IET Research Journals

Research Article

IET Journals
The Institution of Engineering and Technology

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

A joint distribution matching model for distribution-adaptation-based cross-project defect prediction

Shaojian Qiu[1], *Lu Lu*[1,2] ✉, *Siyu Jiang*[3]

[1] *School of Computer Science and Engineering, South China University of Technology, Guangzhou 510000, People's Republic of China*
[2] *Modern Industrial Technology Research Institute, South China University of Technology, Zhongshan 528400, People's Republic of China*
[3] *School of Software Engineering, South China University of Technology, Guangzhou 510000, People's Republic of China*
✉ *E-mail: lul@scut.edu.cn*

Abstract: Using classification methods to predict software defect is receiving a great deal of attention and most of existing studies primarily conduct prediction under the within-project setting. However, there usually has no or very limited labeled data to train an effective prediction model at an early phase of the software lifecycle. Thus, cross-project defect prediction (CPDP) is proposed as an alternative solution, which is learning a defect predictor for a target project by using labeled data from a source project. Differing from previous CPDP methods that mainly apply instances selection and classifiers adjustment to improve the performance, in this paper, we put forward a novel distribution-adaptation-based CPDP approach, joint distribution matching (JDM). Specifically, JDM aims to minimize the joint distribution divergence between the source and target project to improve the CPDP performance. By constructing an adaptive weight vector for the instances of the source project, JDM can be effective and robust at reducing marginal distribution discrepancy and conditional distribution discrepancy simultaneously. Extensive experiments verify that JDM can outperform related distribution-adaptation-based methods on 15 open-source projects that are derived from two types of repositories.

1 Introduction

Predicting the defects of a software project at an early phase of the software lifecycle will effectively help quality-assurance teams search for potential issues and assign test resources [1]. Based on this, many researchers pay close attention to defect-prediction technology and try to detect defect-prone modules or files by machine-learning methods. Most of previous methods [2-6] operate primarily under a within-project setting, meaning that prediction models are trained and applied on files or modules from same software project.

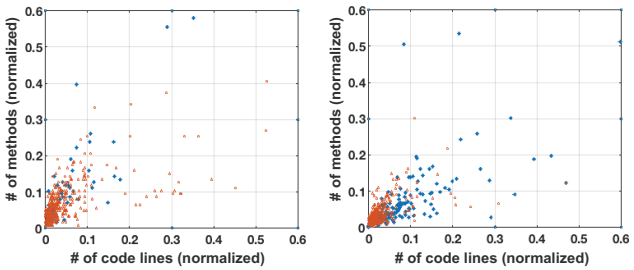
Nevertheless, in real-world scenarios, due to the frequent software iterations and expensive data labeling, there are little to no within-project training data for a new defect prediction task. Fortunately, different companies and organizations share their public data repositories (e.g., AEEEM [7], PROMISE [8]) regarding software defects. By taking advantage of these public data, many cross-project defect-prediction (CPDP) approaches have been proposed. These CPDP methods can be categorized into two subsettings [9]: predicting with only cross-project data [10-15] and predicting with a limited amount of labeled within-project data [16-19].

In this paper, we focus on the former (CPDP with only cross-project data) which is more challenging than latter because it need to handle the cold-start case. The cold-start case is a well-known issue for recommender systems [20] and it also exists in the early stage of a project that have no defect-labeled data. To verify the feasibility of the CPDP with only cross-project data, He et al. [11] investigate on 34 datasets and propose a method that select suitable training data to build a prediction model automatically. A series of experimental results shows that the performance of the improved CPDP method is comparable to within-project defect prediction approaches. In recent years, various researchers have focused on improving the performance of CPDP. Turhan et al. [10] assume that dissimilar data between source and target project easily bring negative effects, so they apply a nearest neighbor filter (Filter) that selects analogous data to fine-tune CPDP models. However, the discarded data may contain useful information for model training. In the CPDP area, the data gravitation (DG) method [21] is usually used to re-weight source instances to weaken the impact of irrelevant source data [12][14]. Ma et al. [12] propose a transfer Naive Bayes (TNB) model that first re-weights source instances via DG, then trains a Naive Bayes model on these reweighted data. Using the same DG process, Ryu et al. [14] also utilize the re-weighted source instances to build a value-cognitive boosting model with a support vector machine (VCB-SVM).

