space transformation (Nam et al., 2013; Wang et al., 2020; Jing et al., 2017; Wang et al., 2020; Zou et al., 2021b) or data normalization (Cruz and Ochimizu, 2009; Canfora et al., 2013). Specifically, Nam et al. (2013) proposed a new CPDP method called TCA+, which extends transfer component analysis (TCA) (Pan et al., 2011) by introducing a set of rules for selecting an appropriate normalization method. Jing et al. (2017) proposed an improved subclass discriminant analysis method to achieve balanced subclasses and then employed the semi-supervised transfer component analysis approach to find the latent common feature space. Wang et al. (2020) used deep belief network to automatically learn semantic features and then used semantic features to build a CPDP model named DBN-CP. Cruz and Ochimizu (2009) utilized log transformation to reduce the distribution difference. Canfora et al. (2013) used z-score normalization method to reduce the distribution difference. Zou et al. (2021b) proposed DMDAJFR, which uses two novel autoencoders to jointly learn the global and local feature representations, and then uses a repetitious pseudo-labels strategy to obtain distribution matching. Wang et al. (2016) proposed DBN-CP for CPDP based on deep belief network. They used deep belief network to automatically learn semantic features from token vectors extracted from the abstract syntax trees of source project, and then trained a prediction model by using the learned semantic metrics. Finally, the trained model was applied on the target project.

In addition, Hosseini et al. (2018) proposed a hybrid method of instance-selection and feature-selection for CPDP by combining genetic instance selection and feature selection. Krishna and Menzies (2019) proposed Bellwether for CPDP. Zou et al. (2021a) proposed correlation feature and instance weights transfer learning (CFIW-TNB), which uses both feature transfer and instance transfer to minimize the distribution differences.

Differing from existing CPDP-CM models, in our ARRAY, we proposed two concepts: feature-weighted similarity and feature-weighted instance weight. Specifically, we introduce the information of feature importance when calculating instance similarity and instance weight. We found that this can largely decrease the calculation bias of both instance similarity and instance weight compared with previous studies.

2.2. Heterogeneous defect prediction

Recently, some HDP methods have been proposed in previous studies (Jing et al., 2015; Nam et al., 2018; MA et al., 2017; Li et al., 2018; Yu et al., 2017).

Zhou et al. (2018) proposed ManualDown, which directly classifies the target instance according to its module size without using source dataset.

Jing et al. (2015) proposed a HDP method named CCA+ based on unified metric representation (UMR) and canonical correlation analysis (CCA) technique. The authors reconstructed the source and target datasets by using UMR and then used the CCA based transfer learning method to find the latent common feature space of both source and target datasets.

Considering the complex nonlinear relationships between software metrics and software defects, MA et al. (2017) proposed kernel canonical correlation analysis method by combining kernel method and CCA-based transfer learning technique.

Li et al. (2018) proposed a new HDP method named CTKCCA by combining cost-sensitive learning, kernel method, and CCA to address the class-imbalance problem and the nonlinear relationship problem.

Niu et al. (2022) proposed a HDP model based on data sampling and kernel manifold discriminant alignment (DSKMDA) to handle the class imbalance issue and the linear inseparability issue.

Yu et al. (2017) presented feature matching transfer (FMT) for HDP. FMT performs feature selection and then obtains the distribution curves of selected features for matching up the features between the source and target datasets.

Nam et al. (2018) also proposed a metric-matching based HDP method. They matched the source and target metrics based on metric similarity measurement (e.g., the Kolmogorov–Smirnov test) and the maximum weighted bipartite matching technique.

Differing from existing HDP models, our ARRAY introduced the model adaptive adjustment strategy, which can automatically select an appropriate model building scheme to obtain a CPDP model according to the characteristics of source dataset. The biggest benefit of this is that it can ensure the robustness of CPDP model, namely more stable high performance.

3. Methodology

3.1. Problem formulation

Let the labeled source project dataset be $D_S = (X_S, Y_S) = \left\{\left(x_i^s, y_i^s\right)\right\}_{i=1}^{n_s}$, where $x_i^s \in \mathbb{R}^{1\times d}$ is a row vector of software metrics corresponding to the ith source project module, $y_i^s \in \{0, 1\}$ denotes the corresponding defect information (i.e., 0 and 1 separately represent the non-defective and defective modules), d is the number of metrics, and D_S is a two-dimensional sample matrix with n_s rows and $(d+1)$ columns. Let the unlabeled target dataset be $D_T = (X_T) = \left\{\left(x_i^t\right)\right\}_{i=1}^{n_t}$, where $x_i^t \in \mathbb{R}^{1\times d}$ is a row vector of metrics for ith target project module. Note that we suppose that the source and target datasets have the same metrics and are collected from different software projects.

The objective of this paper is to build a cross-project defect prediction model based on the labeled source dataset D_S and then uses the model to predict the label (i.e., defective or non-defective) of the unlabeled target dataset X_T.

3.2. Adaptive triple feature weighted transfer naive Bayes

Fig. 1 presents the overall framework of ARRAY. ARRAY combines the supervised CPDP model (i.e., the proposed triple feature-weighted transfer naive Bayes) and the unsupervised CPDP model (i.e., ManualDown) by using the proposed adaptive adjustment strategy. In ARRAY, the 'triple feature weighted' refers to the feature weighted similarity, feature weighted instance similarity, and feature weighted posterior probability. The 'adaptive' refers to ARRAY can automatically select appropriate model according to the characteristics of source dataset, which is explained in detail in Section 3.2.6.

Then, we present the details of ARRAY as follows: (1) preprocessing; (2) feature importance based on maximal information coefficient; (3) feature weighted similarity; (4) feature weighted instance weight; (5) feature weighted posterior probability; (6) adaptive adjustment strategy.

3.2.1. Preprocessing

Considering the potential data quality problems of defect datasets as described in Shepperd et al. (2013), we need to preprocess the datasets as follows: (1) remove the instances having missing value, and (2) remove the duplicated instances.

Missing value may be caused by the mistake of data collectors. Since missing value might results in the failure of model training or testing, we just directly remove the instances having missing value. The reason to remove duplicated instances includes two aspects: (1) duplicated instances in the training dataset not only cannot help to improve model performance, but increase model training time; (2) duplicated instances in testing dataset will result in over-optimistic or over-pessimistic evaluation on the model performance.

For fairness, the baselines (see Section 4.3 for details) also take the same preprocessing progress. Furthermore, ARRAY and the baselines use the same training and testing datasets.

3.2.2. Feature importance

The importance of features can be measured by using feature-class relevance analysis. Feature-class relevance refers to relevance between a feature (a.k.a. software metric) and the class label (i.e., defective or non-defective). The larger feature-class relevance denotes stronger relevant relationship between this feature and the class label. Here, we use maximal information coefficient (MIC) (Reshef et al., 2011) to perform feature-class relevance analysis and then to measure feature importance. Before calculating feature importance, we first eliminate the class-imbalance problem of source dataset by using the synthetic minority oversampling technique and then transform the original independent variable values of source dataset using log filter. Log filter is also applied to the target dataset

Maximal information coefficient (MIC) is a more powerful and robust method for relevance analysis than other methods such as information gain, gain ratio (Quinlan, 1993), Relief (Kira and Rendell, 1992), and ReliefF (Kononenko, 1994). MIC has drawn increasing attention owing to its effectiveness (Xu et al., 2016; Wen et al., 2019). The range of MIC value is [0, 1]. Here, MIC

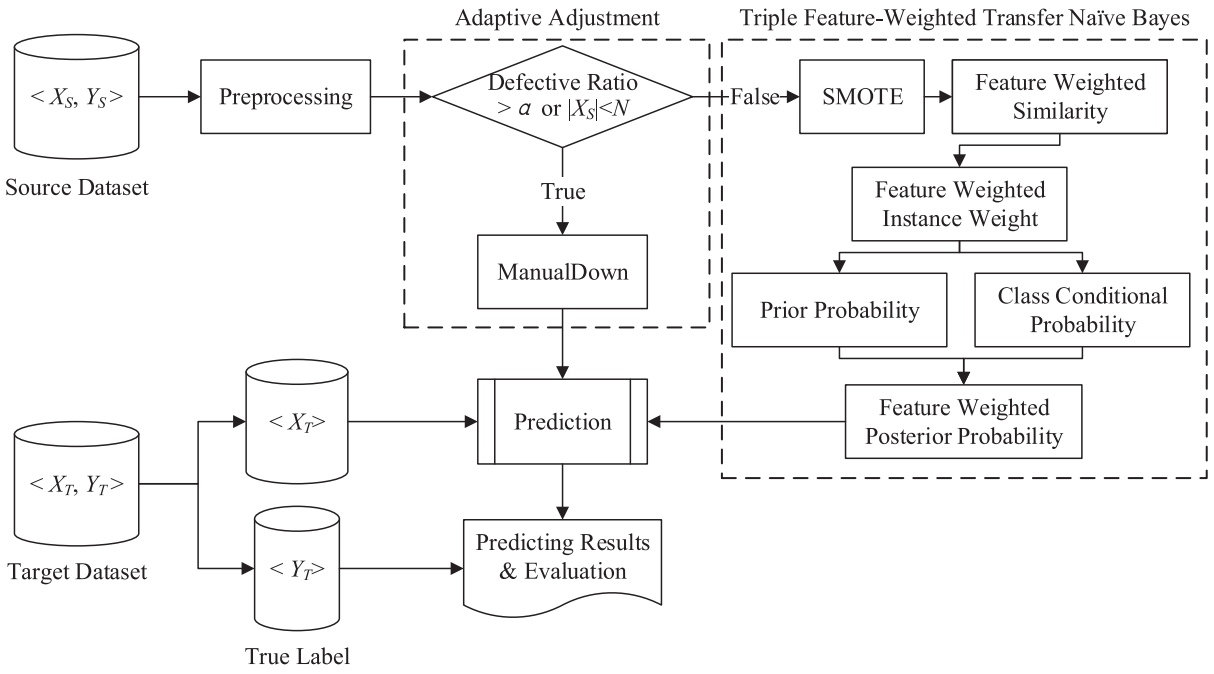


Fig. 1. Framework of the proposed ARRAY for cross-project defect prediction.

is used to determine the importance of features. We use the MATLAB toolbox minepy developed by Albanese et al. (2013) to calculate MIC.

After we obtain the MIC value $MIC(A_k^s, C)$ of each source feature A_k^s, and then normalize it as feature importance θ_k $\left(\sum_{i=1}^{d} \theta_k = 1\right)$.

3.2.3. Feature weighted similarity

In previous studies (TNB Ma et al., 2012 and CFIW-TNB Zou et al., 2021a), the instance similarity is calculated without considering the feature importance based on feature-class relevance analysis. This will result in a large deviation in the calculation of instance similarity. Therefore, we propose the concept of feature weighted similarity. It is the first time that the concept of feature weighted similarity is proposed in the field of software defect prediction.

Specifically, we can easily compute the maximum value max_k^t and the minimum value min_k^t of kth ($k = 1, 2, \ldots, d$) feature in the target dataset X_T.

For each source instance $x_i^s = \left(a_{i1}^s, a_{i2}^s, \ldots, a_{id}^s\right)$, its feature weighted similarity to the target dataset is defined based on the above feature importance θ_k as follows:

$$s_i^{fw} = \sum_{k=1}^{d} h\left(a_{ik}^s\right)\theta_k \tag{1}$$

where $h(a_{ik}^s) = 1$ when $min_k^t \le a_{ik}^s \le max_k^t$, otherwise 0.

