Xu Z, Pang S, Zhang T *et al.* Cross project defect prediction via balanced distribution adaptation based transfer learning. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(5): 1039–1062 Sept. 2019. DOI 10.1007/s11390-019-1959-z

Cross Project Defect Prediction via Balanced Distribution Adaptation Based Transfer Learning

Zhou Xu[1,2,3], Shuai Pang[2], Tao Zhang[1,4,*], *Senior Member*, *CCF*, Xia-Pu Luo[3,*], *Member*, *ACM*, *IEEE*
Jin Liu[2,4,5], *Member*, *CCF*, *IEEE*, Yu-Tian Tang[3], Xiao Yu[2,6], and Lei Xue[3], *Member*, *IEEE*

[1] *College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China*

[2] *School of Computer Science, Wuhan University, Wuhan 430072, China*

[3] *Department of Computing, The Hong Kong Polytechnic University, Hong Kong 999077, China*

[4] *Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences Beijing 100190, China*

[5] *Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China*

[6] *Department of Computer Science, City University of Hong Kong, Hong Kong 999077, China*

E-mail: {zhouxullx, shuaipang}@whu.edu.cn; cstzhang@hrbeu.edu.cn; csxluo@comp.polyu.edu.hk
jinliu@whu.edu.cn; csytang@comp.polyu.edu.hk; xiaoyu_whu@yahoo.com; cslxue@comp.polyu.edu.hk

Received October 22, 2018; revised July 11, 2019.

Abstract Defect prediction assists the rational allocation of testing resources by detecting the potentially defective software modules before releasing products. When a project has no historical labeled defect data, cross project defect prediction (CPDP) is an alternative technique for this scenario. CPDP utilizes labeled defect data of an external project to construct a classification model to predict the module labels of the current project. Transfer learning based CPDP methods are the current mainstream. In general, such methods aim to minimize the distribution differences between the data of the two projects. However, previous methods mainly focus on the marginal distribution difference but ignore the conditional distribution difference, which will lead to unsatisfactory performance. In this work, we use a novel balanced distribution adaptation (BDA) based transfer learning method to narrow this gap. BDA simultaneously considers the two kinds of distribution differences and adaptively assigns different weights to them. To evaluate the effectiveness of BDA for CPDP performance, we conduct experiments on 18 projects from four datasets using six indicators (i.e., F-measure, g-means, Balance, AUC, EARecall, and EAF-measure). Compared with 12 baseline methods, BDA achieves average improvements of 23.8%, 12.5%, 11.5%, 4.7%, 34.2%, and 33.7% in terms of the six indicators respectively over four datasets.

Keywords cross-project defect prediction, transfer learning, balancing distribution, effort-aware indicator

1 Introduction

We are living in an era which can be referred to as software-defined everything[1]. However, defects are inevitable in the source code of software and may cause the failure of the product. Such a failure can lead to poor user experience and even severe economic losses. Thus, identifying the high-risk software modules that may contain defects before releasing the software product is a critical activity for software quality

Regular Paper

This work was partially supported by the National Key Research and Development Program of China under Grant No. 2018YFC1604000, the National Natural Science Foundation of China under Grant Nos. 61602258, 61572374, and U163620068, the China Postdoctoral Science Foundation under Grant No. 2017M621247, the Natural Science Foundation of Heilongjiang Province of China under Grant No. LH2019F008, Heilongjiang Postdoctoral Science Foundation under Grant No. LBH-Z17047, the Open Fund of Key Laboratory of Network Assessment Technology from Chinese Academy of Sciences, Guangxi Key Laboratory of Trusted Software under Grant No. kx201607, the Academic Team Building Plan for Young Scholars from Wuhan University under Grant No. WHU2016012, and Hong Kong GRC (Research Grants Council) Project under Grant Nos. PolyU 152223/17E and PolyU 152239/18E.

*Corresponding Author

assurance[2]. Software defect prediction (SDP) appears to alleviate this issue. SDP detects the most defect-prone modules by analyzing the software history data from the software repositories, such as version control systems (e.g., GitHub and Subversion) and issue tracking systems (e.g., JIRA and Bugzilla).

Most existing studies on SDP focus on building prediction models on the training data, i.e., historical labeled software modules, and then predicting the defect labels of unlabeled modules within the same project, which is referred to as within project defect prediction (WPDP). The training data consist of a set of module features and defect information (i.e., a binary class label or the number of defects)[3]. The module features can be collected from the source code or from the code changes between successive versions. The defect information can be extracted from the commit logs or labeled by the domain experts. In general, training an effective and robust SDP model needs sufficient labeled defect data. However, the process of labeling the software modules is usually labor-intensive and time-consuming. In particular, for ongoing or immature projects, very little or no historical development data are available to extract the label information. In such a case, WPDP is infeasible.

Fortunately, there are publicly available labeled software defect data online whose quality has been recognized by previous researchers. Existing studies proposed to utilize the labeled data of an external project (aka source project) to conduct SDP on the project (aka target project) with limited or no labeled training data[4], which called cross-project defect prediction (CPDP). However, distinct projects have different scales and functional complexity, which may lead to the distribution differences between the data across projects. Thus, the difficulty that needs to be solved for CPDP is to eliminate the differences. Transfer learning based and training data filter based CPDP methods are adopted to address this issue. In this work, we focus on the former ones which are commonly studied.

Transfer learning based CPDP methods transfer the knowledge of the source project to annotate the target project with the aim to minimize the distribution differences of the data between the two projects[5,6]. There are two kinds of distribution differences, i.e., the marginal and the conditional distribution differences. The former one is the distribution of the module features themselves, and the later one is the distribution of the module labels given the values of the module features. Previous transfer learning based CPDP methods, like the method in [6], mainly focus on adapting the marginal distribution difference. However, when the data of two projects are much more dissimilar, the importance of the marginal distribution is higher than that of the conditional distribution, whereas when the data of two projects are similar, the conditional distribution is more important than the marginal distribution[7]. Thus only considering one distribution is not appropriate for all cross-project pairs and will limit the CPDP performance on some cases. To overcome this deficiency, the intuition here is to simultaneously adapt the two kinds of distributions. In this work, we introduce a novel balanced distribution adaptation (BDA)[7] based transfer learning as our CPDP method to tackle the distribution adaptation problem. More specifically, BDA not only considers both marginal and conditional distribution differences between the data of two projects, but also assigns different importance degrees to the two kinds of distributions, and thus it can adapt to various cross-project pairs more effectively.

To simulate the CPDP scenario by using the BDA method, we choose 18 projects from four benchmark defect datasets as our studied corpora. To assess the CPDP performance, we choose four traditional indicators, i.e., F-measure, g-mean, Balance[8] and AUC[9], and two effort-aware indicators, i.e., EARecall and EAF-measure[10,11] as our evaluation measurement. The experimental results show that BDA achieves the best average indicator values in most cases compared with six training data filter methods and six transfer learning methods.

The main contributions of this paper are highlighted as follows.

• We introduce a novel transfer learning method BDA for CPDP. BDA combines the marginal and conditional distribution to reduce the data distribution differences between two projects. In addition, BDA also considers the different importance degrees of the two kinds of distribution differences with a balance factor.

• We perform sufficient experiments on total 66 cross-project pairs from 18 project data of four benchmark datasets to evaluate the effectiveness of BDA. The experimental results show the superior of BDA compared with 12 baseline CPDP methods.

Paper Organization. The remainder of the paper is organized as follows. We introduce the related work on exiting CPDP methods in Section 2. We describe the technical details of the used BDA method in Section 3. We present our experimental setup, such as

the research questions, benchmark datasets, and performance indicators in Section 4. We analyze our experimental results in Section 5. We discuss the impacts of different regularization parameters, feature dimensions, and classifiers on the BDA performance in Section 6. We list four kinds of threats to validity in Section 7. Finally, we give the conclusions of this paper and the future work in Section 8.

2 Related Work

To the best of our knowledge, Briand *et al.*[12] were the first to explore whether the CPDP model built on one system for another system was worth investigating. However, the experimental results on two java systems implied that such a model achieved poor performance. Another early study about CPDP is performed by Zimmermann *et al.*[13] The experimental results on total 622 cross-project pairs with logistic regression classifier showed that only 3.4% pairs achieved satisfactory performances. The reason of the disappointing results from these early studies is that they conducted CPDP by using all modules of the source project to train the classification model without considering the data distribution differences of the two projects. To address this issue, recently, researchers have proposed different methods to narrow the gap of the distribution differences between the cross project data. Existing related studies can be roughly divided into two groups: the training data filter based CPDP methods and the transfer learning based CPDP methods.

2.1 Training Data Filter Based CPDP Methods

The training data filter based CPDP methods select part of the modules from the source project based on a specific rule, such as the similarity towards the modules of the target project. These selected modules are relevant to the modules of the target project, which helps reduce the data distribution differences between the two projects. Thus the classification model trained on the selected modules is more targeted to the target project.

To the best of our knowledge, Turhan *et al.*[14] were among the first to introduce the concept of training data filter into the CPDP task. They proposed a nearest neighbor filter method, called NN-Filter, to select some modules from the source project that are close to the modules of the target project. More specifically, for each module of the target project, NN-Filter selects its top 10 nearest modules in the source project, and then removes the duplication ones from those selected modules. The remaining modules are used as the training set to train the classification model. They used the common features of 12 NASA projects to build the cross-project model, and found that the model improved the probability of detecting defects but also dramatically increased the false positive rate.

Peters *et al.*[15] proposed a module subset selection strategy, called Peter-Filter, with a clustering method. More specifically, Peter-Filter first combines the modules of the source and the target projects into a whole, then uses the k-means clustering method to divide the modules into several groups, and only reserves the groups that contain at least one module of the target project. For each module of the source project in the remaining groups, Peter-Filter selects its nearest module from the target project. Those selected modules are deemed as the popular modules. Then for each selected popular module, Peter-Filter selects its closest module (called the greatest fan) from the source project as one member of the final training set. The experiments on 56 defect data showed that Peter-Filter is more effective to improve the CPDP performance than NN-Filter and better than the WPDP setting on small projects.

Kawata *et al.*[16] proposed a relevancy filter method called DBSCAN (we call Kawata-Filter in this work) for training set simplification. More specifically, Kawata-Filter first mixes the modules of two projects, and then employs the DBSCAN clustering method to cluster the mixed modules into several groups. Kawata-Filter discards the groups that do not contain any module of the target project. Then the modules of the source project in the remaining groups are fed into the classification model. The experiments on 56 defect data showed that Kawata-Filter achieved better performance than NN-Filter and Peter-Filter in terms of AUC and g-measure.

Following Kawata *et al.*'s work[16], Yu *et al.*[17] proposed a new data filter method called DFAC (we call Yu-Filter in this work). Compared with Kawata-Filter, Yu-Filter just replaces the DBSCAN clustering method with the agglomerative clustering method. All other steps are the same. The experimental results on 15 defect data showed that Yu-Filter made a small performance improvement compared with Kawata-Filter.

Different from the above studies, we address the issue of data distribution differences for the CPDP task by using a transfer learning method, not from the perspective of filtering the training data.

In addition, some previous studies[18−20] proposed another type of data filter methods which aims to se-

lect some source projects that are similar to the target project under the assumption that a set of source projects are available. Different from these studies, we focus on one-to-one CPDP, i.e., only using one project as the source project without involving data selection at the project level.

2.2 Transfer Learning Based CPDP Methods

Transfer learning based CPDP methods utilize various transfer learning methods to transform the data of the source and the target projects into a new feature space. In the new space, the distributions of the two transformed data are more similar. Thus, the classification model trained on the new source project data will be more effective than that trained on the original data.

To the best of our knowledge, Ma *et al.*[5] were among the first to introduce transfer learning into CPDP. Instead of discarding some modules of the source project, they proposed a transfer naive Bayes (TNB) method to transfer the valuable information of the source project into that of the target project. TNB first utilizes the data gravitation formula to measure the similarity of the modules of the source project to the modules of the target project, assigns different weights to the modules of the source project based on the similarity, and then integrates the weight information into the Bayes formula to develop a weighted Naive Bayes method for transfer learning. The experiments on seven defect data from dataset NASA and three defect data from dataset SOFTLAB showed that TNB achieved better performances than the NN-Filter method.

Nam *et al.*[6] proposed an extended transfer component analysis (TCA) method, TCA+, to learn some transfer components for cross project data in a kernel Hilbert space. TCA+ first defines some rules to find the optimum data normalization strategy, and then applies the original TCA method to make the data distributions of the two projects closer. The experiments on five defect data from the AEEEM dataset and three defect data from the RELINK dataset showed that TCA+ achieved competitive performance compared with the WPDP setting and the original TCA method.

Chen *et al.*[21] proposed a transfer learning method, called double transfer boosting (DTB), for CPDP. DTB first re-weights the modules of the source project based on data gravitation formula and then applies a transfer boosting method to eliminate some negative modules from the source project. The experiments showed that DTB outperformed four baseline CPDP methods and achieved better performances than three WPDP methods. The main drawback of DTB is that the used transfer boosting method needs the participation of some labeled modules from the target project, which limits its usage in the scenario that the target project has no labeled modules.

Ryu *et al.*[22] proposed a transfer cost-sensitive boosting (TCSBoost) method that considers both knowledge transfer and class imbalance for CPDP. TCSBoost first calculates the similarity weight between the source and the target projects, employs a resampling method to rebalance the data distribution of the defective and the non-defective classes of the source project, and then applies a cost-sensitive boost method to deal with the distribution differences between the two projects. Like the DTB method, TCSBoost requires a small amount of labeled modules of the target project, which hinders its usage in the general CPDP scenario.

Liu *et al.*[23] proposed a two-phase transfer learning (TPTL) model. In the first stage, TPTL selects two source projects, having the highest distribution similarity to the target project and the best performance respectively, as candidates from a set of source projects. In the second stage, TPTL utilizes the TCA+ method to build two transfer learning models based on the two candidates to conduct CPDP. The focus of this work is on the selection of the candidate source projects.

These transfer learning based CPDP methods do not take into full consideration of both the marginal and the conditional distribution differences between the cross project data. More specifically, the marginal distribution is the probability associated with a variate without regarding to the value of the other variate. For two dependent variates, the conditional distribution focuses on computing the probability associated with one variate given information about the value of the other variate[24]. In this work, we make a step forward to consider both distribution differences simultaneously and their different importance degrees, aiming to further improve the CPDP performance.

3 Method

3.1 Notations

We first define some notations used in BDA.