These CPDP approaches mentioned above focus primarily on instances selection and classifiers adjustment. In fact, reducing distribution discrepancy between source and target projects is also an effective way to improve the performance of CPDP. We call this type of approach distribution-adaptation-based CPDP method. To explore the effectiveness of distribution-adaptation-based CPDP method, Nam et al. [13] propose an approach called TCA+ which extends a transfer learning approach, transfer component analysis (TCA) [22], with some customized normalizing rules. Jing et al. [15] also apply the semi-supervised TCA (SSTCA) method when they conduct the CPDP task. Specifically, TCA aims to find a features transformation that reduces marginal probability distribution discrepancy to facilitate CPDP. However, in practice, just considering the discrepancy of the marginal probability distribution is not enough. Instead, it is necessary to consider the challenging scenario in which the source and target projects are different in both marginal and conditional probability distributions [23].

Let us use a toy example to demonstrate this scenario. As Figure 1 shows, we display the data of two real-world projects, Eclipse JDT Core (JDT) and Apache Lucene (LC). For effective demonstration and comparison, two dimension features, 'number of code lines' and 'number of methods', are chosen to plot on the X axis and Y axis, respectively. We normalize these data by Min-Max Scaling and only show the main distribution in coordinates. The blue (red) points indicate the defective (clean) instances of projects. In this figure, the discrepancies of marginal probability distribution and conditional probability distribution are simultaneously existed between two projects ($Pr(X_s) \neq Pr(X_t)$ and $Pr(Y_s|X_s) \neq Pr(Y_t|X_t)$).

To address this challenging scenario, we put forward a novel CPDP approach named joint distribution matching (JDM) that aims



(a) Source Project (JDT) (b) Target Project (LC)

Fig. 1: In our problem, both marginal distribution discrepancy and conditional distribution discrepancy are existed between the source and target projects.

to adapt the marginal and conditional distributions jointly in a reproducing-kernel Hilbert space (RHKS), then use a nonparametric transfer learning method named kernel mean matching (KMM) [24] to construct an instance-weights vector for the source project. By integrating these two steps, JDM can simultaneously reduce the divergences in both the marginal distribution and conditional distribution between the source and target projects.

In this paper, our major contributions are:

- Unlike prior CPDP methods, which focus on filtering the instances or adjusting the classifiers, this paper focuses on the up-to-date distribution-adaptation-based approach to improve CPDP performance. Specifically, this paper proposes that, in CPDP task, it is necessary to consider the scenario in which the source and target projects are different in both marginal and conditional probability distributions.

- This paper proposes a novel CPDP approach called JDM. It constructs the measurement for the divergences in both the marginal and conditional distributions between source and target projects. Further, JDM adopts an effective optimization method to reduce the divergences and improve CPDP performance.

- To validate the effectiveness of our proposed JDM, we compared it with seven distribution-adaptation-based CPDP methods by employing two base classifiers (logistic regression, LR and random forest, RF) and 15 real-world projects (five projects derived from the AEEEM repository and 10 derived from the PROMISE repository). The experimental results, based on the Scott-Knott Effect Size Difference (ESD) test, indicate that JDM can perform better than other methods examined.

The rest of this paper is organized as follows. In Section 2, we review relevant works. In Section 3, we illuminate our JDM approach in detail. In Section 4, we provide our experimental setting, design and results to validate the effectiveness of our approach. Finally, we discuss additional settings of JDM and threats to validity in Section 5. Section 6 concludes our work and outlook the future work.

2 Related Work

CPDP methods are used when a project (target project) has no or only very limited training data for a supervised defect-prediction task, but a similar project (source project) has sufficient training data. In previous researches, CPDP methods are categorized into two main types: predicting only using the data in source project [10-15, 25] and predicting with a small ratio of labeled data in target project [16-19]. In this paper, we focus on the scenario in which the target project has no labeled data. In this section, we briefly review the existing CPDP approaches without labeled target-project data.

In early studies of CPDP, Zimmermann et al. [25] apply 12 real-world software projects and 622 combinations to analyze the performance of the cross-project model and find that only 3.4% of results are acceptable. This experimental result means that CPDP is still a challenging task. In a more empirical investigation, He et al. [11] put forward a method that automatically selects suitable source data to build a prediction model for the target project. After conducting a series of experiments on 34 datasets obtained from 10 open-source projects, the results show that the performance of improved CPDP methods are comparable to within-project defect-prediction approaches.