3.2.4. Feature weighted instance weight

Instance similarity plays an important role in accurately calculating instance weight. Differing from the definition of instance weight in previous studies (TNB Ma et al., 2012 and CFIW-TNB Zou et al., 2021a), we propose the concept of feature weighted instance weight based on feature importance and feature weighted similarity. It is the first time to propose such a feature-weighted instance weight in the field of software defect prediction.

In the field of data science, the data gravitation (Peng et al., 2009) simulates the universal gravitation. Specifically, let the mass of each feature in target dataset be M, then the mass of target dataset $D_T \in \mathbb{R}^{n_t \times d}$ can be expressed as $n_t * d * M$, and the mass of a source instance $x_i^s \in \mathbb{R}^{1\times d}$ with feature weighted similarity s_i^{fw} can be represented as $s_i^{fw} * d * M$. Thus, in this paper, the feature weighted instance weight of ith source instance x_i^s with feature weighted similarity s_i^{fw} is defined as follows:

$$w_i^{fw} = \frac{\left(s_i^{fw} * d * M\right)(n_t * d * M)}{\left(\sum_{k=1}^{d}\theta_k - s_i^{fw} + 1\right)^2} \propto \frac{s_i^{fw}}{\left(\sum_{k=1}^{d}\theta_k - s_i^{fw} + 1\right)^2} \tag{2}$$

where s_i^{fw} denotes the feature weighted similarity of ith source instance as shown in Eq. (1).

3.2.5. Feature weighted posterior probability

Although the idea of feature weighted posterior probability has been proposed in previous study (see CFIW-TNB Zou et al., 2021a for details), we propose a novel form of feature weighted posterior probability.

Based on the training dataset and feature weighted instance weight, the prior probability of each class c is calculated:

$$P(c) = \frac{\sum_{i=1}^{n_s} w_i^{fw}\sigma\left(c_i, c\right) + 1}{\sum_{i=1}^{n_s} w_i^{fw} + n_c} \tag{3}$$

where c_i denotes the actual class label of ith source instance, n_s represents the number of source instances, n_c means the number of different classes ($n_c = 2$ in this paper), and $\sigma(x, y) = 1$ when x equals to y, otherwise $\sigma(x, y) = 0$.

Given a target instance x_j^t, the conditional probability of kth feature a_{jk}^t in x_j^t with respect to the given class c is calculated as follows:

$$P(a_{jk}^t \mid c) = \frac{\sum_{i=1}^{n_s} w_i^{fw}\sigma\left(a_{ik}^s, a_{jk}^t\right)\sigma\left(c_i, c\right) + 1}{\sum_{i=1}^{n_s} w_i^{fw}\sigma\left(c_i, c\right) + n_k} \tag{4}$$

where a_{ik}^s represents the value of kth feature of ith source instance, n_k denotes the number of unique values of kth feature in source dataset, and w_i^{fw} is the weight of ith source instance.

Based on the prior probability and conditional probability, the feature weighted posterior probability is defined as follows:

$$c\left(x_i^t\right) = \underset{c\in\{0,1\}}{argmax} P\left(c\left|x_i^t\right.\right) = \underset{c\in\{0,1\}}{argmax} \frac{P(c)\prod_{k=1}^{d}\left(P\left(a_{ik}^t\left|c\right.\right)\right)^{exp(\theta_k)}}{\sum_{c\in\{0,1\}} P(c)\prod_{k=1}^{d}\left(P\left(a_{ik}^t\left|c\right.\right)\right)^{exp(\theta_k)}} \quad (5)$$

where θ_k denotes the importance of kth feature. When calculating the conditional probability and the feature-weighted posterior probability, all the numeric features in the source and target datasets are first discretized using MDL-based discretization method (Fayyad and Irani, 1993).

3.2.6. Adaptive adjustment strategy

The reason that we propose the adaptive adjustment strategy is to achieve a highly robust CPDP model by considering both unsupervised CPDP and supervised CPDP. Here, the robustness of CPDP model refers to the ability to keep high-performance and stability of a defect prediction model even in extreme situations (e.g., the defect ratio of source dataset is extremely high or the size of source dataset is very small).

Our adaptive adjustment strategy includes two aspects: defective ratio based adaptive adjustment and dataset size based adaptive adjustment.

(1) Defective ratio based adaptive adjustment strategy

For a labeled source dataset, the defective ratio (DefRatio for short) refers to the percentage of defective instances in this dataset.

According to the benchmark datasets (see Section 4.2), we can see that the DefRatio of most datasets are much smaller than 0.5. If the defective ratio of a source dataset is too larger (e.g., $>$ 0.8), we think that the defect knowledge learned from this source dataset is difficult to transfer to other defect dataset. In this case, we use the following method to perform prediction on the target dataset.

If DefRatio is larger than α, we directly use the unsupervised CPDP model ManualDown (Zhou et al., 2018) to predict the label of target dataset without using the source dataset. ManualDown assumes that large sized instances are more likely to be defective. Specifically, we first rank target instances in descending order according to the metric Lines of Code (LOC). We then classify the top k% (in this paper, we use the default value 50%) instances in the ranking list as defective and the remaining instances as non-defective.

(2) Dataset size based adaptive adjustment strategy

Obviously, if the number of instances of given source dataset is very small, it is very hard to learn useful defect knowledge from source dataset and then use it on the target dataset. However, what is kind of defect dataset can be regarded as 'small' dataset, which is an open problem. Here, we suggest that if the number of instances of a dataset is smaller than N, this dataset is a small dataset. We suggest that the actual value of N should be smaller than 100.

With respect to small source dataset, we also directly use the unsupervised defect prediction model ManualDown to predict the label of target dataset.

In summary, our ARRAY combines the supervised CPDP model (i.e., the proposed triple feature-weighted transfer naive Bayes) and the unsupervised CPDP model (i.e., ManualDown) by using the proposed adaptive adjustment strategy. Specifically, if the defective ratio of source dataset is too large or the size of source dataset is too small, the supervised CPDP model is very difficult to learn useful defect knowledge and then transfer it to the target dataset. In this case, we just use the unsupervised defect prediction model, i.e., ManualDown, to predict the label of target data. Otherwise, we will build a supervised CPDP model, i.e., triple feature-weighted transfer naive Bayes.

4. Experimental design

4.1. Research question

RQ1: How much improvement can ARRAY obtain compared with previous CPDP models?

Motivation. We proposed a novel CPDP-CM method ARRAY. Therefore, it is necessary to evaluate its performance and investigate whether ARRAY can outperform the existing CPDP models.

Approach. We perform experiments on the benchmark datasets (see Section 4.2 for details) and evaluate the predictive performance in terms of F1, AUC, and MCC. We compare ARRAY with state-of-the-art CPDP methods including CPDP-CM and HDP models (see Section 4.3). The evaluation method is presented in Section 4.5.1. We also conduct statistical test (see Section 4.6) to demonstrate whether there is significant difference between ARRAY and the baselines.

RQ2: Does ARRAY significantly outperform deep semantic feature based CPDP model?

Motivation. Deep semantic features have been proposed to build CPDP models. However, there is evidence to show that whether traditional metrics based CPDP model can outperform the deep semantic feature based CPDP model. Therefore, we want to investigate that whether ARRAY outperforms the deep semantic features based CPDP models, which use deep learning techniques to automatically learn semantic features from the abstract syntax tree of project source code and then build CPDP models.

Approach. DBN-CP (Wang et al., 2020) is the most famous deep semantic feature based CPDP model. We will compare ARRAY with DBN-CP in terms of MCC, AUC, and F1. It is important to note that our ARRAY is a traditional metrics based CPDP method, but DBN-CP is deep semantic features based CPDP method. To ensure the fairness of comparative experiments, we must ensure that they are trained on the same modules and simultaneously tested on the same modules. To this end, we first need to find the project source code of benchmark datasets, because the benchmark datasets are traditional metrics based datasets. Unfortunately, we just found the project source code of PROMISE projects and the source code of prop-6 and tomcat is not found. Furthermore, the csv file of project synapse-1.2 does not include the information of module name and module path. Thus, only 9 projects' source code are valid. Therefore, we perform experiments on 9 PROMISE projects, which are ant-1.3, camel-1.6, ivy-2.0, jedit-4.1, log4j-1.2, poi-2.0, velocity-1.4, xalan-2.4, and xerces-1.2. We also take the same model evaluation settings and statistical test methods as in RQ1.

4.2. Benchmark datasets

A total of 34 defect datasets from five communities including AEEEM (D'Ambros et al., 2012), PROMISE (Menzies et al., 2015), ReLink (Wu et al., 2011), NASA (Menzies et al., 2015), and JIRA (Yatish et al., 2019) are used as the benchmark datasets. They have been widely used in previous studies (Li et al., 2018; Nam et al., 2018; Yu et al., 2017; Tong et al., 2021). Table 1 shows the statistics of these datasets. Owing to the space limitation, the details of metrics are not presented here.

Each dataset in AEEEM consists of 61 software metrics, which are 17 source code metrics (i.e., CK metrics Kemerer and Chidamber, 1994 and other 11 object-oriented metrics), 17 entropy-of-source-code metrics, 17 churn-of-source-code metrics, 5 entropy-of-change metrics, and some other metrics.

Table 1
34 benchmark datasets used in this study.

Group	Dataset	# of metrics	# of instances	
			All	Defective (%)
AEEEM (D'Ambros et al., 2012)	EQ	61	324	129(39.81%)
	JDT	61	997	206(20.66%)
	Lucene	61	691	64(9.26%)
	Mylyn	61	1862	245(13.16%)
	PDE	61	1497	209(13.96%)
PROMISE (Menzies et al., 2015)	ant-1.3	20	125	20(16%)
	camel-1.6	20	965	188(19.48%)
	ivy-2.0	20	352	40(11.36%)
	jedit-4.1	20	312	79(25.32%)
	log4j-1.2	20	205	189(92.2%)
	poi-2.0	20	314	37(11.78%)
	prop-6	20	660	66(10%)
	synapse-1.2	20	269	86(31.97%)
	tomcat	20	858	77(8.97%)
	velocity-1.4	20	196	147(75%)
	xalan-2.4	20	723	110(15.21%)
	xerces-1.2	20	440	71(16.14%)
ReLink (Wu et al., 2011)	Apache	26	194	98(50.52%)
	Safe	26	56	22(39.29%)
	Zxing	26	399	118(29.57%)
NASA (Menzies et al., 2015)	CM1	37	327	42(12.84%)
	JM1	21	7782	1672(21.49%)
	KC1	21	1183	314(26.54%)
	MW1	37	253	27(10.67%)
	PC1	37	705	61(8.65%)
	PC2	36	745	16(2.15%)
	PC3	37	1077	134(12.44%)
JIRA (Yatish et al., 2019)	activemq-5.0.0	65	1884	293(15.55%)
	derby-10.5.1.1	65	2705	383(14.16%)
	groovy-1..._1	65	821	70(8.53%)
	hbase-0.94.0	65	1059	218(20.59%)
	hive-0.9.0	65	1416	283(19.99%)
	jruby-1.1	65	731	87(11.9%)
	wicket-1.3...a2	65	1763	130(7.37%)

PROMISE consists of multiple open-source JAVA projects such as ant, ivy, and xalan, where each defect dataset includes 20 software metrics including CK metrics (Kemerer and Chidamber, 1994), QMOOD metrics (Bansiya and Davis, 2002), and code complexity metrics.

ReLink includes three open-source projects (i.e, Apache, Safe, and Zxing) and each defect dataset collected from the corresponding project has 26 complexity metrics.