Assume the source project as $\boldsymbol{D}_{\mathrm{S}} = \{\boldsymbol{X}_{\mathrm{S}}, \boldsymbol{Y}_{\mathrm{S}}\} = \{\boldsymbol{x}_{\mathrm{s}}^i, \boldsymbol{y}_{\mathrm{s}}^i\}_{i=1}^{n_{\mathrm{s}}}$, where $\boldsymbol{x}_{\mathrm{s}}^i$ denotes the i-th module, $\boldsymbol{X}_{\mathrm{S}} \in \mathbb{R}^{n_{\mathrm{s}} \times d_{\mathrm{s}}}$ denotes the feature set of the source project,

and d_s and n_s denote the feature dimension and the number of modules respectively. In other words, the row of matrix $\boldsymbol{X}_S$ denotes the software modules and the column of matrix $\boldsymbol{X}_S$ denotes the module feature. In addition, $\boldsymbol{y}_s^i$ denotes the label of the i-th module and $\boldsymbol{Y}_S \in \mathbb{R}^{n_s \times 1}$ denotes the label set of the source project. Similarly, assume the target project as $\boldsymbol{D}_T = \{\boldsymbol{X}_T, \boldsymbol{Y}_T\} = \{\boldsymbol{x}_t^j, \boldsymbol{y}_t^j\}_{j=1}^{n_t}$, where $\boldsymbol{x}_t^j$ denotes the j-th module, $\boldsymbol{X}_T \in \mathbb{R}^{n_t \times d_t}$ denotes the feature set of the target project, and d_t and n_t denote the feature dimension and the number of modules respectively. In our work, $d_s = d_t$, that is, the cross project data share the same feature dimension. $\boldsymbol{y}_t^j$ denotes the label of the j-th module and $\boldsymbol{Y}_T \in \mathbb{R}^{n_t \times 1}$ denotes the label set of the target project. Note that the labels of the target project are unknown and need to be predicted. In addition, we define the feature space of source and target projects as $\mathcal{X}_s$ and $\mathcal{X}_t$ respectively, and the label space of source and target projects as $\mathcal{Y}_s$ and $\mathcal{Y}_t$ respectively. In the CPDP scenario, the defect data of the two projects have the same feature space, i.e., $\mathcal{X}_s = \mathcal{X}_t$ and the same label space, i.e., $\mathcal{Y}_s = \mathcal{Y}_t$, but have different marginal distributions, i.e., $P(\boldsymbol{x}_s) \neq P(\boldsymbol{x}_t)$ and different conditional distributions, i.e., $P(\boldsymbol{y}_s|\boldsymbol{x}_s) \neq P(\boldsymbol{y}_t|\boldsymbol{x}_t)$. For the CPDP task, BDA adaptively minimizes the marginal distribution difference, i.e., $d(P(\boldsymbol{x}_s), P(\boldsymbol{x}_t))$, and the conditional distribution difference, i.e., $d(P(\boldsymbol{y}_s|\boldsymbol{x}_s), P(\boldsymbol{y}_t|\boldsymbol{x}_t))$ simultaneously, aiming at learning the labels $\boldsymbol{y}_t$ of the data of the target project $\boldsymbol{D}_T$ by utilizing the labeled data of the source project $\boldsymbol{D}_S$.

3.2 Balanced Distribution Adaptation (BDA)

The ideal transfer learning for CPDP should consider both the marginal and the conditional distribution differences between the source and the target projects. That is, it needs to minimize the distance between $\boldsymbol{D}_S$ and $\boldsymbol{D}_T$ as follows:

$$\begin{aligned} & d(\boldsymbol{D}_S, \boldsymbol{D}_T) \\ = \ & d(P(\boldsymbol{x}_s), P(\boldsymbol{x}_t)) + d(P(\boldsymbol{y}_s|\boldsymbol{x}_s), P(\boldsymbol{y}_t|\boldsymbol{x}_t)). \end{aligned} \quad (1)$$

However, the main drawback of (1) is that it treats the importance of the two kinds of distributions equally. However, when the dissimilarity of the two projects is large, the margin distribution should be paid more attention, whereas when the similarity of the two projects is large, the conditional distribution should be more concerned. Therefore, for different cross project data, it is not reasonable to simply combine the two kinds of distributions with the same importance (i.e., weight). To overcome this deficiency, BDA is proposed to solve this issue by adaptively assigning different weights to the two kinds of distributions based on various cross-project pairs. BDA is formulated as follows:

$$\begin{aligned} d(\boldsymbol{D}_S, \boldsymbol{D}_T) \approx \ & (1-\mu) d(P(\boldsymbol{x}_s), P(\boldsymbol{x}_t)) + \\ & \mu d(P(\boldsymbol{y}_s|\boldsymbol{x}_s), P(\boldsymbol{y}_t|\boldsymbol{x}_t)), \end{aligned} \quad (2)$$

where $\mu \in [0, 1]$ is a balance factor that is used to highlight different importance degrees of the two kinds of distributions. When the dissimilarity of the cross project data is larger, μ tends to be 0, which means that the importance of the marginal distribution is emphasized; whereas when the cross project data are more similar, μ tends to be 1, which means that the conditional distribution is more important. Since the balance factor μ can adaptively adjust the importance of the two kinds of distributions for the specific cross-project pair, BDA has the potential to generate a targeted training set for the CPDP task.

However, the labels of the target project are not available in advance because they are the outputs of the CPDP task. In other words, $\boldsymbol{y}_t$ is unknown, which leads to that it is not feasible to calculate the term $P(\boldsymbol{y}_t|\boldsymbol{x}_t)$. An alternative way is to use the class conditional distribution $P(\boldsymbol{x}_t|\boldsymbol{y}_t)$ to approximate the conditional distribution of the cross project data. The fact here is that when the amount of modules is adequate, the values of $P(\boldsymbol{x}_t|\boldsymbol{y}_t)$ and $P(\boldsymbol{y}_t|\boldsymbol{x}_t)$ are approximately equal according to the sufficient statistics[25]. To calculate the class conditional distribution, a basic classifier is built on the source project data $\boldsymbol{D}_S$ and the trained model is used to predict the labels of the target project data $\boldsymbol{D}_T$. Since the predicted labels may be not accurate at first, multiple iterations are employed to refine the labels until the results are stable.

To calculate the discrepancy between two marginal distributions, i.e., $d(P(\boldsymbol{x}_s), P(\boldsymbol{x}_t))$, and the two conditional distributions, i.e., $d(P(\boldsymbol{y}_s|\boldsymbol{x}_s), P(\boldsymbol{y}_t|\boldsymbol{x}_t))$ in (2), the maximum mean discrepancy (MMD) method[26] is applied to empirically estimate them. Then (2) is rewritten as follows:

$$\begin{aligned} & d(\boldsymbol{D}_S, \boldsymbol{D}_T) \\ = \ & (1-\mu) \| \frac{1}{n_s} \sum_{i=1}^{n_s} \boldsymbol{x}_s^i - \frac{1}{n_t} \sum_{j=1}^{n_t} \boldsymbol{x}_t^j \|_{\mathcal{H}}^2 + \\ & \mu \sum_{c=1}^{C} \| \frac{1}{n_s^c} \sum_{\boldsymbol{x}_s^i \in \boldsymbol{D}_S^{(c)}} \boldsymbol{x}_s^i - \frac{1}{n_t^c} \sum_{\boldsymbol{x}_t^j \in \boldsymbol{D}_T^{(c)}} \boldsymbol{x}_t^j \|_{\mathcal{H}}^2, \end{aligned} \quad (3)$$

where $\mathcal{H}$ means the reproducing kernel Hilbert space, C denotes the number of different labels ($C = 2$ in the CPDP scenario), $\boldsymbol{D}_{\mathrm{S}}^{(c)}$ and $\boldsymbol{D}_{\mathrm{T}}^{(c)}$ denote the modules with label c in the source and the target projects respectively, and $n_{\mathrm{s}}^c = |\boldsymbol{D}_{\mathrm{S}}^{(c)}|$ and $n_{\mathrm{t}}^c = |\boldsymbol{D}_{\mathrm{T}}^{(c)}|$ denote the number of modules belonging to $\boldsymbol{D}_{\mathrm{S}}^{(c)}$ and $\boldsymbol{D}_{\mathrm{T}}^{(c)}$ respectively. The first term and the second term in (3) represent the marginal distribution discrepancy and the conditional distribution discrepancy among the cross project data, respectively.

Using the matrix tricks and regularization, (3) is equal to the following formula:

$$\begin{aligned}&\min \operatorname{tr}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}((1-\mu)\boldsymbol{M}_0+\mu\sum_{c=1}^{\mathrm{C}}\boldsymbol{M}_c)\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})+\lambda\|A\|_F^2\\&\text{s.t. } \boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}=\boldsymbol{I},\quad 0\leqslant\mu\leqslant 1,\end{aligned}\tag{4}$$

where $\boldsymbol{X}$ denotes the input data matrix that combines the feature sets of the source project $\boldsymbol{X}_{\mathrm{S}}$ and the target project $\boldsymbol{X}_{\mathrm{T}}$, $\boldsymbol{A}$ denotes a transformation matrix, $\boldsymbol{I}\in\mathbb{R}^{(n_{\mathrm{s}}+n_{\mathrm{t}})\times(n_{\mathrm{s}}+n_{\mathrm{t}})}$ denotes the identity matrix, and $\boldsymbol{H}=\boldsymbol{I}-(1/n)\boldsymbol{1}$ denotes a centering matrix. In addition, $\boldsymbol{M}_0$ and $\boldsymbol{M}_c$ are MMD matrices that can be calculated using (5) and (6) as follows:

$$(\boldsymbol{M}_0)_{ij}=\begin{cases}\dfrac{1}{n_{\mathrm{s}}^2}, & \text{if } \boldsymbol{x}_i,\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{S}},\\ \dfrac{1}{n_{\mathrm{t}}^2}, & \text{if } \boldsymbol{x}_i,\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{T}},\\ -\dfrac{1}{n_{\mathrm{s}}n_{\mathrm{t}}}, & \text{otherwise,}\end{cases}\tag{5}$$

$$(\boldsymbol{M}_c)_{ij}=\begin{cases}\dfrac{1}{n_{\mathrm{s}}^{c2}}, & \text{if } \boldsymbol{x}_i,\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{S}}^{(c)},\\ \dfrac{1}{n_{\mathrm{t}}^{c2}}, & \text{if } \boldsymbol{x}_i,\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{T}}^{(c)},\\ -\dfrac{1}{n_{\mathrm{s}}^c n_{\mathrm{t}}^c}, & \text{if } \begin{cases}\boldsymbol{x}_i\in\boldsymbol{D}_{\mathrm{S}}^{(c)},\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{T}}^{(c)},\\ \boldsymbol{x}_i\in\boldsymbol{D}_{\mathrm{T}}^{(c)},\boldsymbol{x}_j\in\boldsymbol{D}_{\mathrm{S}}^{(c)},\end{cases}\\ 0, & \text{otherwise.}\end{cases}\tag{6}$$

The first term in (4) with balance factor μ is used to adapt the importance degrees of the marginal and conditional distributions, and the second term with parameter λ is a regularization term where $\|\boldsymbol{A}\|_F^2$ is the Frobenius norm. The first constraint condition is used to ensure that the transformed data $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}$ preserve the inner structure properties of the original data, and the second one constrains the value range of the balance factor μ.

To solve (4), we define the Lagrange multipliers as $\Phi=(\phi_1,\phi_2,...,\phi_d)$, and then (4) can be rewritten as follows:

$$\begin{aligned}L=\ &\operatorname{tr}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}((1-\mu)\boldsymbol{M}_0+\mu\sum_{c=1}^{\mathrm{C}}\boldsymbol{M}_c)\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})+\\&\lambda\|\boldsymbol{A}\|_F^2+\operatorname{tr}((\boldsymbol{I}-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A})\Phi).\end{aligned}\tag{7}$$

We set the first-order derivative of (7) in terms of $\boldsymbol{A}$ to 0, i.e., $\partial L/\partial\boldsymbol{A}=0$, and the optimization is transformed into a generalized eigendecomposition problem as follows:

$$\begin{aligned}&(\boldsymbol{X}((1-\mu)\boldsymbol{M}_0+\mu\sum_{c=1}^{\mathrm{C}}\boldsymbol{M}_c)\boldsymbol{X}^{\mathrm{T}}+\lambda\boldsymbol{I})\boldsymbol{A}\\&=\boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\Phi.\end{aligned}\tag{8}$$

As a result, the optimum transformation matrix $\boldsymbol{A}$ is obtained as the solution of (8). Given a threshold of feature dimension that we want to preserve for the new feature set, we can get the transformed data of the source and the target projects.

4 Experimental Setup

4.1 Research Questions

To evaluate the effectiveness of the BDA method for the CPDP performance, we design the following two research questions (RQ).

*RQ*1. Is BDA more effective than the training data filter based CPDP methods?

Motivation. Training data filter based CPDP methods alleviate the data distribution differences of two projects by selecting some modules from the source project that are representative to the modules of the target project. Such methods do not change the feature spaces of the two projects. However, some of the discarded modules may contain important information to distinguish the modules of different classes. Different from this kind of methods, BDA just transfers the feature spaces while reserving all modules of the source project avoiding the information loss. This question is designed to investigate whether BDA is superior to the training data filter based methods for CPDP performance improvement.

*RQ*2. Does BDA perform better than the transfer learning based CPDP methods?

Motivation. The distribution differences of the cross project data come from two aspects: the marginal and the conditional distribution differences. In addition, according to the distinct similarity levels between the data of the two projects, the importance degrees of the two

kinds of distributions vary. However, existing transfer learning based CPDP methods neither simultaneously consider the two kinds of distributions, nor focus on their different importance degrees. This question is designed to investigate whether the method considering both distributions and their importance degrees (i.e., BDA) will further improve the CPDP performance compared with other transfer learning methods.

4.2 Benchmark Datasets

In this work, we conduct large-scale experiments on four defect benchmark datasets, including the AEEEM, NASA, SOFTLAB, and RELINK datasets.

• *AEEEM Dataset*. This dataset was denoted by D'Ambros *et al.*[27] The name comes from the first letter of its five projects, i.e., Apache Lucene (LC), Equinox (EQ), Eclipse JDT Core (JDT), Eclipse PDE UI (PDE), and Mylyn (ML). Each project data have 61 features, including 17 source code features, five previous-defect features, five entropy-of-change features, 17 entropy-of-source-code features, and 17 churn-of-source code features[27]. The linearly decayed entropy based and weighted churn based features are verified to be closely related to defect information.