More recently, various researchers have focused on how to improve the CPDP performance. Turhan et al. [10] analyze the applicability of cross-project data and design a nearest neighbor filter that selects analogy data to fine-tune CPDP models. Following instance-based CPDP principles, Ma et al. [12] develop a TNB model, that uses the DG method to re-weight source instances and train the Naive Bayes model on these weighted data. The experimental results indicate that TNB is more accurate in terms of area under curve (AUC). Using the same data DG method to re-weight source instances, Ryu et al. [14] propose an approach named VCB-SVM that uses a value-cognitive boosting model with a SVM to address the class imbalance issue in the CPDP application. However, TNB or VCB-SVM is designed only for Naive Bayes [26] or SVM [27]. Unlike the Filter, TNB, and VCB-SVM which select or reweight source data, Nam et al. [13] propose a feature representation method named TCA+. TCA+ extends a transfer-learning approach TCA [22] with customized normalizing rules to transfer defect knowledge across projects. The experimental results for eight open-source projects show that TCA+ can improve the performance of CPDP. Jing et al. [15] apply the SSTCA [22] method when they conduct the CPDP task. However, TCA+ and SSTCA ignore to consider the discrepancy of the conditional probability distribution across projects.

Besides, some studies on distribution adaptation [28] are also associated with this paper. Huang el al. [24] put forward a nonparametric method, KMM, which generates resampling weights for training data without distribution estimation. The method works by adapting the marginal distributions between training and testing data in a RHKS. In our paper, we extend the KMM method to match the joint probability distribution between source and target projects. Long el al. [23] propose a distribution-adaptation-based approach, referred to as joint distribution adaptation (JDA), to adapt both the marginal distribution and conditional distribution of training and testing data. In our paper, we apply the measuring approach in JDA to calculate the discrepancy of joint distribution and adjust the distribution-adaptation procedure.

3 Joint Adaptation Matching Methodology

In this section, we present the JDM approach in detail. The frequently used notations are shown in Table 1.

3.1 *Problem Definition*

Given a labeled source project P_s and unlabeled target project P_t under the assumption that their feature spaces $\mathcal{X}_s = \mathcal{X}_t$ and binary label sets $\mathcal{Y}_s = \mathcal{Y}_t = \{clean, defective\}$. Let $\mathbf{x_s} =$

Table 1 Summary of frequently used mathematical notations

Notations	Mathematical Meanings
P_s, P_t	The source/target project
m, n	Number of source/target project instances
d	Number of feature space dimension
M_s, M_t	The marginal distribution of source/target project
C_s, C_t	The conditional distribution of source/target project
J_s, J_t	The joint distribution of source/target project
MMD$(\cdot, \cdot)$	Maximum Mean Discrepancy between distributions
$\boldsymbol{\alpha}$	The weight vector for the instances of source project

$\{x_1, \ldots, x_m\} \in \mathbb{R}^{m*d}$ denote the instances of source project and $\mathbf{x_t} = \{x_{m+1}, \ldots, x_{m+n}\} \in \mathbb{R}^{n*d}$ denote the instances of target project. y_s and y_t respectively represent the labels for $\mathbf{x_s}$ and $\mathbf{x_t}$. In general, if two projects P_s and P_t are different, they may have a different marginal distribution $M_s(\mathbf{x_s}) \neq M_t(\mathbf{x_t})$ and different conditional distribution $C_s(y_s|\mathbf{x_s}) \neq C_t(y_t|\mathbf{x_t})$. In this paper, referring to [23], we use joint distribution to represent the composition of the marginal and conditional distributions.

To address this challenging scenario, we design the JDM approach to minimize the divergences between 1) $M_s(\mathbf{x_s})$ and $M_t(\mathbf{x_t})$; 2) $C_s(y_s|\mathbf{x_s})$ and $C_t(y_t|\mathbf{x_t})$. There are two mainly steps in our proposed approach. The first step is to construct a divergence measurement that considers the marginal and conditional distributions in the meantime. For minimizing the divergence, the second step is to provide a method of estimating instance weights to match the joint distribution of P_s and P_t.

3.2 Constructing the Divergence Measurement

In the above problem setting, $\mathbf{x_s}$ and $\mathbf{x_t}$ are drawn from two projects whose distributions are different. Many criteria can be used to estimate distance of distributions (e.g., Kullback Leibler divergence (KL) [29] and Maximum Mean Discrepancy (MMD) [30]). To avoid building the parametric estimators and estimating intermediate density, we choose the nonparametric method, MMD, to estimate the distribution distance.