Most NASA projects are programmed by C language, and we just consider C language projects expect for the projects programmed by other language. Note that these datasets have 20 common metrics. When building CPDP models, only these 20 common metrics are used.

Each dataset in JIRA contains 65 software metrics: 54 code metrics, 5 process metrics, and 6 ownership metrics. Each module in JIRA refers to a java file.

4.3. Baselines

To demonstrate the superiority of ARRAY, seven state-of-the-art CPDP models are used as the baselines: HDP-KS (Nam et al., 2018), FMT (Yu et al., 2017), CFIW-TNB (Zou et al., 2021a), ManualDown (Zhou et al., 2018), DMDAJFR (Zou et al., 2021b), and DBN-CP (Wang et al., 2020).

(1) HDP-KS. It is the most famous HDP model, which uses Kolmogorov–Smirnov test to match up the metrics between source and target datasets.

(2) FMT. Differing from HDP-KS, FMT uses the distance of distribution curve between source and target datasets to match up the metrics.

Table 2
Confusion matrix.

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

(3) CFIW-TNB. CFIW-TNB, proposed by Zou et al. (2021a) (2021), uses both feature transfer and instance transfer for CPDP.

(4) ManualDown. It directly classifies target dataset according to module size without using source dataset. Owing to its good performance, Zhou et al. (2018) recommended future CPDP studies should consider it as a baseline.

(5) DMDAJFR. It, proposed by Zou et al. (2021b), uses denoising autoencoders to jointly learn the global and local feature representations of source and target datasets simultaneously.

(6) BurakMHD. It was proposed by Bhat and Farooq (2022). BurakMhd uses Hamming distance to compute the similarity and then selects the similar source instances as the training dataset.

(7) DBN-CP. Wang et al. (2020) leveraged the deep belief network (DBN) to automatically learn semantic features from project source code and then built a CPDP model called DBN-CP.

4.4. Performance evaluation measures

Three well-known performance measures including Matthews correlation coefficient (MCC), the area under the receiver operating characteristic curve (AUC), and F1-score (F1 for short) are utilized. These measures have been widely used in previous studies (Chen et al., 2015; Ryu and Baik, 2016; Li et al., 2018; Zhang et al., 2016; Song et al., 2019).

By convention, the defective modules are regarded as positive class samples and the non-defective modules as negative class ones. Based on this consensus, aforementioned measures can be calculated according to the confusion matrix as shown in Table 2.

Firstly, the probability of detection (PD, a.k.a. recall and true positive rate) and the probability of false alarm (PF, a.k.a. false positive rate) are separately defined as follows:

$$PD = \frac{TP}{TP + FN}; PF = \frac{FP}{FP + TN} \tag{6}$$

F1 is the harmonic mean of PD and Precision, which is defined as follows:

$$F1 = \frac{2 * Precision * PD}{(Precision + PD)} \tag{7}$$

where *Precision* is obtained by dividing *TP* by $(TP + FP)$.

MCC (Matthews, 1975) is an overall performance measure for taking *TP*, *TN*, *FP*, and *FN* into consideration. The range of MCC value is $[-1,1]$, where 1 denotes a perfect prediction, and -1 indicates complete disagreement between actual and predicted values. MCC is defined as:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{8}$$

AUC refers to the area under of receiver operating characteristic (ROC) curve which is plotted in a 2-D space with PF as the *x*-axis and PD as the *y*-axis. Its range is [0, 1]. The bigger the AUC is, the better the prediction performance is. AUC is a widely used measure, because it is rarely affected by class-imbalance problem according to Tantithamthavorn et al. (2020).

4.5. Experimental settings

4.5.1. Model evaluation method

In WPDP, *k*-fold cross-validation (CV) (Hastie et al., 2009) is the most commonly used model evaluation technique. However, *k*-fold CV is not suitable for evaluating CPDP models, because in CPDP scenario the distributions of source dataset (training dataset) and target dataset (testing dataset) are different.

In this paper, a total of 34 datasets from five groups are used as the benchmark datasets (see Section 4.2). Datasets in different groups have different software metrics, but datasets in the same group have the same metric sets except NASA datasets. NASA datasets have 20 common metrics which are selected to build and evaluate CPDP models. For any given group, one dataset is selected as the source data and then another dataset in this group is selected as the target dataset. In this way, we can obtain 242 ($5*4+12*11+3*2+7*6+7*6$) combinations of the source and target datasets. With respect to any combination, e.g., EQ⇒JDT, the simplest way to evaluate a CPDP model is that train the model on EQ and then use the model on JDT for only once. However, this way may cause two potential problems. Specifically, on one hand, it may result in over-pessimistic or over-optimistic evaluation on the model's performance. On the other hand, we cannot investigate the performance stability (e.g., standard deviation) of CPDP models on each combination. To avoid these potential problems and alleviate the effect of randomness: (1) for any combination, we first randomly select 90% instances from the source dataset as the training data to train ARRAY and next use the trained model on the target dataset as done in previous studies (Li et al., 2018; Tong et al., 2021). The reason of random selection of 90% source instances is that it can not only ensure the training data is not always the same, but also 90% instances can represent the source dataset well. (2) for each CPDP method, we repeat above steps 30 times (30 is the minimum needed sample size to satisfy the central limit theorem) and calculate the average performance on each combination. Owing to the space limitation, we cannot report the results of all 242 combinations. We instead merge the results of the combinations that have the same target dataset and report the average performance.

Owing to the space limitation, we do not present the performance of CPDP models on each combination. We just combine the experimental results of the combinations having the same target dataset and then report the average performance of the corresponding model on the target dataset.

4.5.2. Parameter settings

Firstly, As we described in Section 3, ARRAY will use SMOTE (Chawla et al., 2002) to alleviate the class-imbalance problem of the source dataset. SMOTE has two key parameters P and K. P means the percentage of SMOTE instances to create. K denotes the number of the nearest neighbors (in this paper, we just use the default value 5). Denote the number of defective and non-defect modules in the original source dataset as *numPos* and *numNeg*, respectively. Suppose the ideal ratio of number of defective modules to non-defective ones is *ratio* after SMOTE. Then we set the value of P to be $round((numNeg*ratio-numPos)/numPos)*100$, where $round(X)$ rounds each element of X to the nearest integer. For example, if the source dataset have 100 positive instances and 300 negative instances, and we hope the final ratio of positive instances to the negative ones after SMOTE to be 1. And then we can obtain that $P = round((300*1-100)/100)*100 = 200$.

Secondly, with respect to adaptive adjustment strategy, α and N are empirically set to 0.45 and 80, respectively.

With respect to the parameters of baselines, we just use the same parameters as that in the original papers.

4.6. Statistical test methods

Two statistical test methods including Wilcoxon signed-rank test (at significance level 0.05) (Wilcoxon, 1945) and Scott-Knott effect size difference (ESD) test (at significance level 0.05) (Tantithamthavorn et al., 2017) are used to perform statistical testing.

Wilcoxon signed-rank test is a non-parametric pair-wise test. The reasons of using Wilcoxon signed-rank test mainly includes two aspects: (1) The parametric pair-wise statistical test methods usually assume that the sample must follow a specific distribution, e.g., t-test requires that the sample is normally distributed. However, it cannot be guaranteed that the defect data must obey the normal distribution. Wilcoxon signed-rank test does not assume that the data must follow a specific distribution. (2) Wilcoxon signed-rank test has been widely used in the previous software defect prediction studies (Nam et al., 2018; Yu et al., 2017; Ni et al., 2022). To control for false discovery, We further utilize the Benjamini–Hochberg procedure (Martínez-Cagigal, 2021) to adjust p-values. Win/Tie/Lose evaluation is used to analyze the performance, which is widely utilized in many previous studies (Zhou et al., 2018; Nam et al., 2018; Liu et al., 2019).

If the results of Wilcoxon signed-rank test show that ARRAY significantly outperforms the given baseline, we further compute Cliff's δ (Cliff, 2014). Cliff's δ is a non-parametric effect size measure to quantify the magnitude of difference. The magnitude is evaluated as follows (Cliff, 2014): Negligible ($|\delta| < 0.147$), Small ($0.147 \leq |\delta| < 0.330$), Medium ($0.330 \leq |\delta| < 0.474$), and Large ($|\delta| \geq 0.474$).

Scott-Knott ESD test (Tantithamthavorn et al., 2017) is a novel multiple comparison test (a.k.a. post-hoc test) method, which has not the overlap problem caused by other multiple comparison test methods, e.g., Nemenyi's post-hoc test. Scott-Knott ESD test uses hierarchical cluster analysis to divide different methods into significantly distinct groups where the effect size is non-negligible. Scott-Knott ESD test have been widely used in previous software defect prediction studies (Li et al., 2019b; Tantithamthavorn et al., 2019; Pascarella et al., 2020).

5. Experimental results

5.1. Results of RQ1: How much improvement does ARRAY obtain compared with previous cross-project defect prediction models?

Results. Table 3 presents the comparison results of ARRAY and the baselines in MCC in the fashion of *mean*. The best value on each target dataset is in boldface. All values are in percentage (%) except for the last two rows, i.e., 'Improvement' and 'Win/Tie/Lose'. From the table, we can see that: (1) The MCC of our ARRAY ranges from 0.058 (on velocity-1.4 and log4j-1.2) to 0.431 (on hive-0.9.0) across all benchmark datasets. On most datasets, ARRAY obtained the best performance. (2) On average, ARRAY achieved the best performance (0.261) and improved the performance over the baselines by 18.4% (for ManualDown) ∼ 177.8% (for HDP-KS) in terms of MCC. (3) ARRAY always significantly performed better than the baselines with at least medium effect size on most datasets.

Table 4 presents the comparison results of ARRAY and the baselines in AUC in the fashion of *mean*. The best value on each target dataset is in boldface. From the table, we notice that: (1) The AUC of ARRAY varies from 0.515 (on xerces-1.2) to 0.871 (on jruby-1.1). On most datasets, ARRAY always obtained the best performance. (2) On average, ARRAY obtained the largest AUC (i.e., 0.712) and improved the performance over the baselines by 6.2% (for FMT)∼ 29.8% (for DMDAJFR). (3) Although ARRAY lost

Table 3
MCC comparisons of ARRAY and the baselines. The best value is in boldface.