• *NASA Dataset*. This dataset is the most popular defect data in previous defect prediction studies[8,28−33]. The project data are extracted from a software system or sub-system and consist of a set of static code features, including McCabe complexity, Halstead complexity, and some miscellaneous features. These features are informative predictors to the software quality. In this work, we use five out of 14 project data (i.e., CM1, MW1, PC1, PC3, and PC4) as our studied corpora since they share the same 38 features.

• *SOFTLAB Dataset*. The five project data, i.e., ar1, ar3, ar4, ar5, and ar6 in this dataset come from a Turkish software company which develops embedded controllers for home appliances[34]. Each project data consist of 29 static code features.

• *RELINK Dataset*. This dataset, denoted by Wu *et al.*[35], contains three projects, i.e., Apache HTTP Server (Apache), OpenIntents Safe (Safe), and ZXing. Each data include 26 static code features. The links between the defect information and the change logs have been manually verified.

The basic properties for each project data in the four benchmark datasets are presented in Table 1. The first four columns report the dataset name, the project name, a brief description of the project, and the development language (lang.), respectively. # F, # M, # DM, and % DM denote the number of features, the number of modules, the number of defective modules, and the percentage of defective modules, respectively. The last column lists the prediction granularity (gran.) of the modules. Each module represents a class, a function, or a file at different granularity levels.

Table 1. Properties of Projects in 4 Benchmark Datasets

Dataset	Project	Description	Lang.	# F	# M	# DM	% DM (%)	Gran.
AEEEM	EQ	OSGi framework	Java	61	324	129	39.8	Class
	JDT	IDE development			997	206	20.7	
	LC	Search Engine library			691	64	9.3	
	ML	Task management			1 862	245	13.2	
	PDE	IDE development			1 497	209	14.0	
NASA	CM1	Spacecraft instrument	C	38	327	42	12.8	Function
	MW1	A zero gravity experiment about combustion			251	25	10.0	
	PC1	Flight software for earth orbiting satellite			696	55	7.9	
	PC3	Flight software for earth orbiting satellite			1 073	132	12.3	
	PC4	Flight software for earth orbiting satellite			1 276	176	13.8	
SOFTLAB	ar1	Embedded controller for white-goods	C	29	121	9	7.4	Function
	ar3	Washing machine			63	8	12.7	
	ar4	Dishwasher			107	20	18.7	
	ar5	Refrigerator			36	8	22.2	
	ar6	Embedded controller for white goods			102	15	14.7	
RELINK	Apache	Web server	Java	26	194	98	50.5	File
	Safe	Security			56	22	39.3	
	Zxing	Bar-code scanning library			399	118	29.6	

4.3 Classification Model

In this work, we select the logistic regression (LR) classifier[36] to train the CPDP classification model. This classifier is an extension of the linear regression model with logistic function and frequently used in previous defect prediction studies[6,13,29,37−44]. We use the third-party implementation, i.e., the LIBLINEAR package[45], following the previous studies[6,42−44]. We will discuss the impacts of different classifiers on the CPDP performance of BDA in Subsection 6.3.

4.4 Performance Indicators

It is a typical binary classification for predicting whether a module in the target project is defective or not. To evaluate the effectiveness of the BDA method for CPDP, we employ six commonly-used indicators in previous defect prediction studies as our performance measures, including four traditional indicators (i.e., F-measure, g-mean, Balance, and AUC) and two effort-aware indicators (i.e., EARecall and EAF-measure). Before giving the definitions of these indicators, we first introduce some basic terms.

For a binary classification task, there are four possible outputs: true positive (TP) denotes the number of actually defective modules that are correctly predicted; true negative (TN) denotes the number of actually non-defective modules that are correctly predicted; false positive (FP) denotes the number of actually non-defective modules that are incorrectly predicted; false negative (FN) denotes the number of actually defective modules that are incorrectly predicted. Given the four terms, we can obtain another three basic terms: the possibility of detection (pd or recall) is defined as $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$, the possibility of false alarm (pf) is defined as $\frac{\mathrm{FP}}{\mathrm{FP+TN}}$, and the precision is defined as $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$.

F-measure is the harmonic average of the precision and recall. Its general formula is defined as follows

$$F\text{-}measure = \frac{(1+\theta^2) \times precision \times recall}{\theta^2 \times precision + recall}, \quad (9)$$

where θ is a positive real parameter to emphasize the importance of the recall and the precision. There are three commonly-used types of F-measure, including F_1 ($\theta = 1$), $F_{0.5}$ ($\theta = 0.5$), and F_2 ($\theta = 2$). Thereinto, F_2 gives higher weight to recall than to precision, which means that it places more emphasis on FN[46]. In SDP application, defective modules are the main focuses because they can cause the software failure. Therefore, an effective SDP method should correctly detect more defective modules, which is related to the definition of recall. Thus, in this work, we follow previous studies[47−49] to choose F-measure with $\theta = 2$ as one of our performance indicators.

g-mean is the geometric mean of recall and 1-pf as

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

Balance is proposed by [8] which calculates the Euclidean distance between the actual (pd, pf) point and the optimal $(\mathrm{pd}', \mathrm{pf}')$, i.e, (1, 0). This indicator is frequently used in previous defect prediction studies[8,50−52]. Balance is defined as follows:

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

AUC calculates the area under the ROC curve to measure the performance of a classification model[53]. The ROC curve is a two-dimensional plane with pd as the y-axis and pf as the x-axis. AUC is independent of the classification threshold.

Effort-aware F-measure (EAF-measure) is calculated under the scenario that only limited test efforts are available for quality assurance activities. Ideally, the testers expect to obtain greater profits within fewer test efforts. The lines of code (LOC) that are inspected during the testing process are treated as the test efforts and the percentage of discovered actually defective modules is treated as the profits. In general, the available test efforts are set to 20% of total LOC. We follow the previous studies[10,11,54] to calculate EAF-measure. The calculation process is described as follows. 1) We train a classification model on the transformed source project data and predict the transformed target project data into two groups, i.e., the predicted defective group and the predicted non-defective group. 2) The modules in the two groups are sorted in ascending order based on their LOC value individually. 3) We merge the two sorting results in which the result of the predicted defective group is listed on the top. 4) We imitate the testers to inspect the modules one by one and record the cumulative percentage of the inspected LOC. 5) We stop the inspection process until the cumulative percentage first reaches 20% of total LOC. 6) We count the following three terms: t_{nd} (denoting the number of defective modules in the target project), t'_n (denoting the number of inspected modules within the inspection of 20% of LOC), and t'_{nd} (denoting the number of discovered actually defective modules within the inspection of 20% of LOC).

After obtaining the three terms, we can calculate two additional indicators effort-aware precision (EAPrecision) and effort-aware recall (EARecall) as follows.

$$EAPrecision = t'_{nd}/t'_n,$$
$$EARecall = t'_{nd}/t_{nd}.$$

Like the definition of F-measure, EAF-measure is defined as (EAPrecision and EARecall are abbreviated to EAP and EAR, respectively):

$$EAF\text{-}measure = \frac{(1+\theta_1^2) \times EAP \times EAR}{\theta_1^2 \times EAP + EAR}. \quad (10)$$

Like F-measure, we also set θ_1 to 2 for EAF-measure. In addition, if the denominators in (9) and (10) are equal to 0, F-measure and EAF-measure make no sense. At this point, we set F-measure and EAF-measure to 0, which denotes the worst performance.

4.5 Cross-Project Prediction Setting

In this work, we organize the cross-project setting on the defect data within the same benchmark dataset and conduct one-to-one CPDP. For example, if the defect data of project EQ in the AEEEM dataset are selected as the target project data, total four cross-project pairs are formed by treating other four projects in the AEEEM dataset as the source project one by one. Thus, we can obtain 20, 20, 20, and 6 cross-project pairs for the AEEEM dataset, the NASA dataset, the SOFTLAB dataset, and the RELINK dataset, respectively.

4.6 Parameter Configuration

In the BDA method, there are four parameters that need to be specified. 1) For the regularization parameter λ in (8), we set it to 0.1 and will discuss the impacts of different λ values on the CPDP performance of BDA in Subsection 6.2. 2) As mentioned in Subsection 3.2, we use multiple iterations to refine the predicted labels. In our experiment, we set the maximal iterations to 10. 3) For the balance factor μ in (8), it is a project-specific parameter, which means that the μ value varies for different cross-project pairs. In other words, the μ value is estimated based on the data distributions of the two projects. Unfortunately, there is no effective way for such estimation[7]. In a real application scenario, it is feasible to use the cross-validation strategy to determine the optimum μ value. Since this work just makes an initial exploration to investigate whether considering both two kinds of distributions with different weights can further improve the CPDP performance, we set 11 different μ values, i.e., 0, 0.1,..., 0.9, 1, and run BDA on each μ value to search the optimum value. 4) For the desired feature dimensions of the transformed source and target projects, in this work, we just choose to reserve 5% of total feature number and will discuss the impacts of different feature dimensions on the CPDP performance of BDA in Subsection 6.1. The code of BDA and the used benchmark datasets are available in our online supplementary materials①.

4.7 Statistical Test

In this work, the Frideman test with a post-hoc test (called Nemenyi test)[55] is employed to analyze the performance differences between DBA and the baseline methods with the significant level α at 0.05. The advantage of this test is that it does not require the performance values to follow a particular distribution. The performance values among the methods have statistically significant differences when the Friedman test gets a small p value (less than 0.05). If the significant differences exist, then the Nemenyi test is used to find which methods have or do not have significant differences with each other. More specifically, for a method pair, if their rank difference exceeds a critical difference (CD), then the two methods are deemed to have significant differences; on the contrary, they have no significant differences. The CD value is calculated as follows:

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

where L denotes the number of methods compared, M denotes the number of cross-project pairs, and $q_{(\alpha,L)}$ is a critical value based on L and the significance level α which is available online②. The Frideman test with the Nemenyi test is widely adopted in previous studies to statistically analyze the differences among various SDP methods[27,29,39,43,44,47,56,57].

Original Nemenyi test has a limitation that it may generate the overlapping groups for the methods[30,57]. In other words, Nemenyi test may assign a method into multiple groups in which the methods in the same group have no significant differences while the methods in distinct groups have significant differences. In this work, we follow the strategy in [57] to remedy this drawback. More specifically, we define the best rank and the worst

①https://sites.google.com/view/bda-cpdp/, July 2019.

②http://www.cin.ufpe.br/~fatc/AM/Nemenyi_critval.pdf, July 2019.

rank of these methods as r_b and r_w respectively. If the absolute delta value $|r_b - r_w|$ is larger than twice the CD value, the methods are assigned to three non-overlapping groups: 1) for the method with rank r_i, if the absolute delta value $|r_i - r_b|$ is less than CD, then it is assigned to the top rank group; 2) for the method with rank r_j, if the absolute delta value $|r_j - r_w|$ is less than CD, then it is assigned to the bottom rank group; 3) the other methods are assigned to the middle rank group. And if the absolute delta value $|r_b - r_w|$ lies between the CD value and twice the CD value, the methods are assigned to two non-overlapping groups: the method with rank r_k is assigned to the top rank group (or the bottom rank group) if r_k is closer to r_b (or r_w). In addition, if the absolute delta value $|r_b - r_w|$ is less than the CD value, all methods are divided into one group without significant differences.

4.8 Experimental Environment

We conduct the experiments on our computer which is equipped with a 16-core Intel® Xeon® E3-1270@3.8 GHz CPU, 32.0 GB RAM. The BDA method is run on MATLAB 2014a.

5 Experimental Results

5.1 Results Analysis for RQ1

5.1.1 Analysis Method for RQ1

To answer this question, we employ some training data filter based CPDP models as our baseline methods including ALL, NN-Filter, Peter-Filter, Yu-Filter, and HISNN. ALL means that we use all the modules of the source project to train the classification model without any data filter process. We treat this method as a special data filter method and the most basic setting for comparison. NN-Filter, Peter-Filter, and Yu-Filter are described in Subsection 2.1. HISNN[51] is a nearest-neighbor based hybrid training data selection method. This method uses a k-nearest neighbor to learn the local knowledge and employs Naive Bayes to learn the global knowledge. Note that HISNN uses a hybrid rule to determine the module labels of the target project, we could not calculate the AUC indicator without the output probability. We also implement the Kawata-Filter method with the parameter setting in the original paper and try some other settings. However, this method identifies many modules as the noise and discards them on majority cross-project pairs, which makes it impossible for us to get modules from the source project to form the training set in most cases. Thus we do not choose this method for comparison. In addition, Zhou *et al.*[58] proposed a simple unsupervised model which uses two simple ranking strategies to calculate the traditional binary classifier indicators and the effort-aware indicators. More specifically, the method calculates the traditional indicators by employing a ManualDown technique which considers a larger module to be more defect-prone and calculates the effort-aware indicators by utilizing a ManualUP technique which considers a smaller module to be more defect-prone. The two techniques rank the modules based on their LOC in descending order and ascending order, respectively. Since the unsupervised model combines ManualUp and ManualDown, we call it ManualUD. Like the method ALL, ManualUD does not apply any data filter process, and we add it in this question as a baseline method.

5.1.2 Analysis Results of RQ1

1) *Results for BDA and 6 Training Data Filter Based Baseline Methods on the AEEEM Dataset.* Table 2 reports the average values of the six indicators for BDA and six training data filter methods on the AEEEM dataset. The values in bold in the table denote the best performance, which is also the case in all the following tables. It shows that BDA achieves the best average performance values on all indicators. More specifically, compared with the six baseline methods, BDA achieves improvements of 14.9%–102.6% in terms of F-measure, of 6.2%–50.8% in terms of g-mean, of 7.9%–49.8% in terms of Balance, of 8.7%–22.1% in terms of AUC, of 21.3%–82.0% in terms of EARecall, and of 22.9%–176.2% in terms of EAF-measure. The detailed experimental results and performance improvements are available in our online materials. Fig.1 visualizes the corresponding results of statistical test. The methods that have significant differences are drawn in different colors. The p values (all less than 0.05) of the Friedman test indicate that there exist significant differences among the seven methods on all indicators. The Nemenyi test results illustrate that BDA belongs to the top rank group and always ranks the first on all indicators. In addition, BDA performs no significant differences compared with ManualUD in terms of F-measure, g-mean, and Balance, and compared with Yu-Filter in terms of AUC and EAF-measure.