According to [30], the empirical estimate of MMD between $M_s(\mathbf{x_s})$ and $M_t(\mathbf{x_t})$ is:

$$\text{MMD}(M_s, M_t) = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) - \frac{1}{n} \sum_{j=m+1}^{m+n} \phi(x_j) \right\|_{\mathcal{H}}^2 \quad (1)$$

where $\|.\|_{\mathcal{H}}^2$ is the RKHS norm and $\phi(\cdot)$ represents a feature mapping on the RKHS.

When calculating the MMD between $C_s(y_s|\mathbf{x_s})$ and $C_t(y_t|\mathbf{x_t})$, because the posterior probabilities $C_s(y_s|\mathbf{x_s})$ and $C_t(y_t|\mathbf{x_t})$ are quite involved, we turn to utilize the sufficient statistics of class-conditional distributions $C_s(\mathbf{x_s}|y_s)$ and $C_t(\mathbf{x_t}|y_t)$ instead. Besides, in the our CPDP task, there are no labeled data in the target project, so $C_t(\mathbf{x_t}|y_t)$ cannot be directly calculated. To address this issue, we apply the predictor, trained by the labeled source data, to predict the pseudo-labels for unlabeled target data. Though some pseudo-labels may be incorrect due to the distribution discrepancy, we assume that the pseudo class centroids are not far from the true class centroids [23]. In our JDM, we select the combination of distribution-adaptation method (e.g. TCA or KMM) and a base classifier (e.g., LR or RF) as the initial pseudo-label predictor. For further study, we show some experimental results to illuminate the influence of different initial pseudo-labels predictors in the discussion section of this paper.

Combining the adjustment of posterior probabilities and the operation of pseudo-labels mentioned above, the conditional distribution discrepancy for clean and defective classes can be written as (2). In this paper, we mark clean as 0 and defective as 1, the label $l \in \{clean, defective\} = \{0, 1\}$.

$$\text{MMD}(C_s^l, C_t^l) = \left\| \frac{1}{m_l} \sum_{x_i \in P_s^l} \phi(x_i) - \frac{1}{n_l} \sum_{x_j \in P_t^l} \phi(x_j) \right\|_{\mathcal{H}}^2 \quad (2)$$

where $P_s^l = \{x_i : x_i \in P_s \wedge y(x_i) = l\}$ and $P_t^l = \{x_j : x_j \in P_t \wedge y(x_j) = l\}$, m_l and n_l respectively represent the number of instances in P_s^l and P_t^l.

To calculate the divergence between the joint distributions of source and target projects, we apply the measurement function in [23] that simultaneously considers the MMDs of marginal and conditional distributions. The divergence can be formulated as:

$$D(J_s, J_t) = \text{MMD}(M_s, M_t) + \sum_{l=0}^{\{0,1\}} \text{MMD}(C_s^l, C_t^l) \quad (3)$$

Incorporating equations (1), (2), and (3) leads to the divergence on joint distribution, computed as:

$$\begin{aligned} D(J_s, J_t) = & \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) - \frac{1}{n} \sum_{j=m+1}^{m+n} \phi(x_j) \right\|_{\mathcal{H}}^2 \\ & + \sum_{l=0}^{\{0,1\}} \left\| \frac{1}{m_l} \sum_{x_i \in P_s^l} \phi(x_i) - \frac{1}{n_l} \sum_{x_j \in P_s^l} \phi(x_j) \right\|_{\mathcal{H}}^2 \end{aligned} \quad (4)$$

3.3 Matching on Joint Distribution

In the distribution adaptation area, two types of methods are commonly used to reduce the above joint distribution divergence: minimizing with feature representation (e.g., TCA [22], SSTCA [22] and JDA [23]) and minimizing with instance reweighting (e.g., KMM [24] and KILEP [31]). Through experimental comparison, we find that the feature transformation functions, JDA, are difficult to converge and the performance is not stable in our CPDP datasets. Considering the alternative instance-based function, we adopt KMM algorithm to resolve the minimizing task.

The KMM algorithm, which is a nonparametric method, directly infers weights for the source project without a distribution estimation [24]. This algorithm utilizes both source and target data to build a transferring bridge. Here, KMM also introduces MMD mentioned in the second part, which is used to measure the similarity of the two distributions. The main purpose of KMM is to calculate appropriate resampling weight for each instance of source project to minimize the MMD in an RKHS.