Target	ARRAY	HDP-KS Nam et al. (2018)	FMT Yu et al. (2017)	CFIW-TNB Zou et al. (2021a)	ManualDown Zhou et al. (2018)	DMDAJFR Zou et al. (2021b)	BurakMHD Bhat and Farooq (2022)
EQ	29.9	20.0(L)	27.1(S)	21.4(L)	25.3(L)	10.1(L)	**32.4**
JDT	38.9	25.3(L)	**41.4**	19.7(L)	29.5(L)	20.9(L)	30.5(L)
Lucene	27.1	24.1	**27.2**	26.3	11.1(L)	6.9(L)	16.3(L)
Mylyn	**21.8**	13.6(L)	20.4(S)	14.5(L)	19.2(L)	5.1(L)	7.0(L)
PDE	**23.6**	14.7(L)	23.2	13.7(L)	22.7(M)	9.5(L)	16.8(L)
ant-1.3	**35.6**	4.9(L)	13.5(L)	8.0(L)	35.3(S)	16.5(L)	19.3(L)
camel-1.6	**12.7**	1.6(L)	10.4(S)	2.5(L)	8.9(L)	3.7(L)	8.1(L)
ivy-2.0	**30.9**	10.1(L)	20.8(M)	6.8(L)	27.3(M)	15.2(L)	16.7(L)
jedit-4.1	**38.9**	6.7(L)	19.8(L)	1.7(L)	36.4(M)	15.7(L)	22.4(M)
log4j-1.2	**5.8**	2.7(M)	3.5(S)	2.0(L)	−5.7(L)	4.0(S)	3.4(S)
poi-2.0	**15.8**	5.8(L)	13.8(N)	6.9(M)	10.6(L)	7.1(L)	6.6(M)
prop-6	16.6	5.2(L)	8.8(L)	6.2(L)	**17.8**	6.7(L)	10.6(M)
synapse-1.2	27.7	3.1(L)	9.2(L)	6.4(L)	**31.6**	12.6(L)	16.7(M)
tomcat	**25.7**	7.8(L)	20.2(N)	3.9(L)	23.4(M)	11.8(L)	16.7(S)
velocity-1.4	**5.8**	−5.4(L)	−3.3(L)	−9.2(L)	−11.8(L)	−4.4(L)	−1.8(S)
xalan-2.4	30.1	9.5(L)	17.0(L)	4.8(L)	**33.1**	12.8(L)	17.3(L)
xerces-1.2	**7.2**	1.6(M)	4.7(S)	6.9	−1.5(L)	2.7(M)	0.0(L)
Apache	38.1	17.1(L)	29.3(M)	9.2(L)	**47.4**	21.2(L)	18.8(L)
Safe	40.5	33.7	41.9	29.5(L)	**43.9**	20.5(L)	16.6(N)
Zxing	**26.3**	6.4(L)	8.7(L)	2.3(L)	**26.3**	11.0(L)	12.9(L)
CM1	17.6	3.7(L)	12.0(L)	7.9(L)	**18.4**	0.7(L)	12.4(N)
JM1	**20.4**	5.4(L)	18.4(M)	1.4(L)	20.1(S)	0.9(L)	15.8(L)
KC1	20.7	1.3(L)	17.7(M)	−0.2(L)	**21.1**	4.8(L)	4.6(L)
MW1	**21.5**	3.1(L)	20.9	13.4(L)	17.6(L)	6.1(L)	9.1(L)
PC1	19.6	7.4(L)	15.6(L)	14.0(L)	**19.7**	4.6(L)	9.6(L)
PC2	**16.3**	4.2(L)	14.1(S)	2.2(L)	11.2(L)	7.1(L)	7.7(L)
PC3	**24.1**	4.2(L)	13.8(L)	−1.2(L)	22.5(L)	8.8(L)	19.9(S)
activemq-5.0.0	**41.6**	5.3(L)	27.4(L)	11.0(L)	26.5(L)	14.0(L)	31.8(L)
derby-10.5.1.1	**30.8**	6.3(L)	22.6(L)	1.5(L)	25.9(L)	−0.1(L)	22.6(S)
groovy-1_6_BETA_1	**24.2**	16.5(L)	20.4(M)	13.3(L)	17.0(L)	12.1(L)	12.4(L)
hbase-0.94.0	**34.1**	13.0(L)	18.6(L)	−1.6(L)	31.8(L)	14.4(L)	18.9(L)
hive-0.9.0	**43.1**	14.8(L)	28.4(L)	5.3(L)	31.9(L)	13.2(L)	16.6(L)
jruby-1.1	**42.1**	14.2(L)	39.2(S)	19.1(L)	29.9(L)	22.2(L)	28.7(L)
wicket-1.3.0-beta2	**31.1**	11.1(L)	23.9(L)	9.3(L)	23.8(L)	7.3(L)	23.7(L)
Average	26.1	9.4	19.1	8.2	22	9.6	15.3
Improvement	–	177.8%	36.2%	217.7%	18.4%	172.1%	70.1%
Win/Tie/Lose	–	32/2/0	29/4/1	32/2/0	25/4/5	34/0/0	33/0/1

Note: (1) Cliff's δ magnitude: L-Large, M-Medium, S-Small, N-Negligible. (2) All values are in percentage (%) except the last two rows.

Table 4
AUC comparisons of ARRAY and the baselines. The best value is in boldface.

Target	ARRAY	HDP-KS Nam et al. (2018)	FMT Yu et al. (2017)	CFIW-TNB Zou et al. (2021a)	ManualDown Zhou et al. (2018)	DMDAJFR Zou et al. (2021b)	BurakMHD Bhat and Farooq (2022)
EQ	**73.1**	66.7(L)	70.1(L)	65.8(L)	62.6(L)	53.4(L)	67.5(L)
JDT	**77.8**	70.0(L)	76.9	70.2(L)	68.1(L)	61.1(L)	68.6(L)
Lucene	72.9	69.6(M)	**75.8**	67.4(M)	59.5(L)	54.2(L)	66.6(L)
Mylyn	64.0	62.4	**65.9**	59.3(L)	64.2	51.7(L)	57.4(L)
PDE	70.7	62.5(L)	**71.6**	64.3(L)	66.3(L)	54.7(L)	64.0(L)
ant-1.3	**78.7**	64.1(L)	65.7(L)	64.3(L)	74.0(L)	59.2(L)	64.7(L)
camel-1.6	**58.2**	53.0(L)	57.7	53.5(L)	55.5(L)	51.3(L)	56.6(S)
ivy-2.0	**77.9**	63.0(L)	69.9(M)	62.3(L)	71.3(L)	58.8(L)	65.3(L)
jedit-4.1	**76.8**	62.9(L)	66.7(L)	58.9(L)	70.9(L)	57.0(L)	63.2(L)
log4j-1.2	**58.4**	53.2(L)	55.6(S)	56.0(S)	44.6(L)	53.0(L)	55.1(S)
poi-2.0	**64.2**	55.3(L)	60.1(S)	56.1(L)	58.0(L)	54.0(L)	56.1(M)
prop-6	**65.4**	56.0(L)	60.2(N)	59.9(M)	63.0(M)	53.0(L)	59.9(M)
synapse-1.2	**68.2**	61.1(S)	61.5(M)	60.6(M)	66.7(M)	54.7(L)	61.0(S)
tomcat	**74.7**	61.4(M)	69.8(N)	59.6(L)	69.8(L)	58.2(L)	66.3(S)
velocity-1.4	**51.9**	50.2	49.9	50.9	43.3(L)	48.2(S)	51.6
xalan-2.4	**74.9**	63.4(L)	65.2(L)	58.7(L)	72.7(L)	57.0(L)	63.5(L)
xerces-1.2	**51.5**	50.3(S)	47.4(L)	49.3(S)	49.0(M)	51.1	48.2(L)
Apache	65.5	67.1	70.7	54.3(L)	**73.7**	60.5(M)	59.4(N)
Safe	72.7	**76.0**	75.4	66.1(L)	72.5(S)	59.3(L)	52.8(L)
Zxing	**64.3**	57.7(N)	56.3(L)	52.7(L)	**64.3**	55.3(L)	60.6(L)

(*continued on next page*)

to FMT on 4 out of 34 datasets, ARRAY significantly outperformed FMT on 25 datasets. AFWTNB always significantly outperformed the baselines except FMT with at least small effect size on almost all datasets.

Table 5 presents the comparison results of ARRAY and the baselines in f1 in the fashion of *mean*. The best value on each target dataset is in boldface. From the table, we notice that: (1) The Balance of ARRAY ranges from 0.138 (on PC2) to 0.724 (on

Table 4 (*continued*).

Target	ARRAY	HDP-KS Nam et al. (2018)	FMT Yu et al. (2017)	CFIW-TNB Zou et al. (2021a)	ManualDown Zhou et al. (2018)	DMDAJFR Zou et al. (2021b)	BurakMHD Bhat and Farooq (2022)
CM1	**68.3**	61.5(S)	63.4(L)	62.9(L)	63.7(L)	50.3(L)	62.0(S)
JM1	**65.5**	57.3(M)	64.8(N)	50.5(L)	62.3(L)	50.4(L)	58.2(L)
KC1	62.8	50.8(L)	**63.8**	53.9(L)	62.0(S)	51.3(L)	53.3(M)
MW1	72.3	63.3(S)	69.7(S)	**72.6**	64.2(L)	53.3(L)	53.1(L)
PC1	**74.8**	61.5(M)	68.9(L)	56.8(L)	67.5(L)	52.6(L)	63.0(L)
PC2	**80.0**	70.3(N)	78.7(N)	55.2(L)	69.2(L)	56.3(L)	68.6(L)
PC3	**75.1**	61.0(L)	68.7(L)	55.5(L)	67.1(L)	54.6(L)	72.4(L)
activemq-5.0.0	**80.4**	58.8(L)	75.5(L)	73.5(L)	67.8(L)	57.0(L)	75.2(L)
derby-10.5.1.1	**78.2**	59.4(L)	69.3(L)	70.1(L)	68.5(L)	50.0(L)	65.6(L)
groovy-1_6_BETA_1	**75.2**	65.5(L)	70.4(L)	66.1(S)	65.5(L)	57.1(L)	59.1(L)
hbase-0.94.0	**77.3**	65.1(L)	68.7(L)	72.4(L)	69.7(L)	57.2(L)	58.5(L)
hive-0.9.0	**80.4**	65.6(L)	72.4(L)	67.1(L)	69.6(L)	54.8(L)	56.4(L)
jruby-1.1	**87.1**	65.8(L)	80.5(L)	81.5(L)	72.7(L)	61.5(L)	78.9(L)
wicket-1.3.0-beta2	**82.6**	59.9(L)	74.1(L)	75.3(L)	72.5(L)	54.2(L)	75.6(L)
Average	71.2	61.5	67.1	61.9	65.1	54.9	62
Improvement	–	15.8%	6.2%	15.1%	9.5%	29.8%	14.9%
Win/Tie/Lose	–	30/4/0	25/4/5	32/1/1	31/1/2	33/1/0	33/1/0

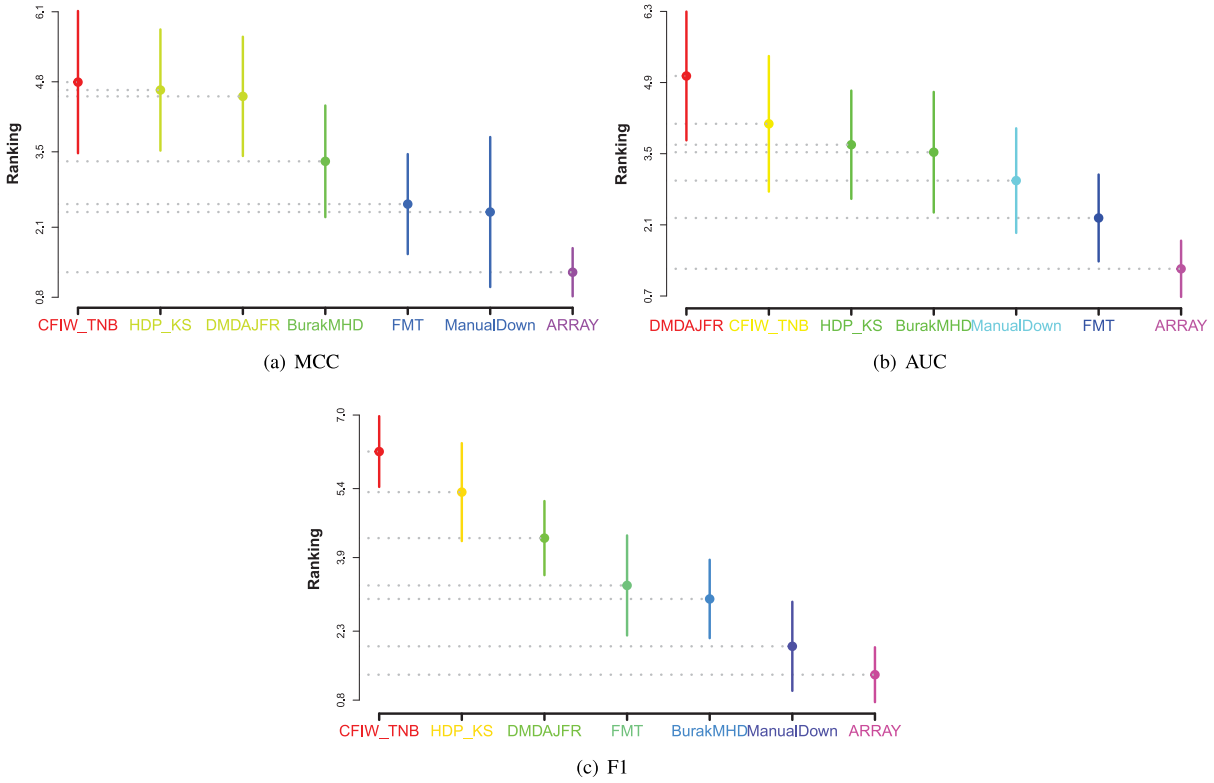
Table 5
F1 comparisons of ARRAY and the baselines. The best value is in boldface.