2) *Results for BDA and 6 Training Data Filter Based Baseline Methods on the NASA Dataset.* Table 3 reports the average values of the six indicators for the seven methods on the NASA dataset. It shows that

Table 2. Average Values of 6 Indicators for BDA and 6 Training Data Filter Methods on the AEEEM Dataset

Indicator	ALL	NN-Filter	Peter-Filter	Yu-Filter	HISNN	ManualUD	BDA
F-measure	0.409	0.424	0.408	0.448	0.267	0.471	**0.541**
g-mean	0.599	0.610	0.584	0.616	0.463	0.657	**0.698**
Balance	0.587	0.600	0.578	0.607	0.464	0.644	**0.695**
AUC	0.668	0.663	0.611	0.686	—	0.670	**0.746**
EARecall	0.303	0.322	0.325	0.333	0.274	0.222	**0.404**
EAF-measure	0.273	0.287	0.273	0.292	0.242	0.130	**0.359**

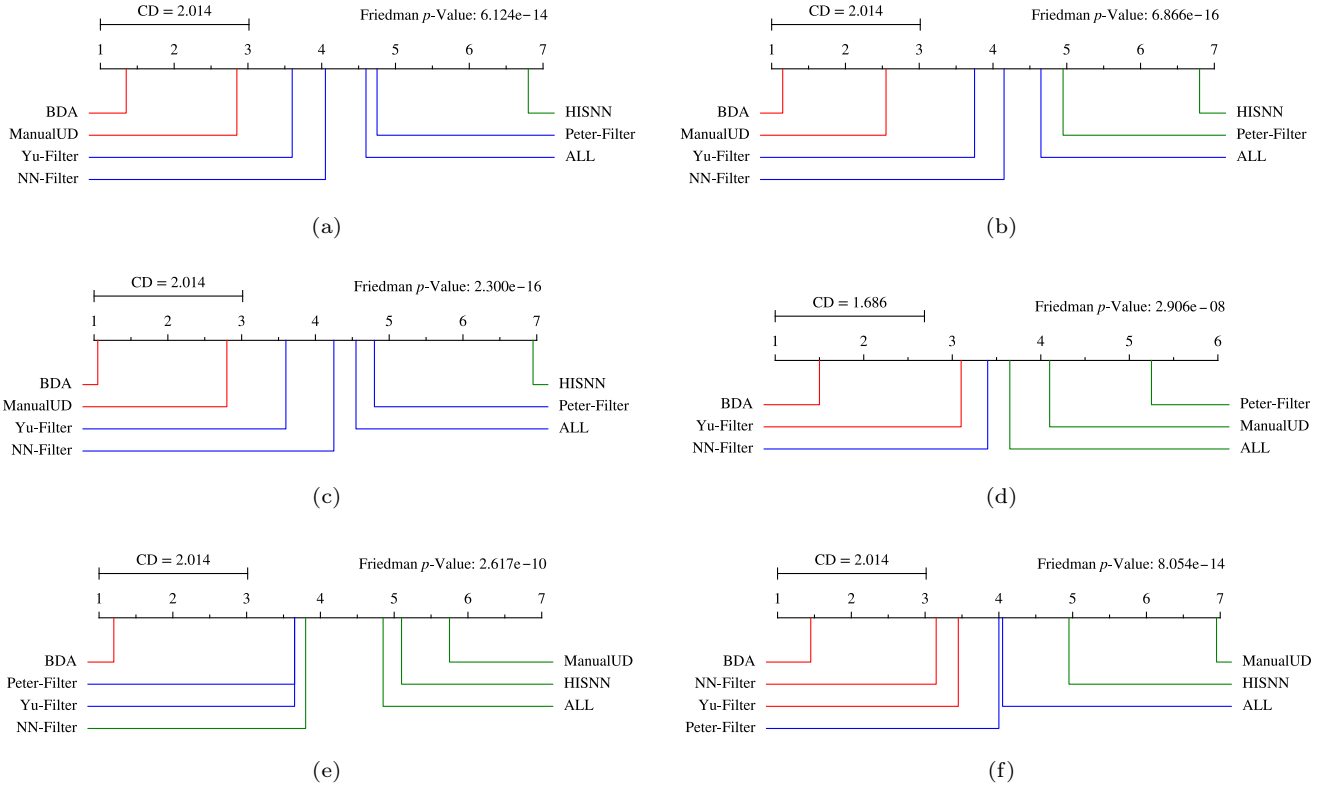


Fig.1. Statistic results with the Nemenyi test among BAD and 6 training data filter based methods on the AEEEM dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

Table 3. Average Values of 6 Indicators for BDA and 6 Training Data Filter Methods on the NASA Dataset

Indicator	ALL	NN-Filter	Peter-Filter	Yu-Filter	HISNN	ManualUD	BDA
F-measure	0.391	0.419	0.315	0.389	0.213	**0.528**	0.489
g-mean	0.609	0.638	0.538	0.615	0.406	0.645	**0.684**
Balance	0.607	0.636	0.536	0.608	0.435	0.640	**0.677**
AUC	0.645	0.686	0.563	0.687	—	0.651	**0.722**
EARecall	0.268	0.262	0.235	0.244	0.230	**0.423**	0.305
EAF-measure	0.218	0.218	0.178	0.218	0.169	**0.263**	0.245

BDA obtains the best average performance values in terms of g-mean, Balance, and AUC, while ManualUD performs the best in terms of the other three indicators. More specifically, compared with the six baseline methods, BDA achieves improvements of 6.0%–68.5% in terms of g-mean, of 5.8%–55.6% in terms of Balance, and of 5.1%–28.2% in terms of AUC. However, BDA is 7.4%, 27.9%, and 6.8% lower than ManualUD in terms of F-measure, EARecall, and EAF-measure, respectively. Fig.2 visualizes the corresponding results of statistical test. The p values (all less than 0.05) of the Friedman test indicate that the seven methods have

significant differences among on all indicators. The results of the Nemenyi test illustrate that BDA belongs to the top rank group on five indicators (excepts for EAF-measure) and ranks the first or second on all indicators. In addition, BAD has no significant differences compared with 3, 4, 4, 3, 1, and 6 baseline methods in terms of the six indicators, respectively.

3) *Results for BDA and 6 Training Data Filter Based Baseline Methods on the SOFTLAB Dataset.* Table 4 reports the average values of the six indicators for the seven methods on the SOFTLAB dataset. It shows that BDA gets the best average performance values on all indicators again. More specifically, compared with the six baseline methods, BDA achieves improvements of 17.1%–77.2% in terms of F-measure, of 7.0%–48.4% in terms of g-mean, of 6.0%–46.5% in terms of Balance, of 2.9%–17.9% in terms of AUC, of 31.5%–89.9% in terms of EARecall, and of 31.3%–94.5% in terms of EAF-measure. Fig.3 visualizes the corresponding results of statistical test. The p values (all less than 0.05) of the Friedman test also indicate that the seven methods have significant performance differences on all indicators. The results of the Nemenyi test illustrate that BDA belongs to the top rank group and ranks the first or second on all indicators. In addition, BDA performs significantly better than all baseline methods in terms of EAF-measure, but has no significant differences compared with 1, 1, 3, 3, and 1 baseline methods in terms of the first five indicators, respectively.

4) *Results for BDA and 6 Training Data Filter Based Baseline Methods on the RELINK Dataset.* Table 5 reports the average values of the six indicators for the seven methods on the RELINK dataset. It shows that ManualUD achieves the best average performance on all indicators. But the average performance by BDA is superior to that by the other baseline methods on five indicators (except for AUC). More specifically, BDA is 7.3%, 8.5%, 8.8%, 5.5%, 19.7%, and 7.1% lower than ManualUD in terms of the six indicators, respectively. Fig.4 visualizes the corresponding results of the statistical test. The p values (all less than 0.05) of the Friedman test indicate that there are significant differences among the seven methods on five indicators except for AUC. The results of the Nemenyi test illustrate that BDA belongs to the top rank group in terms of F-measure, g-mean, EARecall, and EAF-measure. In addition, ManualUD performs significantly better than all other methods in terms of Balance, and all methods perform no significant differences in terms of AUC.

Summary. On average, compared with the six training data filter based baseline methods, BDA achieves average improvements of 30.7%, 14.9%, 14.3%, 6.8%, 35.4%, and 39.1% in terms of six indicators respectively over the four benchmark datasets.

5.2 Results Analysis for RQ2

5.2.1 Analysis Method for RQ2

To answer this question, we select six transfer learning based baseline methods for comparison. The brief descriptions of the six baseline methods are as follows.

• *TCA*. The transfer component analysis (TCA) method[26] only considers the margin distribution differences across the project data.

• *TCA+*. Before performing TCA, the TCA+ method[6] selects a specific data normalization strategy based on some designed rules to preprocess the data of the two projects.

• *CDT*. We design a conditional distribution based transfer learning (CDT) method that only considers the conditional distribution differences between the data of the two projects for comparison.

• *JDT*. JDT (called JDA in [25]) is a joint distributions based transfer learning method that considers both the marginal and the conditional distribution differences with equal weights.

• *TNB*. Transfer naive Bayes (TNB)[5] introduces the weight information of the modules into the Bayes formula.

• *FeSCH*. The feature selection using clusters of hybrid data (FeSCH) method[4] is a two-step feature selection based transfer learning method. This method consists of a feature clustering stage with a density-based clustering method and a feature selection stage with a ranking strategy.

5.2.2 Analysis Reult of RQ2

1) *Results for BDA and 6 Transfer Learning Based Baseline Methods on the AEEEM Dataset.* Table 6 reports the average values of the six indicators for BDA and six transfer learning methods on the AEEEM dataset. It shows that BDA achieves the best average performance values on all indicators. More specifically, compared with the six baseline methods, BDA achieves improvements of 0.9%–15.1% in terms of F-measure, of 2.3%–13.7% in terms of g-mean, of 2.7%–15.4% in terms of Balance, of 0.0%–6.1% in terms of AUC, of 6.3%–17.4% in terms of EARecall, and of 5.3%–20.9% in terms of EAF-measure.

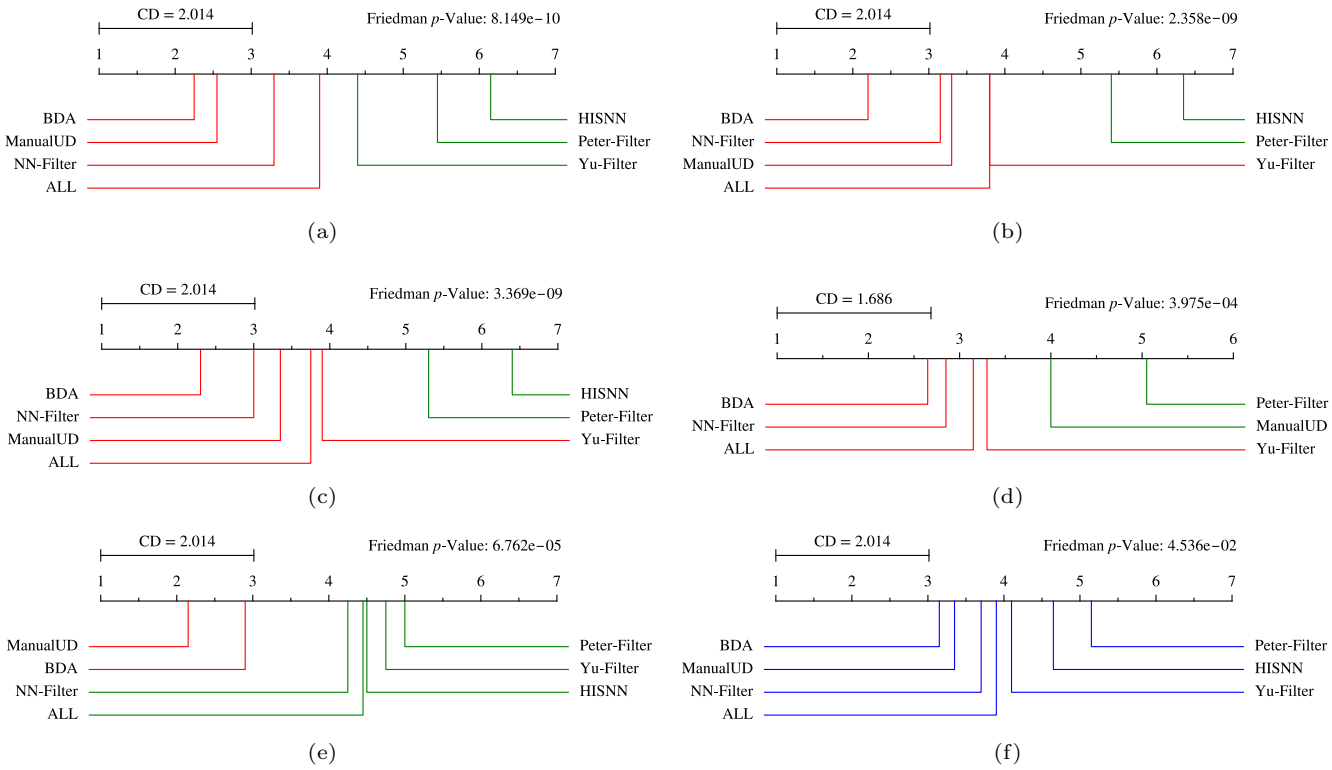


Fig.2. Statistic results with the Nemenyi test among BAD and 6 training data filter based methods on the NASA dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

Table 4. Average Values of 6 Indicators for BDA and 6 Training Data Filter Methods on the SOFTLAB Dataset

Indicator	ALL	NN-Filter	Peter-Filter	Yu-Filter	HISNN	ManualUD	BDA
F-measure	0.534	0.552	0.537	0.522	0.355	0.529	**0.629**
g-mean	0.696	0.710	0.699	0.693	0.512	0.661	**0.760**
Balance	0.689	0.701	0.694	0.687	0.507	0.646	**0.743**
AUC	0.748	0.768	0.764	0.760	—	0.670	**0.790**
EARecall	0.258	0.254	0.240	0.198	0.286	0.284	**0.376**
EAF-measure	0.243	0.240	0.222	0.193	0.239	0.164	**0.319**

Table 5. Average Values of 6 Indicators for BDA and 6 Training Data Filter Methods on the RELINK Dataset

Indicator	ALL	NN-Filter	Peter-Filter	Yu-Filter	HISNN	ManualUD	BDA
F-measure	0.543	0.534	0.569	0.546	0.500	**0.698**	0.647
g-mean	0.642	0.635	0.632	0.636	0.581	**0.703**	0.643
Balance	0.633	0.624	0.626	0.629	0.571	**0.701**	0.639
AUC	0.704	0.700	0.702	0.699	—	**0.704**	0.665
EARecall	0.233	0.219	0.280	0.231	0.267	**0.451**	0.362
EAF-measure	0.258	0.243	0.295	0.253	0.278	**0.395**	0.367

Fig.5 visualizes the corresponding results of statistical test. The p values indicate that there exist significant differences among the seven methods on all indicators. The results of the Nemenyi test illustrate that BDA always belongs to the top rank group and ranks the first or second in terms of all indicators. In addition, BDA has no significant differences compared with JDT and CDT on F-measure, g-mean, AUC, and EAF-measure, and compared with JDT on Balance and EARecall.