Using the matching operation for marginal distribution as the example, after space mapping and instance weighting, the MMD for the marginal distribution of source and target projects can be formulated as:

$$\begin{aligned} \text{MMD}(M_s, M_t) &= \left\| \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\alpha} \phi(x_i) - \frac{1}{n} \sum_{j=m+1}^{m+n} \phi(x_j) \right\|_{\mathcal{H}}^2 \\ &= \frac{1}{m^2} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} - \frac{2}{m^2} \kappa^\top \boldsymbol{\alpha} + \text{const.} \end{aligned} \quad (5)$$

where $\boldsymbol{\alpha}$ is a m dimensional weight vector that we want to learn for the data of the source project, $K_{ij} := k(x_i, x_j), K \in \mathbb{R}^{m*m}$ and $\kappa_i = \frac{n}{m} \Sigma_{j=m+1}^{m+n} k(x_i, x_j)$. In our paper, we use the gaussian kernel function: $k_{ij} = exp(-\|x_i - x_j\|^2 / 2\sigma)$, and σ is the parameter of function width.

In the same way, we resolve the (4) with the above KMM resolution. The formulation can be rewritten as:

$$\begin{aligned} D(J_s, J_t) = & (\frac{1}{m^2} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} - \frac{2}{m^2} \kappa^\top \boldsymbol{\alpha}) \\ & + \sum_{l=0}^{\{0,1\}} (\frac{1}{m_l^2} \boldsymbol{\alpha}_l^\top K_l \boldsymbol{\alpha}_l - \frac{2}{m_l^2} \kappa_l^\top \boldsymbol{\alpha}_l) + \text{const.} \end{aligned} \quad (6)$$

where $\boldsymbol{\alpha}_l$ is a weight vector for the instances whose label is l, $(K_l)_{ij} := k(x_i^l, x_j^l)$ and $(\kappa_l)_{ij} = \frac{n_l}{m_l} \sum_{x_i \in P_s^l, x_j \in P_t^l} k(x_i, x_j)$.

Now we use all necessary ingredients to construct an optimization problem to find a suitable $\boldsymbol{\alpha}$ to ensure the data distribution of

reweighted source instances is close to the distribution of the target project. In reference to [24], we add three constraints to the formulation: the lower bound (LB) of elements in $\boldsymbol{\alpha}$, the upper bound (UB) of elements in $\boldsymbol{\alpha}$, and $|1/m\Sigma_{i=1}^{m}\alpha_i - 1| \leq \epsilon$. A good choice of ϵ should be $O(\mathrm{UB}/\sqrt{m})$. The constrained optimization problem can be formulated as:

$$\begin{aligned} \min D(J_s, J_t) = \min_{\boldsymbol{\alpha}} [(\frac{1}{2}\boldsymbol{\alpha}^\top K\boldsymbol{\alpha} - \kappa^\top \boldsymbol{\alpha}) \\ + \sum_{l=0}^{\{0,1\}} (\frac{1}{2}\boldsymbol{\alpha}_l^\top K_l \boldsymbol{\alpha}_l - \kappa_l^\top \boldsymbol{\alpha}_l)] \qquad (7) \\ s.t. \qquad |1/m\sum_{i=1}^{m}\alpha_i - 1| \leq \epsilon, LB \leq \alpha_i \leq UB \end{aligned}$$

The formulation (7) is a standard constrained optimization problem which could be solved by many existing solvers and tools. In this paper, we use the CVX toolbox (http://cvxr.com/cvx/) to tackle the minimization process.

Algorithm 1 JDM: Joint Distribution Matching

Input:
P_s: Labeled source project
P_t: Unlabeled target project
$maxIter$: Maximum number of iterations
σ: Kernel parameter
Output:
Adaptive classifier f

1: Use Z-score method to normalize the instances of P_s and P_t, respectively.
2: Initialize pseudo-label vector $pseudoY_0$ for instances in P_t by the combination of distribution-adaptation method (e.g. TCA or KMM) and a base classifier (e.g., LR or RF).
3: **for** $t = 1 \ldots maxIter$ **do**
4: Compute weight vector $\boldsymbol{\alpha_t}$ by solving (7) with σ
5: Normalize $\boldsymbol{\alpha_t}$ by Min-Max Scaling
6: Learn an adaptive classifier f_t on the $\boldsymbol{\alpha_t}$ weighted source data by the selected base classifier
7: Predict the new label vector $pseudoY_i$
8: **if** $pseudoY_t$ is identical with $pseudoY_{t-1}$ **then**
9: Break loop
10: **end if**
11: **end for**
12: Return the last adaptive classifier f_t