Target	ARRAY	HDP-KS Nam et al. (2018)	FMT Yu et al. (2017)	CFIW-TNB Zou et al. (2021a)	ManualDown Zhou et al. (2018)	DMDAJFR Zou et al. (2021b)	BurakMHD Bhat and Farooq (2022)
EQ	58.1	17.5(L)	36.0(L)	15.7(L)	62.1	28.2(L)	**62.4**
JDT	**53.0**	25.5(L)	51.4	9.3(L)	46.4(L)	39.8(L)	46.9(L)
Lucene	**34.6**	21.9(L)	33.6	14.1(L)	21.2(L)	17.0(L)	24.5(L)
Mylyn	**33.6**	17.9(L)	32.2(S)	7.2(L)	31.1(L)	18.8(L)	25.2(L)
PDE	**35.8**	18.7(L)	32.5(L)	6.8(L)	34.3(L)	22.3(L)	30.9(L)
ant-1.3	**45.5**	11.5(L)	22.6(L)	9.7(L)	43.9(S)	30.4(L)	34.3(L)
camel-1.6	32.9	10.2(L)	22.1(L)	3.8(L)	**34.5**	20.4(L)	31.0(M)
ivy-2.0	**37.7**	15.7(L)	31.3(S)	6.0(L)	33.0(L)	26.9(L)	29.2(L)
jedit-4.1	**56.6**	17.4(L)	32.2(L)	4.5(L)	54.9(M)	36.2(L)	45.3(L)
log4j-1.2	48.1	12.9(L)	19.6(L)	9.4(L)	**63.9**	31.5(L)	52.1
poi-2.0	**28.9**	11.3(L)	23.8(L)	5.6(L)	25.8(L)	22.2(L)	24.3(M)
prop-6	30.6	11.8(L)	17.6(L)	7.1(L)	**30.9**	20.8(L)	27.1(M)
synapse-1.2	53.9	13.4(L)	22.1(L)	7.8(L)	**57.9**	33.2(L)	48.1(S)
tomcat	**31.7**	14.0(L)	29.8(N)	2.8(L)	27.8(L)	22.9(L)	28.0(N)
velocity-1.4	42.5	12.4(L)	21.6(L)	0.1(L)	**55.2**	27.3(L)	52.1
xalan-2.4	42.1	16.9(L)	30.0(L)	4.7(L)	**42.5**	28.6(L)	34.4(L)
xerces-1.2	**27.4**	11.1(L)	22.0(L)	7.2(L)	25.6(L)	22.3(L)	24.7(L)
Apache	72.4	29.8(L)	52.4(L)	9.6(L)	**73.8**	59.3(L)	40.0(L)
Safe	66.7	42.6(L)	55.4(L)	25.2(L)	**68.0**	44.5(L)	42.1(L)
Zxing	**53.1**	37.8(L)	26.2(L)	11.1(L)	**53.1**	33.9(L)	36.7(L)
CM1	29.6	4.9(L)	20.6(L)	7.7(L)	**30.2**	12.6(L)	25.4(N)
JM1	39.9	4.6(L)	27.6(L)	0.1(L)	**41.4**	2.8(L)	22.3(L)
KC1	38.8	0.6(L)	27.6(L)	5.7(L)	**46.9**	12.0(L)	34.2(S)
MW1	**30.1**	1.9(L)	24.7(S)	12.5(L)	26.6(L)	15.1(L)	19.4(L)
PC1	**25.7**	9.9(L)	21.0(M)	5.7(L)	24.2(L)	13.7(L)	18.5(L)
PC2	**13.8**	5.3(L)	13.1	2.2(L)	7.3(L)	8.9(M)	7.8(L)
PC3	**34.2**	7.1(L)	22.5(L)	0.0(L)	31.8(L)	18.8(L)	28.8(S)
activemq-5.0.0	**51.5**	5.5(L)	30.9(L)	2.9(L)	39.7(L)	25.6(L)	43.5(L)
derby-10.5.1.1	**40.9**	9.2(L)	31.1(L)	0.2(L)	36.3(L)	0.0(L)	36.4(S)
groovy-1⇒ 6 ⇒BETA⇒1	**30.3**	19.2(L)	27.2(M)	5.2(L)	21.9(L)	18.5(L)	22.0(L)
hbase-0.94.0	**49.2**	13.8(L)	29.8(L)	0.0(L)	47.4(L)	30.1(L)	32.9(L)
hive-0.9.0	**55.7**	14.7(L)	38.7(L)	0.7(L)	47.8(L)	21.2(L)	33.8(L)
jruby-1.1	**47.8**	16.8(L)	45.9(S)	8.7(L)	35.8(L)	30.4(L)	37.0(L)
wicket-1.3.0-beta2	**36.0**	10.4(L)	28.3(L)	3.1(L)	24.2(L)	16.1(L)	26.9(L)
Average	41.4	14.5	29.5	6.5	39.6	23.9	33.2
Improvement	–	185%	40.4%	533.4%	4.5%	73.4%	24.9%
Win/Tie/Lose	–	34/0/0	31/3/0	34/0/0	21/3/10	34/0/0	31/0/3

Apache). On at least half of benchmark datasets, ARRAY obtained the best performance. (2) On average, ARRAY achieved the best performance in terms of F1 and improved the performance over the baselines by at least 4.5% (for ManualDown)∼ 533.4% (for CFIW-TNB). (3) ARRAY nearly always significantly outperformed the baselines except ManualDown on each dataset with at least medium effect size. Although ARRAY lost to ManualDown on 10 out of 34 benchmark datasets, ARRAY wan ManualDown on 21 datasets.

Figs. 2(a) and 2(b), and 2(c) show the results of Scott-Knott ESD test for ARRAY and the baselines in terms of MCC, AUC, and F1, respectively. In the figures, the smaller ranking means better performance for three measures. The dot denotes the mean value of ranking. Methods with the same color have no significant difference. Methods with different color are significantly different and the effect size of difference is non-negligible. From the figures, we can see that:

- With respect to MCC (see Fig. 2(a)), all methods are divided into five groups, i.e., (1) ARRAY, (2) ManualDown and FMT, (3) BurakMHD, (4) DMDRJFR, and (5) HDP-KS and CFIW-TNB, where ARRAY obtained the smallest average ranking. It means that ARRAY significantly performs better



(a) MCC (b) AUC

(c) F1

Fig. 2. The results of scott-knott esd test. Smaller ranking means better performance for three measures. The dot denotes the mean value of ranking across all target datasets. Methods with the same color have no significant difference. Methods having different color are significantly different and the effect size of difference is non-negligible.

than all baselines with non-negligible effect size across all benchmark datasets.

- With respect to AUC (see Fig. 2(b)), all six methods are divided into six different groups, i.e., (1) ARRAY, (2) FMT, (3) ManualDown, (4) HDP-KS and urakMHD, and (5) CFIW-TNB, and (6) DMDAJFR, where ARRAY obtained the smallest average ranking. It means that ARRAY significantly outperforms all baselines with non-negligible effect size across all benchmark datasets.
- With respect to F1 (see Fig. 2(c)), all methods are also divided into seven different groups, i.e., (1) ARRAY, (2) ManualDown, (3) BurakMHD, (4) FMT, (5) DMDAJFR and FMT, (6) HDP-KS, and (7) CFIW-TNB, where ARRAY still had the smallest average ranking value. It indicates that ARRAY significantly outperforms all baselines with non-negligible effect size across all benchmark datasets in terms of F1.

Answer to RQ1: Our ARRAY always significantly performs better than all baselines in terms of MCC, AUC, and F1 across all datasets according to the results of Scott-Knott ESD test. On average, ARRAY improves the performance over all baselines by at least 18.4%, 6.2%, and 4.5% in terms of MCC, AUC, and F1, respectively.

5.2. Results of RQ2: Does ARRAY significantly outperform the deep semantic based cross-project defect prediction model?

Results. Tables 6, 7, and 8 present the comparison results of ARRAY and DBN-CP in the fashion of *mean* in terms of MCC, AUC, and F1, respectively. The best value on each target dataset is in boldface.

Table 6
MCC comparisons of ARRAY and DBN-CP. The best value is in boldface.

Target	ARRAY	DBN-CP (Wang et al., 2020)
ant-1.3	**33.4**	20.5(M)
camel-1.6	**12.2**	6.9(S)
ivy-2.0	**29.2**	13.7(L)
jedit-4.1	**37.4**	10.9(L)
log4j-1.2	**6.6**	2.1(M)
poi-2.0	**15.2**	10.3(S)
velocity-1.4	**6.1**	−0.5(M)
xalan-2.4	**29.2**	12.3(L)
xerces-1.2	**6.4**	2.9(S)
Average	19.5	8.8
Improvement	–	122.1%
Win/Tie/Lose	–	9/0/0

Note: (1) Cliff's δ magnitude: L-Large, M-Medium, S-Small, N-Negligible. (2) All values are in percentage (%) except for the last two rows.

From Table 6, we notice that: (1) the MCC of ARRAY ranges from 0.061 (on velocity-1.4) to 0.374 (on jedit-4.1) across all 9 target datasets and that of DBN-CP ranges from −0.05 (on velocity-1.4) to 0.205 (on ant-1.3); (2) on average, our ARRAY obtained MCC as 0.195, which improves the performance over DBN-CP by 122.1%; (3) according to the result of 'Win/Tie/Lose', ARRAY significantly outperforms DBN-CP on all 9 datasets. Furthermore, ARRAY significantly outperforms DBN-CP with at least medium effect size on 6 out of 9 datasets.

From Table 7, we can see that: (1) the AUC of ARRAY ranges from 0.517 (on xerces-1.2) to 0.772 (on ant-1.3) across all 9 target datasets and that of DBN-CP ranges from 0.479 (on log4j-1.2) to 0.666 (on ant-1.3; 2) on average, our ARRAY obtained the AUC as 0.653, which improves the performance over DBN-CP by 16.5%; (3) according to 'Win/Tie/Lose', ARRAY significantly outperforms DBN-CP on 8 out of 9 datasets. Furthermore, the effect size of

Table 7
AUC comparisons of ARRAY and DBN-CP. The best value is in boldface.