2) *Results for BDA and 6 Transfer Learning Based Baseline Methods on the NASA Dataset.* Table 7 reports the average values of the six indicators for the seven methods on the NASA dataset. It shows that

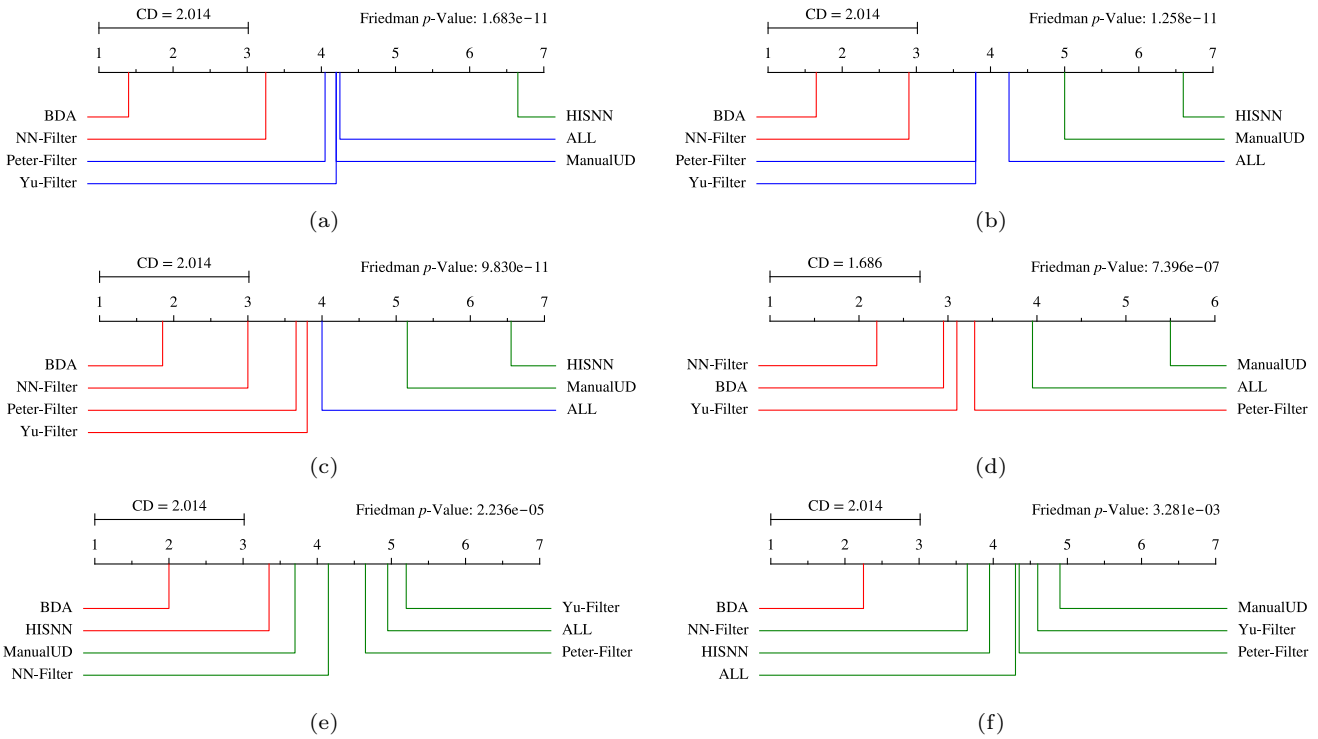


Fig.3. Statistic results with the Nemenyi test among BAD and 6 training data filter based methods on the SOFTLAB dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

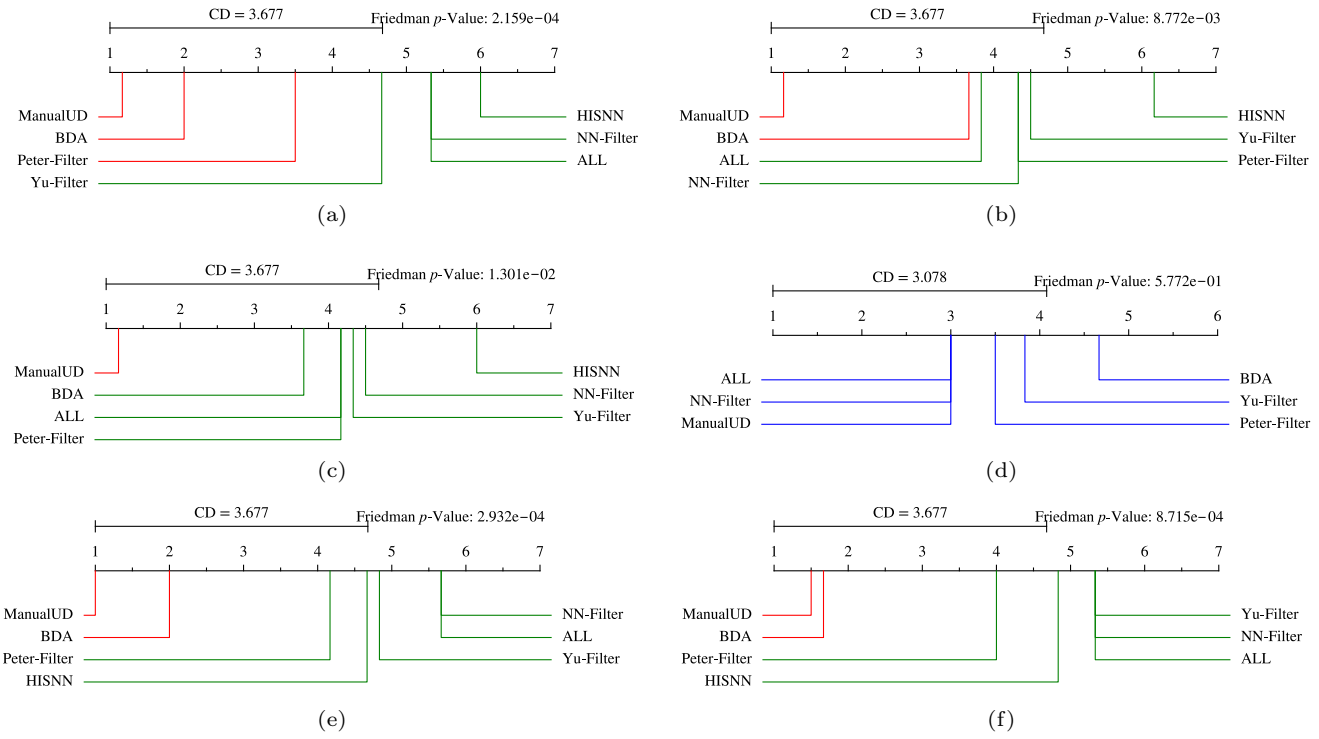


Fig.4. Statistic results with the Nemenyi test among BAD and 6 training data filter based methods on the RELINK dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

Table 6. Average Values of 6 Indicators for BDA and 6 Transfer Learning Methods on the AEEEM Dataset

Indicator	TCA	TCA+	CDT	JDT	TNB	FeSCH	BDA
F-measure	0.512	0.470	0.514	0.519	0.536	0.475	**0.541**
g-mean	0.680	0.638	0.680	0.682	0.614	0.647	**0.698**
Balance	0.674	0.645	0.676	0.677	0.602	0.637	**0.695**
AUC	**0.746**	0.728	0.735	0.741	0.725	0.703	**0.746**
EARecall	0.363	0.357	0.375	0.380	0.374	0.344	**0.404**
EA*F*-measure	0.326	0.320	0.337	0.341	0.297	0.308	**0.359**

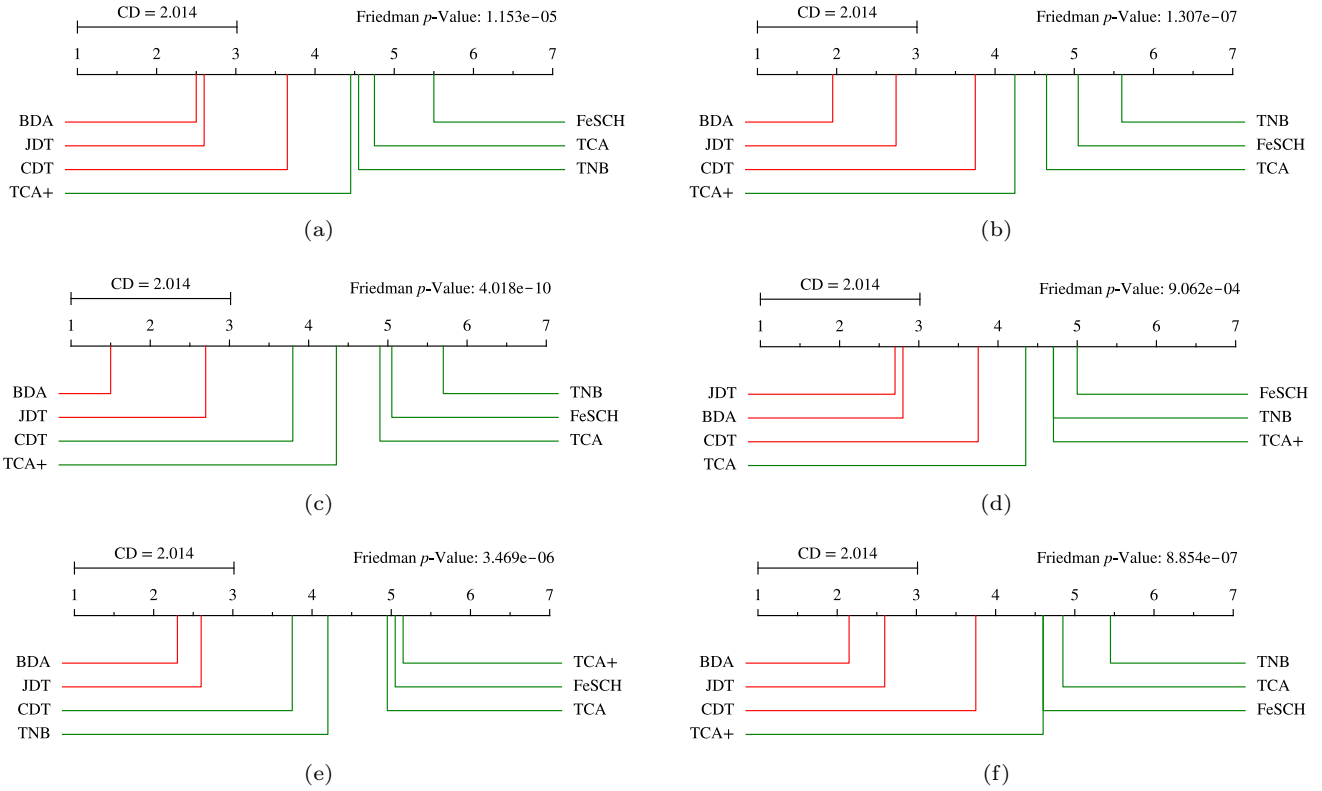


Fig.5. Statistic results with the Nemenyi test among BAD and 6 transfer learning based methods on the AEEEM dataset in terms of 6 indicators. (a) *F*-measure. (b) *g*-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EA*F*-measure.

Table 7. Average Values of 6 Indicators for BDA and 6 Transfer Learning Methods on the NASA Dataset

Indicator	TCA	TCA+	CDT	JDT	TNB	FeSCH	BDA
F-measure	0.454	0.338	0.466	0.459	0.398	0.461	**0.489**
g-mean	0.663	0.526	0.671	0.667	0.621	0.667	**0.684**
Balance	0.662	0.572	0.668	0.664	0.613	0.660	**0.677**
AUC	0.715	0.644	0.717	0.720	0.708	**0.726**	0.722
EARecall	0.242	0.171	0.267	0.249	0.248	**0.330**	0.305
EA*F*-measure	0.206	0.148	0.223	0.210	0.222	**0.268**	0.245

BDA achieves the best average performance values on the first three indicators, while FeSCH achieves the best average performance values on the last three indicators. More specifically, compared with the six baseline methods, BDA achieves improvements of 4.9%–44.7% in terms of *F*-measure, of 1.9%–30.0% in terms of *g*-mean, and of 1.3%–18.4% in terms of Balance. However, BDA is 0.6%, 7.6%, and 8.6% lower than FeSCH in terms of AUC, EARecall, and EA*F*-measure, respectively. Fig.6 visualizes the corresponding results of the statistical test. The p values show that there exist significant differences among the seven methods on all indicators. The results of the Nemenyi test illustrate that BDA belongs to the top rank group but has no signifi-