Importantly, we generally can obtain more accurate pseudo-labels with JDM. Thus, in most cases, if we alternate using the new pseudo-labels to run JDM, we can enhance the prediction quality until convergence (all pseudo-labels are not changes in the next iteration). If researchers use some base classifiers involving randomness (e.g., RF), then JDM cannot converge quickly or may not converge. When using such base classifiers, we can relax the condition of the end of the iteration (e.g., 98% of pseudo-labels are not changes in the next iteration). This pseudo-label refinement procedure [23] is empirically effective in our CPDP datasets, whose details the discussion section presents. Algorithm 1 presents the pseudo-code of our JDM approach to perform CPDP.

4 Experiments

In this section, our JDM approach is implemented to perform CPDP empirically. We first introduce the experimental datasets and evaluation metrics. Then, to investigate the performance of the JDM approach, we design some comparative experiments and present the results.

Table 2 The 5 datasets selected from the AEEEM repository, sorted in order of the name. (Defect Rate: the percentage of defective instances.)

Project Name	Time Period	Instance Count	Defect Rate
Equinox (EQ)	2005.1-2008.6	324	39.8%
Eclipse JDT (JDT)	2005.1-2008.6	997	20.7%
Apache Lucene (LC)	2005.1-2008.10	691	9.3%
Mylyn (ML)	2005.1-2009.3	1862	13.2%
Eclipse PDE UI (PDE)	2005.1-2008.9	1497	14%

Table 3 The 10 datasets selected from the PROMISE repository

Project Name	Project Version	Instance Count	Defect Rate
ant	1.7	745	22.3%
arc	1.0	234	11.5%
camel	1.6	965	19.5%
elearn	1.0	64	7.8%
ivy	2.0	352	11.4%
prop	6.0	660	10%
synapse	1.2	256	33.6%
systemdata	1.0	65	13.9%
tomcat	6.0	858	9.0 %
velocity	1.6	229	34.1 %

4.1 Datasets

To assess the JDM approach, we use two existing defect-prediction repositories (AEEEM [7] and PROMISE [8]). Both are widely-utilized in recent CPDP research [13, 16, 32, 33]. In our experiment, five projects are selected from the AEEEM repository (including 20 one-one CPDP combinations) and 10 open-source projects are selected from the PROMISE repository (including 90 one-one CPDP combinations). Of the five AEEEM projects, each project consists of files that includes 61 comprehensive attributes and a label (defective or clean). Of the 10 PROMISE projects, each project consists of a collection of Java class that includes 20 static code attributes and a label (defective or clean).

Table 2 and Table 3 respectively present the essential information of the two repositories including project name, time period (project versions), number of instances, and percent of defects. To verify the generality of our approach, the repositories are composed of several projects with different sizes and defective rates. Table 4 and Table 5 respectively show the attributes of the AEEEM and PROMISE repositories. Because AEEEM includes 61 attribute, the Table 4 only show some representative attributes. For a detailed description of all the AEEEM attributes, please refer to [7].

4.2 Evaluation Metrics

To measure prediction performance, we use F-measure [34] and AUC [35], which are frequently used in recent CPDP researches [9, 12-16], as the evaluation metrics. These evaluation metrics are defined as follows.

In a classification task, an instance may have four possible outcomes: classifying a truly defective instance into defective (true positive, TP); classifying a clean instance into defective (false positive, FP); classifying a truly defective instance into clean (false negative, FN); and classifying a truly clean instance into clean (true negative, TN). Based on these four outcomes, the F-measure, AUC, and their related metrics are defined below:

Recall, also called probability of detection (pd), is a measure of completeness and defines the probabilities of true defective instances compared with the number of all defective instances:

$$recall = pd = \frac{TP}{TP + FN}$$

Table 4 List of attributes selected from the AEEEM for presentation. The detail descriptions of attributes are referred to [13].

Attribute	Description
ck oo cbo	Coupling between objects
ck oo numberOfLinesOfCode	Number of lines of code
numberOfBugsFoundUntil	Number of all bugs
numberOfCriticalBugsFoundUntil	Number of bugs whose severity is critical and blocker
CvsEntropy	Entropy of code changes
CvsLogEntropy	Logarithmically decayed CvsEntropy
LDHHcbo	Linearly decayed entropy of ck oo cbo
LDHH numberOfLinesOfCode	Linearly decayed entropy of ck oo numberOfLinesOfCode
WCHU cbo	Weighted churn of ck oo cbo
WCHU numberOfLinesOfCode	Weighted churn of ck oo numberOfLinesOfCode

Table 5 List of PROMISE attributes. The descriptions are referred to [16].