Target	ARRAY	DBN-CP (Wang et al., 2020)
ant-1.3	**77.2**	66.6(L)
camel-1.6	**57.9**	53.8(L)
ivy-2.0	**76.8**	63.3(L)
jedit-4.1	**75.5**	58.0(L)
log4j-1.2	**59.1**	47.9(L)
poi-2.0	**63.4**	56.8(L)
velocity-1.4	**52.2**	48.8(S)
xalan-2.4	**74.3**	57.8(L)
xerces-1.2	51.7	**51.8**
Average	65.3	56.1
Improvement	–	16.5%
Win/Tie/Lose	–	8/1/0

Table 8
F1 comparisons of the ARRAY and DBN-CP. The best value is in boldface.

Target	ARRAY	DBN-CP (Wang et al., 2020)
ant-1.3	**43.8**	34.1(M)
camel-1.6	**33.3**	29.6(S)
ivy-2.0	**36.2**	25.6(M)
jedit-4.1	**55.7**	32.8(L)
log4j-1.2	**49.6**	32.6(L)
poi-2.0	**28.4**	24.1(M)
velocity-1.4	**45.1**	33.9(L)
xalan-2.4	**41.3**	29.2(L)
xerces-1.2	**27.4**	21.3(M)
Average	40.1	29.2
Improvement	–	37.1%
Win/Tie/Lose	–	9/0/0

difference between ARRAY and DBN-CP is large on 7 out of above 8 datasets.

From Table 8, we can see that: (1) the F1 of ARRAY ranges from 0.274 (on xerces-1.2) to 0.557 (on jedit-4.1) across all 9 target datasets and that of DBN-CP ranges from 0.213 (on xerces-1.2) to 0.341 (on ant-1.3); (2) on average, our ARRAY obtained the F1 as 0.401, which improves the performance over DBN-CP by 37.1%; (3) according to 'Win/Tie/Lose', ARRAY significantly outperforms DBN-CP on all 9 datasets. Furthermore, the effect size of difference between ARRAY and DBN-CP is at least medium on 8 out of 9 datasets.

> **Answer to RQ2:** ARRAY significantly outperforms DBN-CP on almost all datasets. On average, ARRAY improves the performance over DBN-CP by 122.1%, 16.5%, and 29.2% in terms of MCC, AUC, and F1, respectively. Thus, the deep semantic feature based CPDP model is not necessarily to outperform traditional metrics based CPDP model.

6. Discussion

6.1. Ablation study

As described in Section 3, our ARRAY is mainly characterized by feature weighting methods and adaptive strategy. Here, we want to investigate how feature weighting methods and adaptive strategy separately affect the performance of ARRAY. To this end, we compare ARRAY with two baselines which are TNB (Ma et al., 2012) and TFWTNB, where (1) TFWTNB is created by removing the adaptive adjustment strategy from ARRAY, and (2) TNB does not have feature weighted similarity, feature weighted instance weight and feature weighted posterior probability compared with TFWTNB. To save experiment time, experiments are just conducted on the 20 benchmark datasets from AEEEM, PROMISE, and ReLink. We use the validation method as described in Section 4.5.1. Owing to the space limitation, we just report MCC and AUC.

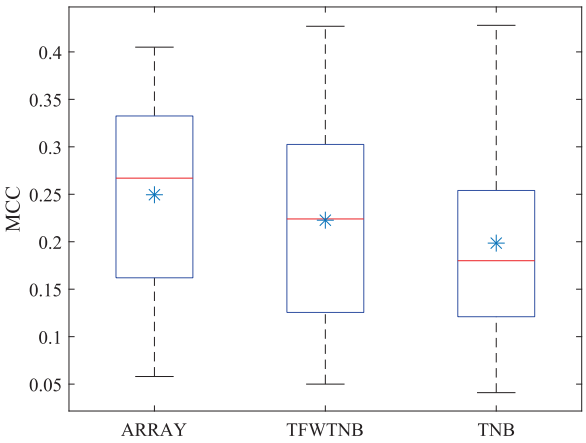


Fig. 3. The results of ablation study in terms of MCC.

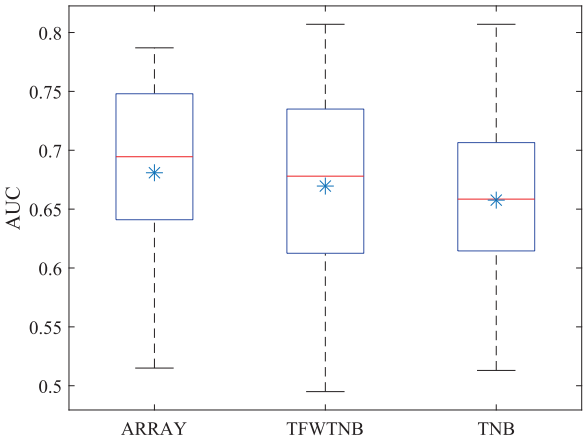


Fig. 4. The results of ablation study in terms of AUC.

Figs. 3 and 4 separately show the boxplots of MCC and AUC of different models. In the figures, the horizontal line in the box denotes the median value, and the asterisk is the mean value. From the figures, we can see that:

(1) ARRAY outperforms TFWTNB on both average and median value in terms of MCC and AUC. It means that the proposed adaptive strategy is useful for improving the performance of ARRAY.

(2) TFWTNB performs better than TNB in terms of both MCC and AUC no matter on average or median value. It indicates that the proposed feature weighting methods is helpful for improving the performance of CPDP models.

Figs. 5 and 6 show the results of Scott-Knott ESD test for ARRAY and the baselines in terms of MCC and AUC, respectively. We notice that:

(1) With respect to MCC (see Fig. 5), all methods are divided into three groups, i.e., (1) ARRAY, (2) TFWTNB, and (3) TNB, where ARRAY obtained the smallest average ranking and TNB got the largest average ranking. It means that both adaptive strategy and feature weighting methods can significantly improve the performance of CPDP models with non-negligible effect size.

(2) With respect to AUC (see Fig. 6), all methods are also divided into three groups, i.e., (1) ARRAY, (2) TFWTNB, and (3) TNB, where ARRAY obtained the smallest average ranking. It can be concluded that both adaptive strategy and feature

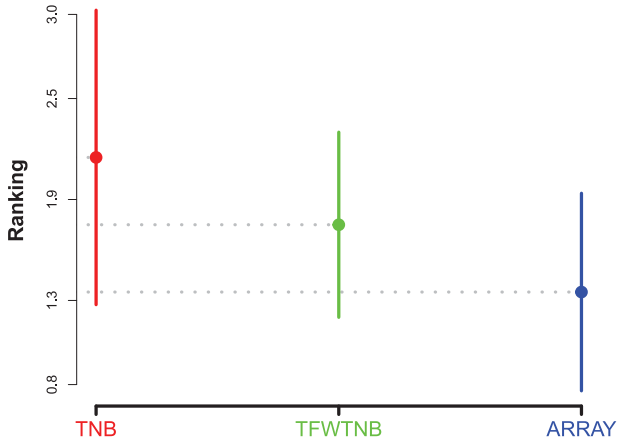


Fig. 5. The results of Scott-Knott ESD test for ablation study in terms of MCC.

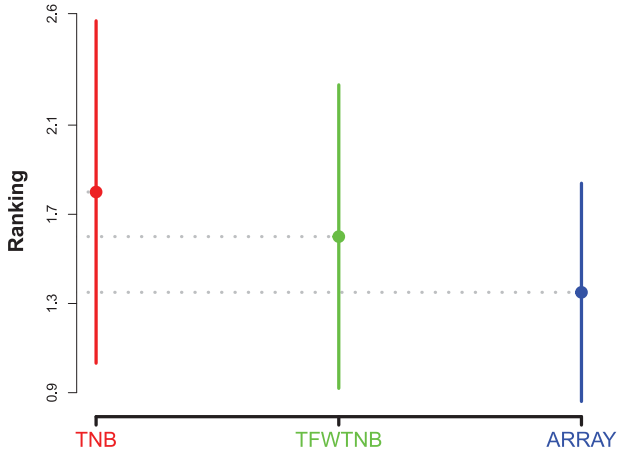


Fig. 6. The results of Scott-Knott ESD test for ablation study in terms of AUC.

weighting methods can significantly improve the performance of CPDP models with non-negligible effect size.

> **Conclusion:** Both feature weighting methods (feature weighted similarity, feature weighted instance weight, and feature weighted posterior probability) and adaptive adjustment strategy are useful for improving the performance of ARRAY with statistical significance.

6.2. Does ARRAY outperform WPDP models?

CPDP has shown great potential compared with WPDP when having no sufficient historical defect data according to previous studies (Turhan et al., 2009; Nam et al., 2018). In this section, we further compare ARRAY with WPDP models to demonstrate the necessity of considering CPDP.

For the sake of fairness, we must ensure that (1) they are compared on the same testing datasets, and (2) they are compared in the scenario of insufficient historical defect dataset, because CPDP is just considered when the labeled training data is insufficient. To this end, given a combination of the source and target datasets, we randomly select a small part ($x = 5\%$, 10% to simulate the scenario of insufficient historical defect dataset) of target instances as the training data to train WPCP methods and then take the remaining target instances as the testing data for both WPDP methods and ARRAY. ARRAY is trained on the source dataset. The

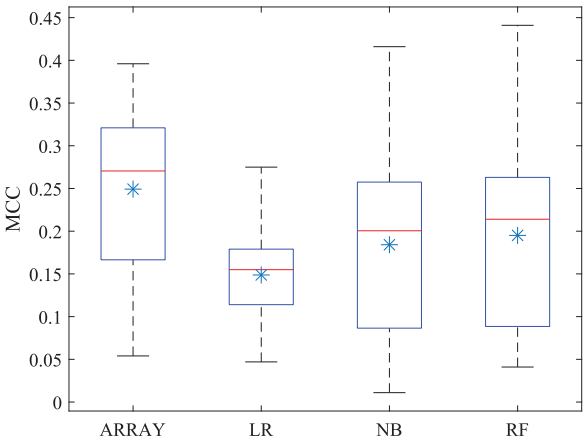


Fig. 7. Comparison result of ARRAY and WPCP models when $x = 5\%$.

reason that we let x be 5% and 10% is to simulate the WPDP scenario that having no sufficient training data. To eliminate the effect of randomness when constructing testing data, for any given combination of the source and target datasets, we repeat above process 20 times and report the average performance. To save experimental time, we just use the datasets from AEEEM, PROMISE, and ReLink as the benchmark datasets. For WPDP, we consider three famous machine learning methods, i.e., logistic regression (LR), naive Bayes (NB), random forest (RF). All WPDP models are implemented by WEKA (Witten et al., 2016) with default parameters. Owing to the space limitation, we just report MCC.

Figs. 7 and 8 show the boxplots of MCC for ARRAY and WPDP models (i.e., LR, NB, and RF) when $x = 5\%$ and $x = 10\%$, respectively. In the figures, the horizontal line in the box denotes the median, and the asterisk is the mean. From the figures, we can notice that:

(1) The performance of each of WPDP models when $x = 10\%$ is better than that of the same WPDP model when 5%. Therefore, it can be concluded that increasing the number of training data are helpful for improving the predictive performance of WPDP models.

(2) ARRAY always outperforms all WPDP models either $x = 5\%$ or $x = 10\%$ no matter on average or median. We can conclude that it is necessary to perform CPDP when having no sufficient training data for WPDP.

> **Conclusion:** When there is no sufficient training data to build WPDP models, it is very necessary to build CPDP models because ARRAY significantly outperforms the WPDP models.