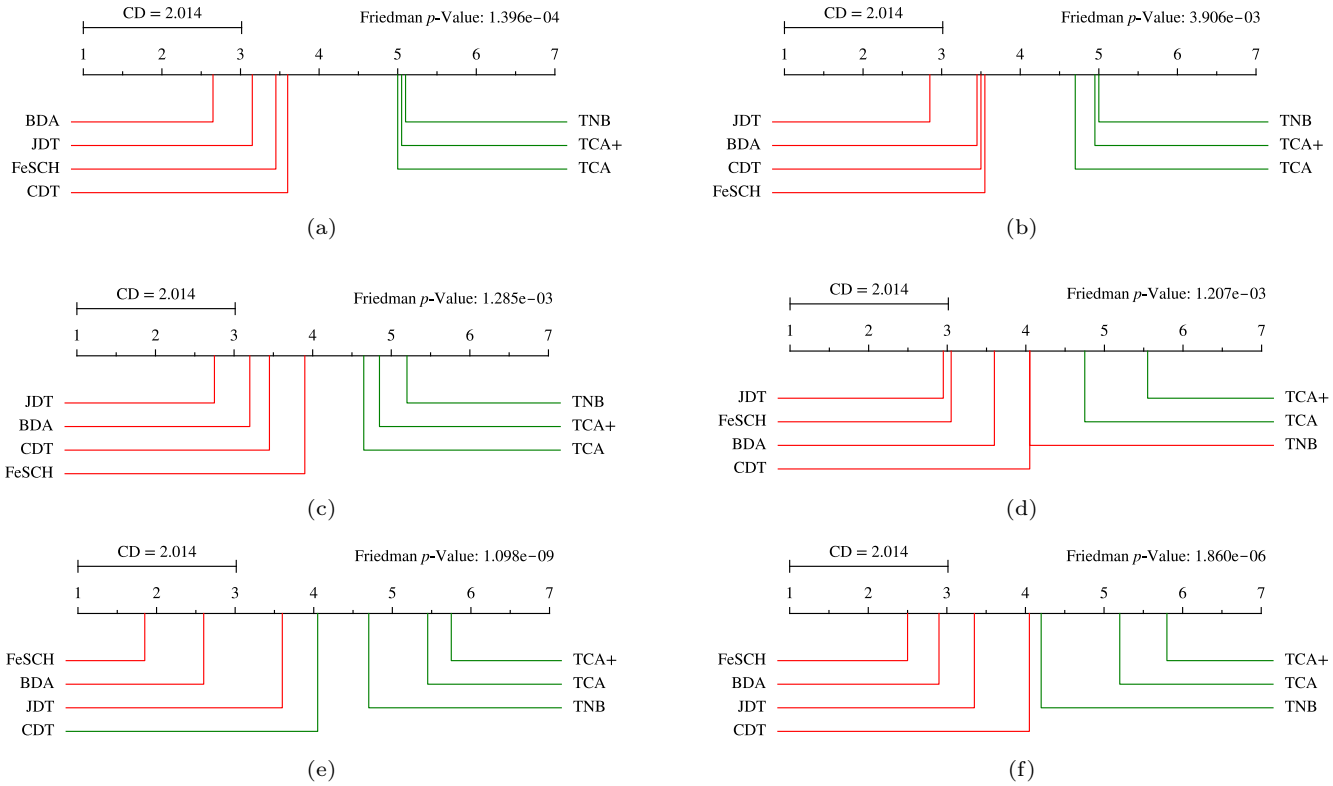


Fig.6. Statistic results with the Nemenyi test among BAD and 6 transfer learning based methods on the NASA dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

Table 8. Average Values of 6 Indicators for BDA and 6 Transfer Learning Methods on the SOFTLAB Dataset

Indicator	TCA	TCA+	CDT	JDT	TNB	FeSCH	BDA
F-measure	0.529	0.463	0.564	0.566	0.536	0.464	**0.629**
g-mean	0.690	0.620	0.713	0.720	0.588	0.629	**0.760**
Balance	0.688	0.617	0.702	0.715	0.576	0.636	**0.743**
AUC	0.757	0.683	0.752	0.756	**0.801**	0.704	0.790
EARecall	0.183	0.227	0.295	0.249	0.254	0.214	**0.376**
EAF-measure	0.174	0.195	0.248	0.223	0.200	0.200	**0.319**

cant differences compared with JDT and FeSCH on all indicators, compared with CDT on five indicators except for EARecall, and compared with TNB on AUC.

3) *Results for BDA and 6 Transfer Learning Based Baseline Methods on the SOFTLAB Dataset.* Table 8 reports the average values of the six indicators for the seven methods on the SOFTLAB dataset. It shows that BDA achieves the best average performance values on five indicators except for AUC. More specifically, compared with the six baseline methods, BDA achieves improvements of 11.1%–35.9% in terms of F-measure, of 5.6%–29.3% in terms of g-mean, of 3.9%–29.0% in terms of Balance, of 27.5%–105.5% in terms of EARecall, and of 28.6%–83.3% in terms of EAF-measure. However, BDA is 1.4% lower than TNB in terms of AUC. Fig.7 visualizes the corresponding results of statistical test. The p values show that there exist significant differences among the seven methods on all indicators. The results of the Nemenyi test illustrate that BDA belongs to the top rank group and always ranks the first or second on all indicators. In addition, BDA has no significant differences compared with JDT and CDT on g-mean, Balance, and EAF-measure, compared with JDT on F-measure, and compared with TNB on AUC.

4) *Results for BDA and 6 Transfer Learning Based Baseline Methods on the RELINK Dataset.* Table 9 reports the average values of the six indicators for the seven methods on the RELINK dataset. It shows that

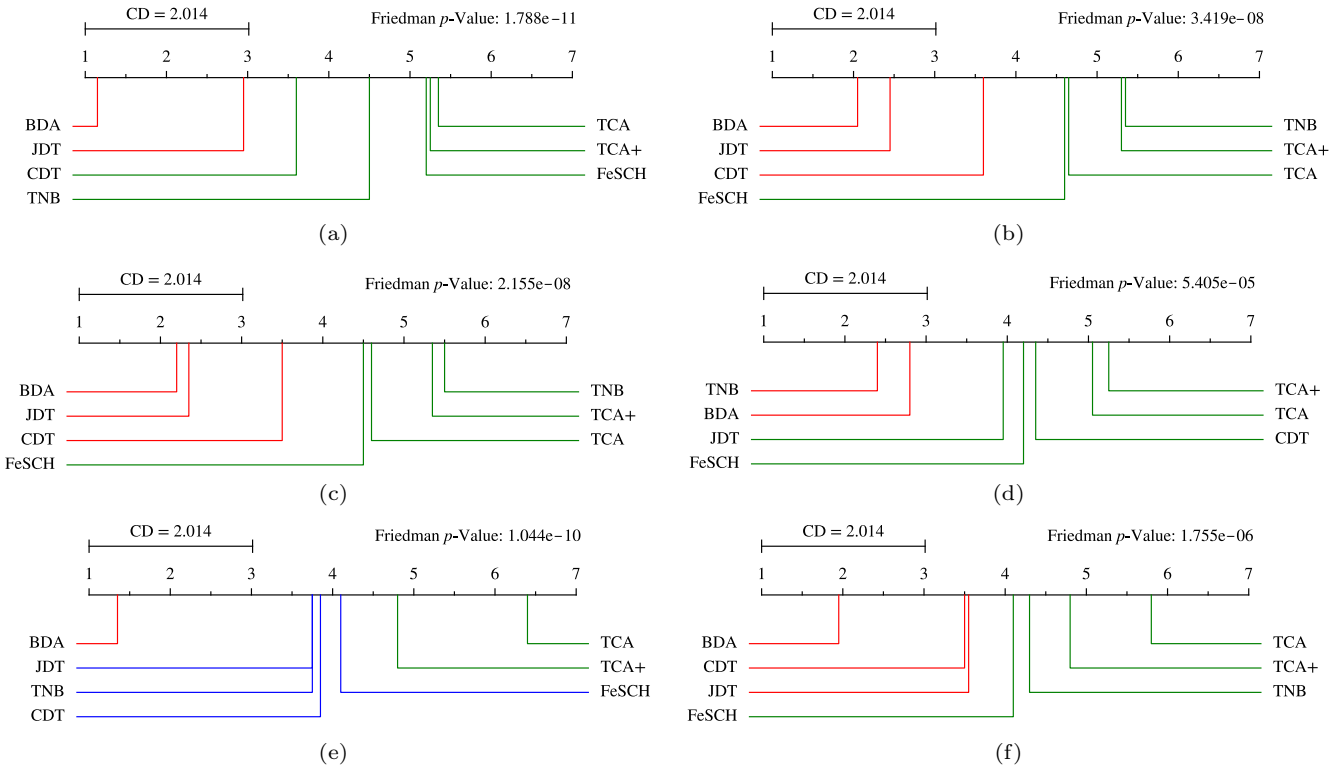


Fig.7. Statistic results with the Nemenyi test among BAD and 6 transfer learning based methods on the SOFTLAB dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

Table 9. Average Values of 6 Indicators for BDA and 6 Transfer Learning Methods on the RELINK Dataset

Indicator	TCA	TCA+	CDT	JDT	TNB	FeSCH	BDA
F-measure	0.537	0.399	0.571	0.578	0.587	0.545	**0.647**
g-mean	0.641	0.439	0.646	0.600	0.603	**0.655**	0.643
Balance	0.630	0.487	0.639	0.599	0.595	**0.640**	0.639
AUC	0.676	0.621	0.674	0.645	0.685	**0.733**	0.665
EARecall	0.229	0.335	0.277	0.320	0.301	0.219	**0.362**
EAF-measure	0.253	0.324	0.297	0.323	0.318	0.246	**0.367**

BDA achieves the best average performance values on F-measure, EARecall and EAF-measure, while FeSCH achieves the best performance values on other three indicators. More specifically, compared with the six baseline methods, BDA achieves improvements of 10.2%–62.2% in terms of F-measure, of 8.1%–65.3% in terms of EARecall, and of 13.3%–49.2% in terms of EAF-measure. However, BDA is 0.5% and 1.8% lower than CDT and FeSCH in terms of g-mean, 0.2% lower than FeSCH in terms of Balance, and 1.6%, 1.3%, 2.9%, and 9.3% lower than TCA, CDT, TNB, and FeSCH respectively in terms of AUC. Fig.8 visualizes the corresponding results of the statistical test. The p values reveal that there are significant performance differences among the seven methods on three indicators except for g-mean, Balance, and AUC. The results of the Nemenyi test illustrate that BDA has no significant differences compared with TNB and CDT on F-measure, compared with JDT, TCA+, and TNB on EARecall and EAF-measure, and compared with all baseline methods on g-mean, Balance, and AUC.

Summary. On average, compared with the six transfer learning based baseline methods, BDA achieves average improvements of 16.9%, 10.1%, 8.7%, 2.7%, 32.9%, and 28.4% in terms of six indicators respectively over the four benchmark datasets.

6 Discussion

In this section, we discuss the impacts of the different parameter settings and classifiers on our experimental results.

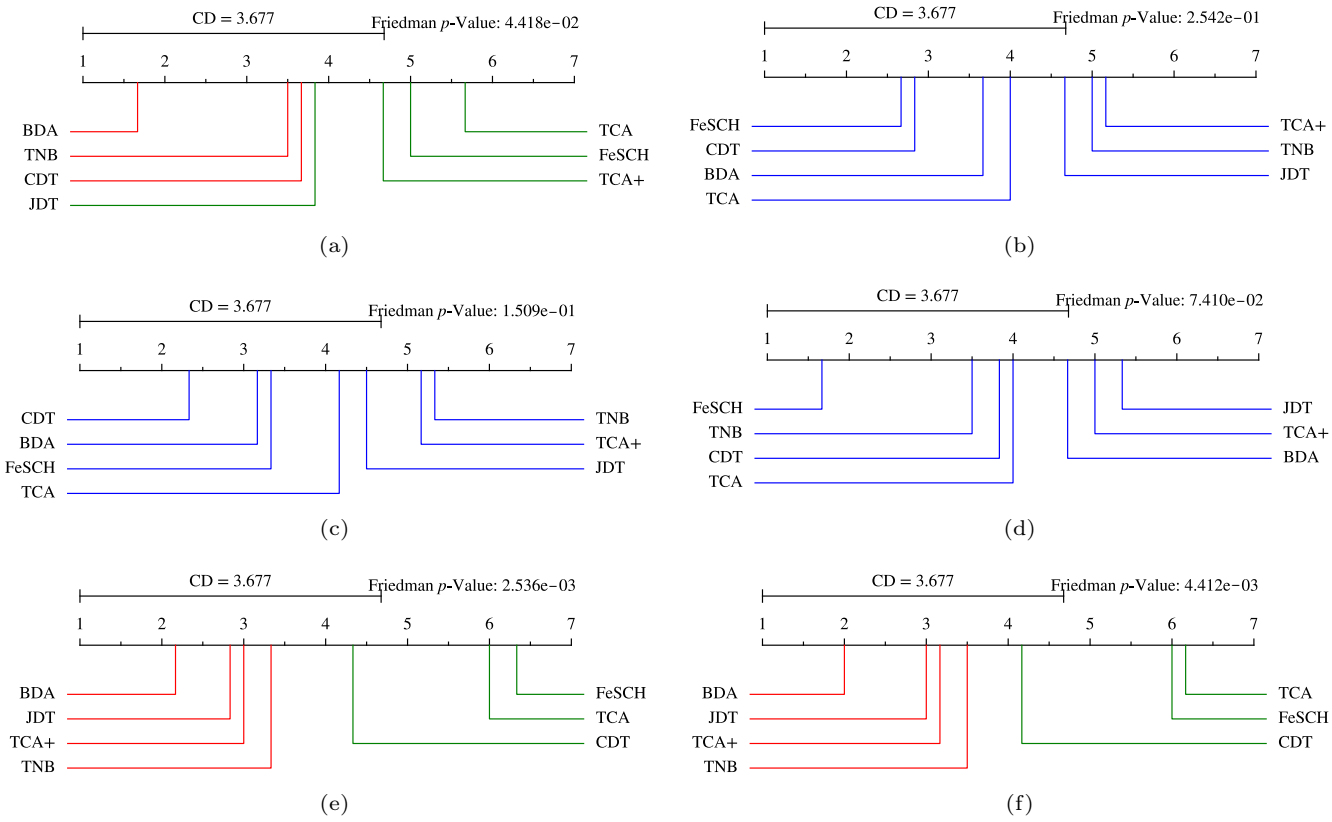


Fig.8. Statistic results with the Nemenyi test among BAD and 6 transfer learning based methods on the RELINK dataset in terms of 6 indicators. (a) F-measure. (b) g-mean. (c) Balance. (d) AUC. (e) EARecall. (f) EAF-measure.

6.1 Parameter Setting of the Selected Feature Dimensions

After transforming the original data of two projects into a new feature space, we empirically select 5% of original feature dimensions to carry on our experiments. In this subsection, we discuss the impacts of different feature dimensions on the CPDP performance of BDA. For this purpose, we set 20 different thresholds for feature dimensions, from 5% to 100% with a step of 5%, for comparison. Fig.9 depicts the six average indicator values of BDA when varying the thresholds of feature dimensions on each benchmark dataset.

From Fig.9, we observe that on the AEEEM dataset, the six average indicator values have a little change under different thresholds. On the NASA dataset, the six average indicator values under thresholds larger than 15% keep stable and a little higher than that under thresholds of 5% and 10%. On the SOFTLAB dataset, the six average indicator values under thresholds larger than 25% keep almost unchanged and lower than that under small thresholds. Overall, BDA with small thresholds achieves better average indicator values. On the RELINK dataset, the six average indicator values under thresholds larger than 40% keep almost stable. When the threshold is lower than 40%, the average F-measure, EARecall, and EAF-measure values decrease with the increasing of the threshold, while the average g-mean, Balance, and AUC values oscillate.