Attribute	Description
dit	The maximum distance from a given class to the root of an inheritance tree
noc	# of children of a given class in an inheritance tree
cbo	# of classes that are coupled to a given class
rfc	# of distinct methods invoked by code in a given class
lcom	# of method pairs in a class that do not share access to any class attributes
lcom3	Another type of lcom metric proposed by Henderson-Sellers
npm	# of public methods in a given class
loc	# of lines of code in a given class
dam	The ratio of the # of private/protected attributes to the total # of attributes in a given class
moa	# of attributes in a given class which are of user-defined types
mfa	# of methods inherited by a given class divided by the total # of methods that can be accessed by the member methods of the given class
cam	The ratio of the sum of the # of different parameter types of every method in a given class to the product of the # of methods in the given class and the # of different method parameter types in the whole class
ic	# of parent classes that a given class is coupled to
cbm	Total # of new or overwritten methods that all inherited methods in a given class are coupled to
amc	The average size of methods in a given class
ca	Afferent coupling, which measures the # of classes that depends upon a given class
ce	Efferent coupling, which measures the # of classes that a given class depends upon
max_cc	The maximum McCabe's cyclomatic complexity (CC) score of methods in a given class
avg_cc	The arithmetic mean of the McCabe's cyclomatic complexity (CC) scores of methods in a given class

Precision is a measure of exactness and defines the probabilities of true defective instances to the number of instances predicted as defective:

$$precision = \frac{TP}{TP + FP}$$

In fact, it is difficult to compare the performance of prediction models by using only precision or recall because there is a trade-off between them. To avoid this issue, we choose **F-measure**, which is a dual assessment of both precision and recall, to evaluate the various approaches in our experiment.

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Probability of False alarm (pf), also called false positive rate, represents the probabilities of false positive instances compared with the number of all clean instances:

$$pf = \frac{FP}{FP + TN}$$

A receiver-operating characteristics (ROC) graph, is a two-dimensional graph in which pd is plotted on the y-axis and pf is plotted on the x-axis, can be used to show the relative trade-off between benefit and cost. In this paper, we also use a single scalar value, **the area under the ROC curve (AUC)**, to compare the performances of the various prediction models. According to the meaning of AUC [35], it is in positive correlation with pd, while in negative correlation with pf. If AUC less than 0.5, it indicates a prediction model with a low pd and a high pf. In CPDP area, we seek the model with a high pd and a low pf. In other words, a higher AUC is better for our prediction task [12].

4.3 Experiment Design

In this paper, we compare the JDM approach with seven related distribution-adaptation and CPDP methods on above 15 open-source projects (five projects are derived from the AEEEM repository, and 10 projects are derived from the PROMISE repository).

First, our algorithm adopts TCA [22] or KMM [24] approaches when build the initial predictor, so we compare JDM with TCA and KMM to observe the improvement with JDM process. Second, because SSTCA [22] is applied in most up-to-date domain-adaptation CPDP method [15], we compare JDM with SSTCA algorithm. Third, considering that JDM is combined with JDA [23] and KMM, we also compare JDM with JDA, which are a renowned distribution-adaptation algorithm who handles the distribution difference. Fourth, because we utilize the instance-based method to handle the CPDP, we also compare two classic instance-based CPDP methods, DG [21] and Filter [10]. Moreover, to make a fair comparison, we respectively use LR and RF as the base classifiers for all the above algorithms, so we also include LR and RF as the baselines for comparisons. Considering that RF involves a degree of randomness, when the base classifier is RF, we ran the model 10 times each and took the mean as the prediction performance.

In the software defect prediction task, the parameter tuning of the model has received attention in recent years [36-38]. Good parameter selection will greatly improve model performance. Therefore, in the experiments this paper describes, we have tuned the parameters for each method and extracted the combination of parameters for the first-ranking prediction result in the Scott-Knott ESD test [39]. Table 6 lists the tuned parameters in our experiment, and Table 7 presents the final parameter selection for each method.