6.3. Implication

The results of this paper have several implications for researchers and practitioners.

ARRAY enriches the cross-project defect prediction research by addressing the existing gaps. We compared with seven state-of-the-art CPDP methods. The experiment results show that ARRAY significantly outperforms these baselines in terms of AUC, MCC, and F1. The improvements are due to two aspects: (1) the feature-weighted similarity, feature-weighted instance weight, and feature-weighted posterior probability reduces the calculation bias of instance similarity, instance weight, and the posterior probability; (2) the adaptive adjustment strategy optimizes

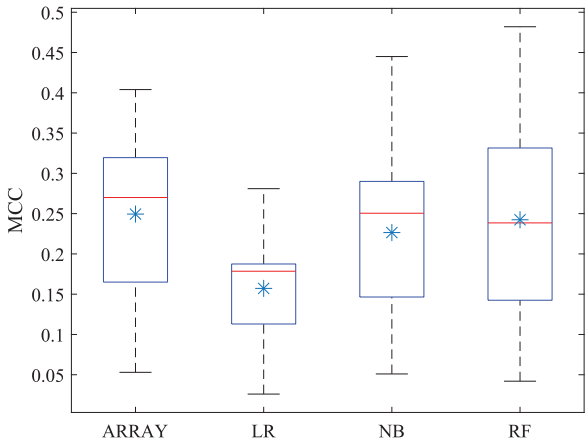


Fig. 8. Comparison result of ARRAY and WPCP models when $x = 10\%$.

model training by automatically selecting the most appropriate training approach based on the characteristics of the source dataset.

ARRAY highlights the need to build CPDP model by using labeled source dataset. Despite prior research indicating that unsupervised CPDP model without using source dataset outperform the CPDP models using source dataset (Zhou et al., 2018), our results show that ARRAY significantly outperforms the unsupervised CPDP model, ManualDown (Zhou et al., 2018). Therefore, it is necessary to consider utilizing the labeled source dataset in CPDP.

ARRAY highlights the necessity of conducting CPDP when there is no sufficient historical defect data to build WPDP models. Although previous studies show that CPDP model can be comparable to WPDP models sometimes (Nam et al., 2018), our experiment results show that ARRAY outperforms state-of-the-art WPDP models. Thus, it is very necessary to consider CPDP when there is no sufficient historical defect data for WPDP.

7. Threats to validity

In this section, we present the potential threats to the validity of our research.

7.1. Internal validity

Threats to the internal validity mainly refer to replication of baselines. Since the source code of all baselines except for DMDAJFR (Zou et al., 2021b) is not publicly available, we carefully implement these baselines according to the original studies. Although we have checked our implementations many times, we still cannot state that our replication are completely identical with that in the original studies. To reduce this threat, we have opened the benchmark datasets and the source code of both the proposed ARRAY and the baselines on GitHub.[1]

7.2. External validity

Threat to external validity refers to the generalization of the findings according to Xia et al. (2016). In this paper, we perform the experiments on 34 different defect datasets from five groups (i.e., AEEEM, PROMISE, ReLink, NASA MDP, and JIRA) and evaluate the prediction performance in terms of F1, AUC, and MCC. To demonstrate the superiority of ARRAY, both Wilcoxon signed-rank test and Scott-Knott ESD test are also used for statistical test. The experimental results show that ARRAY is more promising for CPDP compared with the baselines. However, we still cannot claim that the completely same findings can be obtained on other defect datasets. More defect datasets should be considered to reduce this threat.

7.3. Construct validity

Threats to construct validity in this paper mainly relate to the selection bias of the baselines and the performance measures. The baselines consist of the seven state-of-the-art methods, which are HDP-KS (Nam et al., 2018), FMT (Yu et al., 2017), CFIW-TNB (Zou et al., 2021a), ManualDown (Zhou et al., 2018), DMDAJFR (Zou et al., 2021b), and DBN-CP (Wang et al., 2020). The baselines contain the well-known deep semantic feature based CPDP model (i.e., DBN-CP), the most famous feature-matching based HDP models (i.e., HDP-KS and FMT), and the latest CPDP models (e.g., CFIW-TNB and DMDAJFR). Due to the space limitation, we cannot compare ARRAY with all previous CPDP methods. In this study, three commonly used performance measures including MCC, AUC, and F1 are utilized to evaluate the predictive performance, which have been widely used in previous studies (Chen et al., 2015; Li et al., 2018; Song et al., 2019; Ryu and Baik, 2016; Zhang et al., 2016). However, other overall measures, such as *G-measure* and *G-mean*, also have been used in several previous studies (Peters et al., 2013; Wang and Yao, 2013). Therefore, these threats exist in this study.

8. Conclusion

Cross-project defect prediction (CPDP) has drawn increasing attention of both academic and industry communities and many CPDP methods have been proposed. However, it is still a challenge to build a high-performance CPDP model owing to the inherent factors, especially the distribution difference between source and target datasets.

In this paper, we propose a novel high-performance cross-project defect prediction with common metrics method named ARRAY based on feature weighted similarity, feature weighted instance weight, and model adaptive adjustment. It is the first time to propose the concept feature weighted similarity, feature weighted instance weight and model adaptive adjustment in the field of software defect prediction. We perform experiments on 34 defect datasets and evaluate the performance in terms of F1, MCC, and AUC. To demonstrate the superiority of ARRAY, we compare ARRAY with seven state-of-the-art CPDP methods including HDP-KS (Nam et al., 2018), FMT (Yu et al., 2017), CFIW-TNB (Zou et al., 2021a), ManualDown (Zhou et al., 2018), DMDAJFR (Zou et al., 2021b), and DBN-CP (Wang et al., 2020) by using Wilcoxon signed-rank test and Scott-Knott ESD test, respectively. The experimental results show that (1) ARRAY significantly outperforms all baselines with non-negligible effect size in MCC, AUC, and F1 across all datasets according to the results of Scott-knott ESD test; (2) Both feature weighting methods and adaptive adjust strategy are crucial for improving the performance of CPDP model; (3) we are the first to demonstrate that the traditional metric based CPDP model can outperform the deep semantic feature based CPDP model.

In the future, we plan to perform ARRAY on more defect datasets to further validate its generalization ability. Moreover, we will try to combine ARRAY with deep learning.

[1] https://github.com/HNTong/ARRAY.git.

Table 9
The meanings of acronyms and abbreviations.

Acronyms/ Abbreviations	Meaning
ARRAY	Adaptive triple feature-weighted transfer naive Bayes
CPDP	Cross-project defect prediction
WPDP	Within-project defect prediction
CPDP-CM	Cross-project defect prediction with common metrics
HDP	Heterogeneous defect prediction
SDP	Software defect prediction
PD	Probability of detection
PF	Probability of false alarm
MCC	Matthews correlation coefficient
AUC	Area under roc curve
HDP-KS	Heterogeneous defect prediction with KSAnalyzer
FMT	Feature matching transfer
TNB	Transfer naive Bayes
CFIW-TNB	Correlation feature and instance weights TNB
DBN-CP	Deep belief network cross-project defect prediction
DMDAJFR	Joint feature representation with double marginalized denoising autoencoders
TCA	Transfer component analysis
MIC	maximal information coefficient
SMOTE	Synthetic minority oversampling technique
LR	Logistic regression
NB	Naive Bayes
RF	Random forest

CRediT authorship contribution statement

Haonan Tong: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Visualization. **Wei Lu:** Writing – review & editing. **Weiwei Xing:** Writing – review & editing, Visualization. **Shihai Wang:** Writing – review & editing, Validation.

Declaration of competing interest

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

Data availability

The data that has been used is confidential

Acknowledgments

We would like to thank the reviewers for their valuable comments to improve the quality of this paper. We also thank Zou for providing the source code of DMDAJFR (Zou et al., 2021b). This work is supported by the National Key Research and Development Program of China (No. 2021YFB2900704).

Appendix

Table 9 shows the meanings of acronyms and abbreviations used in this paper.

References

Albanese, D., Filosi, M., Visintainer, R., Riccadonna, S., Jurman, G., Furlanello, C., 2013. Minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers. Bioinformatics 29 (3), 407–408.

B, U.S., Sadam, R., 2023. How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model. J. Syst. Softw. 195, 111522. http://dx.doi.org/10.1016/j.jss.2022.111522.

Bansiya, J., Davis, C.G., 2002. A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28 (28), 4–17.

Bhat, N.A., Farooq, S.U., 2022. An improved method for training data selection for cross-project defect prediction. Arab. J. Sci. Eng. 47 (2), 1939–1954. http://dx.doi.org/10.1007/s13369-021-06088-3.

Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S., 2013. Multi-objective cross-project defect prediction. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. pp. 252–261. http://dx.doi.org/10.1109/ICST.2013.38.

Canfora, G., Lucia, A.D., Penta, M.D., Oliveto, R., Panichella, A., Panichella, S., 2015. Defect prediction as a multiobjective optimization problem. Softw. Test. Verif. Reliab. 25 (4), 426–459.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. SMOTE: Synthetic minority over-sampling technique. J. Artif. Int. Res. 16 (1), 321–357.

Chen, L., Fang, B., Shang, Z., Tang, Y., 2015. Negative samples reduction in cross-company software defects prediction. Inf. Softw. Technol. 62, 67–77. http://dx.doi.org/10.1016/j.infsof.2015.01.014.

Cliff, N., 2014. Ordinal Methods for Behavioral Data Analysis. Psychology Press.

Cruz, A.E.C., Ochimizu, K., 2009. Towards logistic regression models for predicting fault-prone code across software projects. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 460–463.

D'Ambros, M., Lanza, M., Robbes, R., 2012. Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empir. Softw. Engg. 17 (4–5), 531–577. http://dx.doi.org/10.1007/s10664-011-9173-9.

Fayyad, U.M., Irani, K.B., 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. Chambery, France, pp. 1022–1022.

Giray, G., Bennin, K.E., Köksal, Ö., Babur, Ö., Tekinerdogan, B., 2023. On the use of deep learning in software defect prediction. J. Syst. Softw. 195, 111537. http://dx.doi.org/10.1016/j.jss.2022.111537.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, second ed. Springer, New York, NY, pp. 241–247.

He, Z., Shu, F., Yang, Y., Li, M., Wang, Q., 2012. An investigation on the feasibility of cross-project defect prediction. Autom. Softw. Eng. 19 (2), 167–199. http://dx.doi.org/10.1007/s10515-011-0090-3.

Hosseini, S., Turhan, B., Gunarathna, D., 2019. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. Softw. Eng. 45 (2), 111–147. http://dx.doi.org/10.1109/TSE.2017.2770124.

Hosseini, S., Turhan, B., Mäntylä, M., 2018. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Inf. Softw. Technol. 95, 296–312. http://dx.doi.org/10.1016/j.infsof.2017.06.004, URL http://www.sciencedirect.com/science/article/pii/S0950584916303500.

Jing, X., Wu, F., Dong, X., Qi, F., Xu, B., 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2015, ACM, New York, NY, USA, pp. 496–507. http://dx.doi.org/10.1145/2786805.2786813.

Jing, X., Wu, F., Dong, X., Xu, B., 2017. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans. Softw. Eng. 43 (4), 321–339. http://dx.doi.org/10.1109/TSE.2016.2597849.

Kemerer, C., Chidamber, S., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20, 476–493. http://dx.doi.org/10.1109/32.295895.

Kira, K., Rendell, L.A., 1992. A practical approach to feature selection. In: Proceedings of the Ninth International Workshop on Machine Learning. In: ML92, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 249–256.