Overall, BDA achieves similar performance with larger thresholds and better performance with smaller thresholds (except for the NASA dataset). From the above analysis, the thresholds within 5%–20% can be a better choice for BDA.

6.2 Parameter Setting of the Regularization Parameter λ

As mentioned in Subsection 4.6, we set the regularization parameter λ value to 0.1 without any prior knowledge. In this subsection, we discuss the impacts of different parameter λ values on the CPDP performance of BDA. For this purpose, we empirically set 15 different parameter λ values, including 0.001–0.009 with a step of 0.002, 0.01–0.09 with a step of 0.02, and 0.1–0.9 with a step of 0.2 for comparison. Fig.10 depicts the six

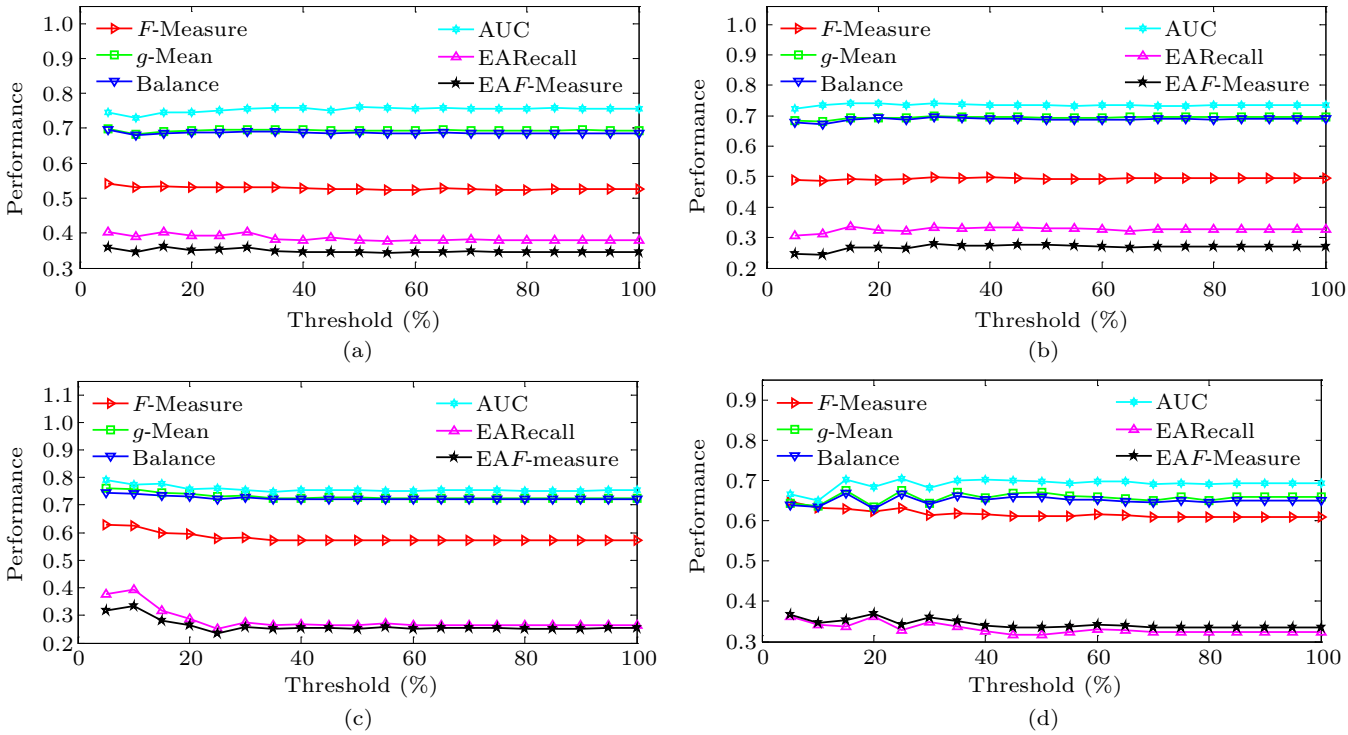


Fig.9. Six average indicator values of BDA when varying the thresholds of feature dimension on each benchmark dataset. (a) AEEEM. (b) NASA. (c) SOFTLAB. (d) RELINK.

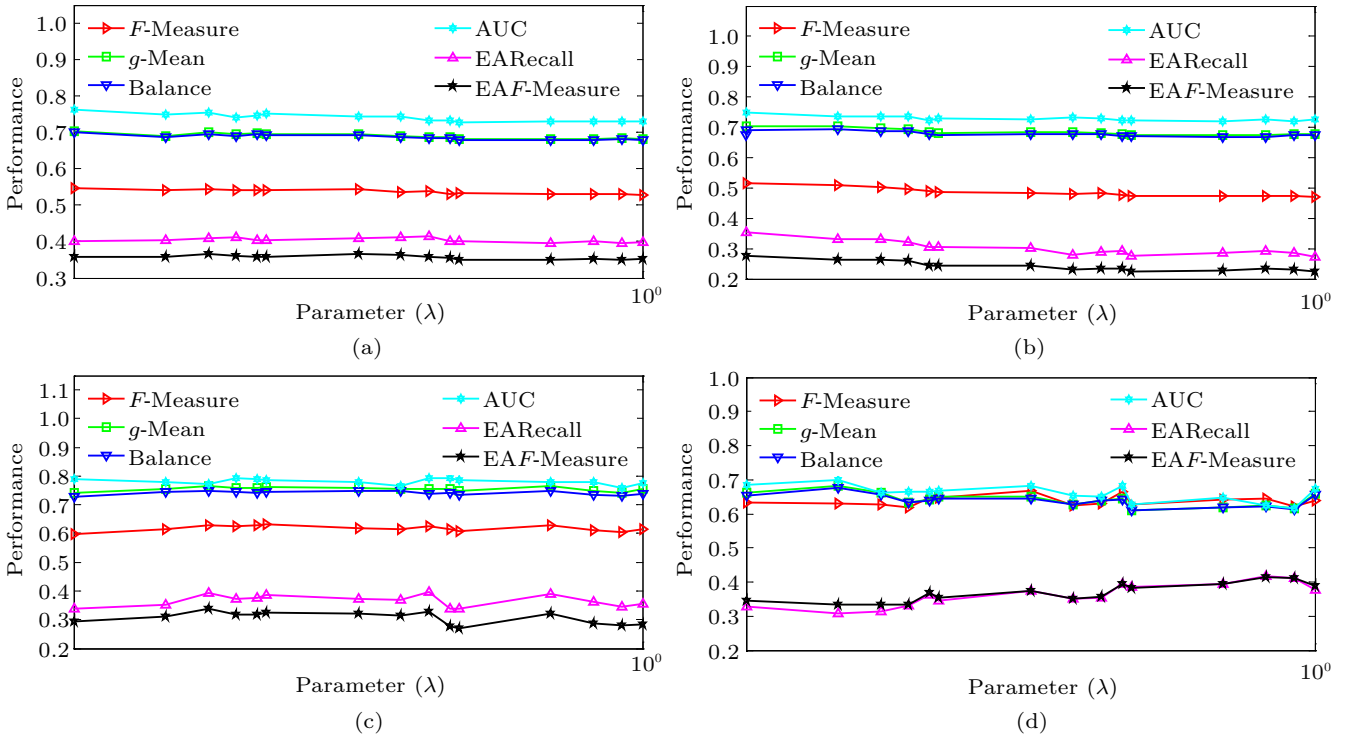


Fig.10. Six average indicator values of BDA with different parameter λ on each benchmark dataset. (a) AEEEM. (b) NASA. (c) SOFTLAB. (d) RELINK.

average indicator values of BDA with different λ values on each benchmark dataset.

From Fig.10, we observe that on the AEEEM dataset, the six average indicator values are barely affected by different λ values. On the NASA dataset, the whole trend is that the average indicator values decrease with the increasing of λ. On SOFTLAB dataset, the overall trend is opposite to that on the NASA dataset. On the RELINK dataset, the average EARecall and EAF-measure values increase with the increasing of λ. Whereas the average F-measure, g-mean, Balance and AUC values fluctuate with different λ, thus we cannot find a general rule for the four indicators.

From the above analysis, the parameter λ has different impacts on the average performance values of BDA over different benchmark datasets. In other words, the selection of the optimum parameter λ depends on the specific dataset studied. In the future, we will try to investigate what property of the dataset affects the selection of this parameter.

6.3 Impacts of Different Classifiers on the Performance of BDA

As mentioned in Subsection 4.3, we choose LR classifier as our basic classification model. In this subsection, we discuss the impacts of different classifiers on the CPDP performance of BDA. For this purpose, we choose five additional classifiers for observations, including naive Bayes (NB), nearest neighbor (NN), classification and regression tree (CART), random forest (RF), and one ensemble classifier Adaboost. These classifiers are widely studied in the software defect prediction domain[30,59]. Fig.11 shows the average performance values in terms of the six indicators on each dataset.

From Fig.11, we can observe that, over the AEEEM dataset, BDA with the LR classifier is obviously supe-

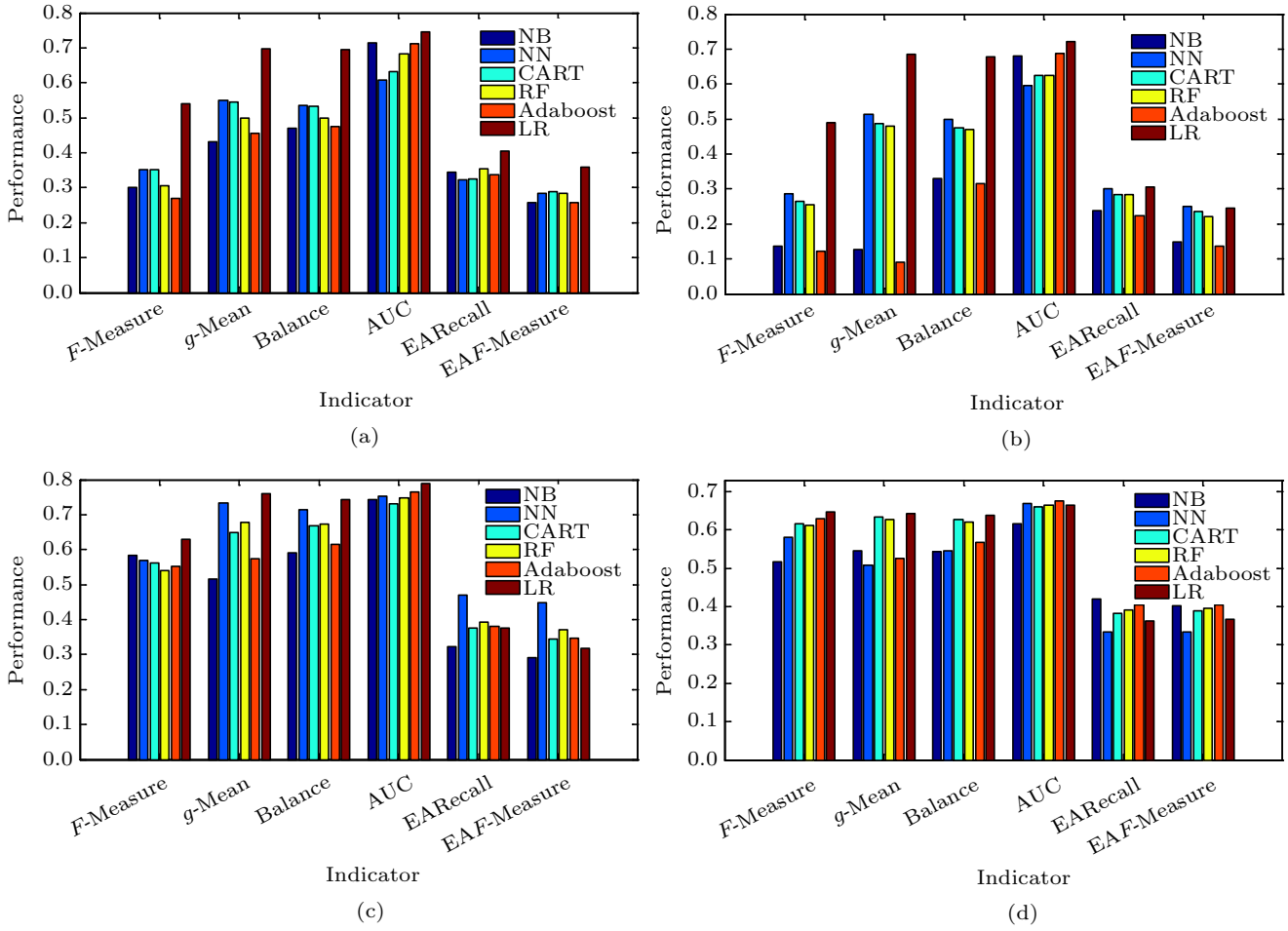


Fig.11. Six average indicator values of BDA with different classifiers on each benchmark dataset. (a) AEEEM. (b) NASA. (c) SOFTLAB dataset. (d) RELINK.

rior to BDA with other five classifiers; over the NASA dataset, BDA with the LR classifier is obviously better than BDA with other five classifiers in terms of four traditional indicators, but achieves the similar performance to BDA with the NN classifier in terms of two effort-aware indicators; over the SOFTLAB and RELINK datasets, BDA with the LR classifier achieves the best average performance compared with BDA with other five classifiers in terms of F-measure, g-mean, and Balance, but worse average performance in terms of two effort-aware indicators.

Overall, BDA with LR classifier performs well in most cases. Thus, LR is the most appropriate classifier for our CPDP method BDA.

7 Threats to Validity

7.1 Threats to Internal Validity

This kind of threats mainly pays attention to the impacts of uncontrolled factors on the experimental results, especially the unexpected faults in the implementation of the methods. To reduce such threats, for the methods whose source code is online available (such as the TCA method), we use the supplied source code to avoid the possible inconsistency of reimplementation. For the methods without source code, we try our best to guarantee the accuracy of the implementation by carefully following the corresponding details in the original papers.

7.2 Threats to External Validity

This kind of threats concerns the generalization of our experimental results to other defect data. To minimize such threats, we choose four benchmark datasets including 18 open source projects. These projects are developed with Java or C languages, and the modules are extracted at function, class or file level. All these guarantee the diversity of our studied corpora. We conduct the experiments on total 66 cross project pairs, which enables our experimental results to have a certain representation. However, we still could not guarantee that our results keep consistent on closed software projects or on open source projects developed with other programming languages.