Kononenko, I., 1994. Estimating attributes: Analysis and extensions of RELIEF. In: Machine Learning: ECML-94, European Conference on Machine Learning, Catania, Italy, April 6-8, 1994, Proceedings. pp. 171–182. http://dx.doi.org/10.1007/3-540-57868-4_57.

Krishna, R., Menzies, T., 2019. Bellwethers: A baseline method for transfer learning. IEEE Trans. Softw. Eng. 45 (11), 1081–1105. http://dx.doi.org/10.1109/TSE.2018.2821670.

Li, Z., Jing, X.-Y., Wu, F., Zhu, X., Xu, B., Ying, S., 2018. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Autom. Softw. Eng. 25 (2), 201–245. http://dx.doi.org/10.1007/s10515-017-0220-7.

Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., Ying, S., 2019a. Heterogeneous defect prediction with two-stage ensemble learning. Autom. Softw. Eng. 26 (3), 599–651. http://dx.doi.org/10.1007/s10515-019-00259-1.

Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., Ying, S., 2019b. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Trans. Softw. Eng. 45 (4), 391–411. http://dx.doi.org/10.1109/TSE.2017.2780222.

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

Ma, Y., Luo, G., Zeng, X., Chen, A., 2012. Transfer learning for cross-company software defect prediction. Inf. Softw. Technol. 54 (3), 248–256.
MA, Y., ZHU, S., CHEN, Y., LI, J., 2017. Kernel CCA based transfer learning for software defect prediction. IEICE Trans. Inf. Syst. 100 (8), 1903–1906. http://dx.doi.org/10.1587/transinf.2016EDL8238.
Martínez-Cagigal, V., 2021. Multiple Testing Toolbox. https://www.mathworks.com/matlabcentral/fileexchange/70604-multiple-testing-toolbox.
Matthews, B.W., 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim. Biophys. Acta 405 (2), 442.
Menzies, T., Krishna, R., Pryor, D., 2015. The promise repository of empirical software engineering data. http://openscience.us/repo.
Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A., 2010. Defect prediction from static code features: Current results, limitations, new approaches. Autom. Softw. Eng. 17 (4), 375–407. http://dx.doi.org/10.1007/s10515-010-0069-5.
Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L., 2018. Heterogeneous defect prediction. IEEE Trans. Softw. Eng. 44 (9), 874–896. http://dx.doi.org/10.1109/TSE.2017.2720603.
Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. In: 2013 35th International Conference on Software Engineering (ICSE). pp. 382–391. http://dx.doi.org/10.1109/ICSE.2013.6606584.
Ni, C., Xia, X., Lo, D., Chen, X., Gu, Q., 2022. Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans. Softw. Eng. 48 (3), 786–802. http://dx.doi.org/10.1109/TSE.2020.3001739.
Niu, J., Li, Z., Chen, H., Dong, X., Jing, X.-Y., 2022. Data sampling and kernel manifold discriminant alignment for mixed-project heterogeneous defect prediction. Softw. Qual. J. 1–35.
Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q., 2011. Domain adaptation via transfer component analysis. IEEE Trans. Neural Netw. 22 (2), 199–210. http://dx.doi.org/10.1109/TNN.2010.2091281.
Panichella, A., Oliveto, R., De Lucia, A., 2014. Cross-project defect prediction models: L'union fait la force. In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). pp. 164–173. http://dx.doi.org/10.1109/CSMR-WCRE.2014.6747166.
Pascarella, L., Palomba, F., Bacchelli, A., 2020. On the performance of method-level bug prediction: A negative result. J. Syst. Softw. 161, 110493. http://dx.doi.org/10.1016/j.jss.2019.110493.
Peng, L., Yang, B., Chen, Y., Abraham, A., 2009. Data gravitation based classification. Inform. Sci. 179 (6), 809–819. http://dx.doi.org/10.1016/j.ins.2008.11.007.
Peters, F., Menzies, T., Gong, L., Zhang, H., 2013. Balancing privacy and utility in cross-company defect prediction. IEEE Trans. Softw. Eng. 39 (8), 1054–1068. http://dx.doi.org/10.1109/TSE.2013.6.
Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C., 2011. Detecting novel associations in large data sets. Science 334 (6062), 1518.
Ryu, D., Baik, J., 2016. Effective multi-objective naive Bayes learning for cross-project defect prediction. Appl. Soft Comput. 49, 1062–1077. http://dx.doi.org/10.1016/j.asoc.2016.04.009.
Ryu, D., Choi, O., Baik, J., 2016. Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empir. Softw. Eng. 21 (1), 43–71. http://dx.doi.org/10.1007/s10664-014-9346-4.
Ryu, D., Jang, J.-I., Baik, J., 2015. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. J. Comput. Sci. Tech. 30 (5), 969–980.
Shepperd, M., Song, Q., Sun, Z., Mair, C., 2013. Data quality: Some comments on the NASA software defect datasets. IEEE Trans. Softw. Eng. 39 (9), 1208–1215. http://dx.doi.org/10.1109/TSE.2013.11.
Song, Q., Guo, Y., Shepperd, M., 2019. A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans. Softw. Eng. 45 (12), 1253–1269. http://dx.doi.org/10.1109/TSE.2018.2836442.
Tantithamthavorn, C., Hassan, A.E., Matsumoto, K., 2020. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Trans. Softw. Eng. 46 (11), 1200–1219. http://dx.doi.org/10.1109/TSE.2018.2876537.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. 43 (1), 1–18. http://dx.doi.org/10.1109/TSE.2016.2584050.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2019. The impact of automated parameter optimization on defect prediction models. IEEE Trans. Softw. Eng. 45 (7), 683–711. http://dx.doi.org/10.1109/TSE.2018.2794977.
Tong, H., Liu, B., Wang, S., 2021. Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. IEEE Trans. Softw. Eng. 47 (9), 1886–1906. http://dx.doi.org/10.1109/TSE.2019.2939303.
Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J., 2009. On the relative value of cross-company and within-company data for defect prediction. Empir. Softw. Eng. 14 (5), 540–578. http://dx.doi.org/10.1007/s10664-008-9103-7.
Wang, S., Liu, T., Nam, J., Tan, L., 2020. Deep semantic feature learning for software defect prediction. IEEE Trans. Softw. Eng. 46 (12), 1267–1293. http://dx.doi.org/10.1109/TSE.2018.2877612.
Wang, S., Liu, T., Tan, L., 2016. Automatically learning semantic features for defect prediction. In: Proceedings of the 38th International Conference on Software Engineering. ICSE '16, ACM, New York, NY, USA, pp. 297–308. http://dx.doi.org/10.1145/2884781.2884804.
Wang, S., Yao, X., 2013. Using class imbalance learning for software defect prediction. IEEE Trans. Reliab. 62 (2), 434–443. http://dx.doi.org/10.1109/TR.2013.2259203.
Wen, T., Dong, D., Chen, Q., Chen, L., Roberts, C., 2019. Maximal information coefficient-based two-stage feature selection method for railway condition monitoring. IEEE Trans. Intell. Transp. Syst. 20 (7), 2681–2690. http://dx.doi.org/10.1109/TITS.2018.2881284.
Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biom. Bull. 1 (6), 80–83.
Witten, I., Frank, E., Hall, M., Pal, C., 2016. Data Mining: Practical Machine Learning Tools and Techniques, Fourth Edition Morgan Kaufmann, Boston.
Wu, R., Zhang, H., Kim, S., Cheung, S.-C., 2011. Relink: Recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. In: ESEC/FSE '11, ACM, New York, NY, USA, pp. 15–25. http://dx.doi.org/10.1145/2025113.2025120.
Xia, X., Lo, D., Pan, S.J., Nagappan, N., Wang, X., 2016. HYDRA: Massively compositional model for cross-project defect prediction. IEEE Trans. Softw. Eng. 42 (10), 977–998.
Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y., Zhang, T., 2019. Software defect prediction based on kernel PCA and weighted extreme learning machine. Inf. Softw. Technol. 106, 182–200. http://dx.doi.org/10.1016/j.infsof.2018.10.004.
Xu, Z., Xuan, J., Liu, J., Cui, X., 2016. MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: IEEE International Conference on Software Analysis, Evolution, and Reengineering. pp. 370–381.
Yatish, S., Jiarpakdee, J., Thongtanunam, P., Tantithamthavorn, C., 2019. Mining software defects: should we consider affected releases? In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp. 654–665.
Yedida, R., Menzies, T., 2021. On the value of oversampling for deep learning in software defect prediction. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2021.3079841.
Yu, Q., Jiang, S., Zhang, Y., 2017. A feature matching and transfer approach for cross-company defect prediction. J. Syst. Softw. 132, 366–378. http://dx.doi.org/10.1016/j.jss.2017.06.070.
Zhang, F., Mockus, A., Keivanloo, I., Zou, Y., 2016. Towards building a universal defect prediction model with rank transformed predictors. Empir. Softw. Eng. 21 (5), 2107–2145. http://dx.doi.org/10.1007/s10664-015-9396-2.
Zhao, L., Shang, Z., Zhao, L., Zhang, T., Tang, Y.Y., 2019. Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. Neurocomputing 352, 64–74. http://dx.doi.org/10.1016/j.neucom.2019.03.076.
Zheng, W., Shen, T., Chen, X., Deng, P., 2022. Interpretability application of the Just-in-Time software defect prediction model. J. Syst. Softw. 188, 111245. http://dx.doi.org/10.1016/j.jss.2022.111245.
Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., Xu, B., 2018. How far we have progressed in the journey? an examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. (TOSEM) 27 (1), 1–51.
Zhu, K., Ying, S., Zhang, N., Zhu, D., 2021. Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. J. Syst. Softw. 180, 111026. http://dx.doi.org/10.1016/j.jss.2021.111026.
Zou, Q., Lu, L., Qiu, S., Gu, X., Cai, Z., 2021a. Correlation feature and instance weights transfer learning for cross project software defect prediction. IET Softw. 15 (1), 55–74. http://dx.doi.org/10.1049/sfw2.12012.
Zou, Q., Lu, L., Yang, Z., Gu, X., Qiu, S., 2021b. Joint feature representation learning and progressive distribution matching for cross-project defect prediction. Inf. Softw. Technol. 137, 106588. http://dx.doi.org/10.1016/j.infsof.2021.106588.

Haonan Tong received the Ph.D. degree in Systems Engineering from the School of Reliability and Systems Engineering, Beihang University, China, in 2020. He is currently a Lecturer at School of Software Engineering, Beijing Jiaotong University. His research interests include mining software repositories, software reliability prediction, and machine learning.

Wei Lu received the B.S. degree in computer science from Fushun Petroleum Institute, Fushun, China, in 1985 and the M.S. degree in computer

science and the Ph.D. degree in information and communication engineering from Sichuan University, Chengdu, China, in 1998 and 2006, respectively. He is currently a Professor with the School of Software Engineering, Beijing Jiaotong University, Beijing, China. His current research interests include computer networks and information systems, and multimedia information processing.

Weiwei Xing received the B.S. degree in computer science and technology and the Ph.D degree in signal and information processing from Beijing Jiaotong University, Beijing, China, in 2001 and 2006, respectively. She is currently a Professor with the School of Software Engineering, Beijing Jiaotong University. Her research interests include intelligent information processing and machine learning.

Shihai Wang received his Ph.D. in computer science from the University of Manchester, UK in 2010. He joined the School of Reliability and Systems Engineering, Science and Technology on Reliability and Environmental Engineering Laboratory, Beihang University, as a lecturer in 2011. Currently, his research interests include software testing, software fault prediction and pattern recognition and applications in software reliability.