7.3 Threats to Conclusion Validity

This kind of threats concentrates on the appropriate usage of the statistical techniques. In this work, we use the Friedman test and the Nemenyi test to check the statistically significant differences between BDA and the baseline methods. Unlike the ANOVA test[60] and Scott-Knott test[30,61] which are based on an assumption that the performance values should fulfill the normality of both the residuals assumption and the homoscedasticity assumption[62], the statistical tests used in this work are non-parametric, which means that they do not require the analyzed data to follow a particular distribution. Thus, using the Friedman test and Nemenyi test makes our statistical analysis more rigorous.

7.4 Threats to Construct Validity

This kind of threats focuses on whether the used performance indicators can represent the real-world requirement of the defect prediction. To alleviate such threats, we use four traditional indicators and two effort-aware indicators. The selected traditional indicators are deemed as comprehensive indicators for defect prediction. In addition, we use two recently proposed effort-aware indicators which are verified to be suitable for performance evaluation in defect prediction scenario[10,54]. Using these indicators enables us to conduct a comprehensive performance evaluation.

8 Conclusions

Due to the limitation of existing transfer learning based CPDP methods that does not consider both marginal and conditional distribution differences of data between different projects, in this work, we introduced a novel BDA method to narrow the gap by combining the two kinds of distribution differences with adaptive weights. We conducted experiments on 18 open-source projects from four benchmark datasets with four traditional and two effort-aware indicators. The experimental results showed that over the four datasets, our method BDA improves by an average of 23.8%, 12.5%, 11.5%, 4.7%, 34.2%, and 33.7% compared with 12 baseline methods (including six training data filter methods and six transfer learning methods) in terms of the six indicators, respectively.

Since defect data naturally have the class imbalance property, in the future work, we will incorporate this issue into BDA. We will also explore how to choose the optimal parameter λ for different benchmark datasets.

References

[1] Mei H. Understanding "software-defined" from an OS perspective: Technical challenges and research issues. *Sci.*

China-Inf. Sci., 2017, 60(12): Article No. 126101.

[2] Lyu M R. Handbook of Software Reliability Engineering. McGraw-Hill, 1996.

[3] Xu Z, Xuan J, Liu J, Cui X. MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In *Proc. the 23rd Int. Conf. Software Analysis, Evolution, and Reengineering*, March 2016, pp.370-381.

[4] Ni C, Liu W S, Chen X, Gu Q, Chen D, Huang G D. A cluster based feature selection method for cross-project software defect prediction. *J. Comput. Sci. Technol.*, 2017, 32(6): 1090-1107.

[5] Ma Y, Luo G, Zeng X, Chen A. Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.*, 2012, 54(3): 248-256.

[6] Nam J, Pan S J, Kim S. Transfer defect learning. In *Proc. the 35th Int. Conf. Software Engineering*, May 2013, pp.382-391.

[7] Wang J, Chen Y, Hao S, Feng W, Shen Z. Balanced distribution adaptation for transfer learning. In *Proc. the 17th Int. Conf. Data Mining*, November 2017, pp.1129-1134.

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

[9] Fawcett T. An introduction to ROC analysis. *Pattern Recognit. Lett.*, 2006, 27(8): 861-874.

[10] Huang Q, Xia X, Lo D. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In *Proc. the 2017 Int. Conf. Software Maintenance and Evolution*, September 2017, pp.159-170.

[11] Xu Z, Li S, Tang Y *et al.* Cross version defect prediction with representative data via sparse subset selection. In *Proc. the 26th Int. Conf. Program Comprehension*, May 2018, pp.132-143.

[12] Briand L C, Melo W L, Wüst J. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans. Softw. Eng.*, 2002, 28(7): 706-720.

[13] Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proc. the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. Foundations of Software Engineering*, August 2009, pp.91-100.

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

[15] Peters F, Menzies T, Marcus A. Better cross company defect prediction. In *Proc. the 10th Working Conf. Mining Software Repositories*, May 2013, pp.409-418.

[16] Kawata K, Amasaki S, Yokogawa T. Improving relevancy filter methods for cross-project defect prediction. In *Proc. the 3rd Int. Conf. Applied Computing and Information Technology*, July 2015, pp.1-12.

[17] Yu X, Zhang J, Zhou P, Liu J. A data filtering method based on agglomerative clustering. In *Proc. the 29th Int. Conf. Software Engineering and Knowledge Engineering*, July 2017, pp.392-397.

[18] He P, Li B, Zhang D, Ma Y. Simplification of training data for cross-project defect prediction. arXiv:1405.0773, 2014. https://arxiv.org/abs/1405.0773, June 2019.

[19] He P, Ma Y, Li B. TDSelector: A training data selection method for cross-project defect prediction. arXiv:1612.09065, 2016. https://arxiv.org/abs/1612.09065, Jun. 2019.

[20] He P, He Y, Yu L, Li B. An improved method for cross-project defect prediction by simplifying training data. *Math. Probl. Eng.*, 2018, 2018: Article No. 2650415.

[21] Chen L, Fang B, Shang Z, Tang Y. Negative samples reduction in cross-company software defects prediction. *Inf. Softw. Technol.*, 2015, 62: 67-77.

[22] Ryu D, Jang J I, Baik J. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw. Qual. J.*, 2017, 25(1): 235-272.

[23] Liu C, Yang D, Xia X, Yan M, Zhang X. A two-phase transfer learning model for cross-project defect prediction. *Inf. Softw. Technol.*, 2019, 107: 125-136.

[24] Forbes C, Evans M, Hastings N, Peacock B. Statistical Distributions (4th edition). John Wiley and Sons, 2010.

[25] Long M, Wang J, Ding G, Sun J, Yu P S. Transfer feature learning with joint distribution adaptation. In *Proc. the 2013 IEEE Int. Conf. Computer Vision*, December 2013, pp.2200-2207.

[26] Pan S J, Tsang I W, Kwok J T, Yang Q. Domain adaptation via transfer component analysis. *IEEE Trans. Neural Networks*, 2011, 22(2): 199-210.

[27] D'Ambros M, Lanza M, Robbes R. Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empir. Softw. Eng.*, 2012, 17(4/5): 531-577.

[28] Shepperd M, Song Q, Sun Z, Mair C. Data quality: Some comments on the NASA software defect datasets. *IEEE Trans. Softw. Eng.*, 2013, 39(9): 1208-1215.

[29] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Trans. Softw. Eng.*, 2008, 34(4): 485-496.

[30] Ghotra B, McIntosh S, Hassan A E. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proc. the 37th Int. Conf. Software Engineering*, May 2015, pp.789-800.

[31] Xu Z, Liu J, Luo X, Yang Z, Zhang Y, Yuan P, Tang Y, Zhang T. Software defect prediction based on kernel PCA and weighted extreme learning machine. *Inf. Softw. Technol.*, 2019, 106: 182-200.

[32] Xu Z, Liu J, Yang Z, An G, Jia X. The impact of feature selection on defect prediction performance: An empirical comparison. In *Proc. the 27th Int. Symp. Software Reliability Engineering*, October 2016, pp.309-320.

[33] Xu Z, Yuan P, Zhang T, Tang Y, Li S, Xia Z. HDA: Cross-project defect prediction via heterogeneous domain adaptation with dictionary learning. *IEEE Access*, 2018, 6: 57597-57613.

[34] Jing X Y, Wu F, Dong X, Qi F, Xu B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proc. the 10th Joint Meeting on Foundations of Software Engineering*, August 31–September 4, 2015, pp.496-507.

[35] Wu R, Zhang H, Kim S, Cheung S C. ReLink: Recovering links between bugs and changes. In *Proc. the 19th ACM SIGSOFT Symp. and the 13th European Conf. Foundations of Software Engineering*, September 2011, pp.15-25.

[36] Han J, Pei J, Kamber M. Data mining: Concepts and Techniques (3rd edition). Morgan Kaufmann, 2011.

[37] Xia X, David L O, Pan S J, Nagappan N, Wang X. HYDRA: Massively compositional model for cross-project defect prediction. *IEEE Trans. Softw. Eng.*, 2016, 42(10): 977-998.

[38] Yang Y, Zhou Y, Lu H, Chen L, Chen Z, Xu B, Zhang Z. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *IEEE Trans. Softw. Eng.*, 2015, 41(4): 331-357.

[39] Nam J, Kim S. CLAMI: Defect prediction on unlabeled datasets (T). In *Proc. the 30th Int. Conf. Automated Software Engineering*, November 2015, pp.452-463.

[40] Yang Y, Harman M, Krinke J *et al.* An empirical study on dependence clusters for effort-aware fault-proneness prediction. In *Proc. the 31st IEEE/ACM Int. Conf. Automated Software Engineering*, September 2016, pp.296-307.

[41] Nam J, Fu W, Kim S *et al.* Heterogeneous defect prediction. *IEEE Trans. Softw. Eng.*, 2018, 44(9): 874-896.

[42] Li Z, Jing X Y, Zhu X, Zhang H. Heterogeneous defect prediction through multiple kernel learning and ensemble learning. In *Proc. the 2017 Int. Conf. Software Maintenance and Evolution*, Sept. 2017, pp.91-102.

[43] Li Z, Jing X Y, Zhu X, Zhang H, Xu B, Ying S. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Trans. Softw. Eng.*, 2019, 45(4): 391-411.

[44] Li Z, Jing X Y, Wu F, Zhu X, Xu B, Ying S. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Autom. Softw. Eng.*, 2018, 25(2): 201-245.

[45] Fan R E, Chang K W, Hsieh C J, Wang X R, Lin C J. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 2008, 9: 1871-1874.

[46] Sasaki Y. The truth of the *F*-measure. *Teach Tutor Mater*, 2007, 1(5): 1-5.

[47] Jiang Y, Cukic B, Ma Y. Techniques for evaluating fault prediction models. *Empir. Softw. Eng.*, 2008, 13(5): 561-595.

[48] Liparas D, Angelis L, Feldt R. Applying the Mahalanobis-Taguchi strategy for software defect diagnosis. *Autom. Softw. Eng.*, 2012, 19(2): 141-165.

[49] Jing X Y, Wu F, Dong X, Xu B. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans. Softw. Eng.*, 2017, 43(4): 321-339.

[50] Wang S, Yao X. Using class imbalance learning for software defect prediction. *IEEE Trans. Reliab.*, 2013, 62(2): 434-443.

[51] Ryu D, Jang J I, Baik J. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *J. Comput. Sci. Technol.*, 2015, 30(5): 969-980.

[52] Li M, Zhang H, Wu R *et al.* Sample-based software defect prediction with active and semi-supervised learning. *Autom. Softw. Eng.*, 2012, 19(2): 201-230.

[53] Ling C X, Huang J, Zhang H. AUC: A statistically consistent and more discriminating measure than accuracy. In *Proc. the 18th Int. Joint Conf. Artificial Intelligence*, August 2003, pp.519-524.

[54] Huang Q, Xia X, Lo D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empir. Softw. Eng.*, doi:10.1007/s10664-018-9661-2.

[55] Demšar J. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 2006, 7: 1-30.

[56] Mende T, Koschke R. Effort-aware defect prediction models. In *Proc. the 14th European. Conf. Software Maintenance and Reengineering*, March 2010, pp.107-116.

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

[58] Zhou Y, Yang Y, Lu H *et al.* How far we have progressed in the journey? An examination of cross-project defect prediction. *ACM Trans. Software Eng. Method.*, 2018, 27(1): Article No. 1.

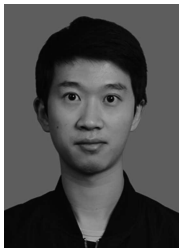
[59] Tantithamthavorn C, McIntosh S, Hassan A E *et al.* The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Softw. Eng.*, 2019, 45(7): 683-672.

[60] Shepperd M, Bowes D, Hall T. Researcher bias: The use of machine learning in software defect prediction. *IEEE Trans. Softw. Eng.*, 2014, 40(6): 603-616.

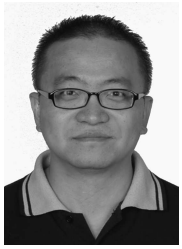
[61] Tantithamthavorn C, McIntosh S, Hassan A E, Matsumoto K. An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Softw. Eng.*, 2017, 43(1): 1-18.

[62] Herbold S. Comments on ScottKnottESD in response to "an empirical comparison of model validation techniques for defect prediction models". *IEEE Trans. Softw. Eng.*, 2017, 43(11): 1091-1094.

Zhou Xu received his B.S. degree in computer science and technology from Huazhong Agricultural University, Wuhan, in 2014. Now he is a joint Ph.D. candidate in the School of Computer Science at Wuhan University, Wuhan, and in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also a temporary research assistant at the College of Computer Science and Technology, Harbin Engineering University, Harbin. His research interests include software defect prediction, feature engineering, and data mining.



Shuai Pang received his B.S. degree in information and computing science from Huazhong Agricultural University, Wuhan, in 2018. Now he is a Master student in the School of Computer Science at Wuhan University, Wuhan. His research interest focuses on software engineering and machine learning.



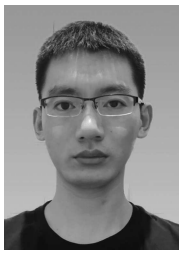
Tao Zhang spent one year as a postdoctoral research fellow in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He got his Ph.D. degree in computer science from the University of Seoul, Seoul, in 2013. Now, he is an associate professor in the College of Computer Science and Technology at Harbin Engineering University, Harbin. His research interest includes mining software repositories and empirical software engineering.

Xia-Pu Luo received his Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2007, and was a Post-Doctoral Research Fellow with the Georgia Institute of Technology, Atlanta. Now, he is an associate professor with the Department of Computing and an associate researcher with the Shenzhen Research Institute, The Hong Kong Polytechnic University, Hong Kong. His current research focuses on smartphone security and privacy, network security and privacy, and Internet measurement.

Jin Liu received his Ph.D. degree in computer science from the State Key Laboratory of Software Engineering, Wuhan University, Wuhan, in 2005. He is currently a professor in the School of Computer Science, Wuhan University, Wuhan. His research interests include software engineering, machine learning, and interactive collaboration on the Web.

Yu-Tian Tang received his Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2018. Currently, he is a post-doctoral research fellow in The Hong Kong Polytechnic University, Hong Kong. His research interests include software product line, system security and machine learning.

Xiao Yu received his B.S. degree in computer science and technology from Hubei University, Wuhan, in 2015. Now he is a joint Ph.D. candidate in the School of Computer Science at Wuhan University, Wuhan, and in the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include software engineering and machine learning.



Lei Xue received his Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2017. Now, he is a post-doctoral research fellow with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His current research focuses on network security, mobile security, and network measurement.