In our experimental comparison, because some datasets tend to produce over- or under-performing models, we also applied the Scott-Knott ESD test to compare the performance of the methods we examined. The Scott-Knott ESD test used herein is a mean comparison method that uses hierarchical clustering to divide a set of measurements (e.g., F-measure and AUC) into statistically different groups with non-negligible differences. The mechanism of the Scott-Knott ESD test consists of two steps: (1) finding a partition that maximizes the measurements means between groups, and (2) either dividing into two groups or combining into one group. For a detailed process description of the Scott-Knott ESD test, please refer to [39]. The implementation of the Scott-Knott ESD test can be obtained in the *sk _esd* function of the *ScottKnottESD* R package.

Table 6 List of tuned parameters in our experiments

Model Name	Parameter	Notation	Tuning Range	Description
JDM & KMM	sigma	σ	[0.01, 0.1, 1, 10, 100]	The function width of gaussian kernel.
TCA & SSTCA & JDA	dim	d	[10, 15, 20, 25, 30]	Output dimension. No less than zero, integer.
	mu	μ	[0.5, 1 ,2 ,3, 4]	The parameter option provided by the literature [22]. No less than zero.
	lambda	λ	[0, 0.5, 1 ,2 ,3]	The parameter option provided by the literature [22]. No less than zero.
Filter	k	k	[10, 15, 20, 25, 30]	k nearest neighbors parameter for the selection of source project data
RF	n_trees	n_t	[30, 50, 100, 150]	The number of trees in the forest.
DG	This study uses untuned DG.			
LR	For LR, we use the same implementation (i.e., LIBLINEAR), and parameters settings as those used by Nam et al. [13] "-S 0" for using LR and "-B -1" for using no bias term.			

Table 7 Parameter selection for each method in different repositories

Base Classifier	Using LR as Classifier		Using RF as Classifier	
Repository	**AEEEM**	**PROMISE**	**AEEEM**	**PROMISE**
JDM	[σ] = [0.1]	[σ] = [0.1]	[n_t, σ] = [100, 1]	[n_t, σ] = [100, 0.1]
KMM	[σ] = [0.01]	[σ] = [0.1]	[n_t, σ] = [100, 1]	[n_t, σ] = [100, 0.1]
TCA	[d, μ, λ] = [10, 1, 1]	[d, μ, λ] = [10, 1, 0.5]	[n_t, d, μ, λ] = [50. 30, 0.5, 1]	[n_t, d, μ, λ] = [100. 10, 2, 2]
SSTCA	[d, μ, λ] = [10, 2, 1]	[d, μ, λ] = [10, 2, 1]	[n_t, d, μ, λ] = [100. 30, 2, 1]	[n_t, d, μ, λ] = [100. 20, 2, 1]
JDA	[d, μ, λ] = [10, 1, 0.5]	[d, μ, λ] = [10, 1, 0.5]	[n_t, d, μ, λ] = [100. 20, 1, 0.5]	[n_t, d, μ, λ] = [100. 30, 1, 2]
Filter	[k] = [15]	[k] = [20]	[n_trees, k] = [150, 30]	[n_trees, k] = [100, 15]
DG	Untuned	Untuned	[n_trees] = 150	[n_trees] = 150
Classifier Only	Untuned	Untuned	[n_trees] = 100	[n_trees] = 100

4.4 Experiment Result

Figures 2 (LR classifier as the base classifier) and Figure 3 (RF classifier as the base classifier) summarize the Scott-Knott ESD test comparison results. We then use boxplots to present the distribution of the performance difference for each of the models examined.

4.4.1 ***Performance Comparison when using LR classifier as the base classifier:*** Through the experiments on the AEEEM and PROMISE repositories with RF, Figure 2 presents the comparative results of the Scott-Knott ESD test. Our JDM approach prevails statistically in performance comparisons of F-measure and AUC with the other seven methods examined.

In detail, regarding the AEEEM repository: **1)** The average F-measure of JDM is 0.437, which respectively outperforms KMM (0.419), SSTCA (0.417), TCA (0.406), JDA (0.378), Filter (0.368), LR (0.358) and DG (0.327) by 4.3%, 4.7%, 7.6%, 15.4%, 18.8%, 22.1%, and 33.7%. **2)** The average AUC of JDM is 0.67, which outperforms SSTCA (0.658), KMM (0.653), TCA (0.648), JDA (0.641), Filter (0.608), LR (0.601) and DG (0.55) by 1.8%, 2.6%, 3.4%, 4.5%, 10.3%, 11.5%, and 21.7%, respectively.

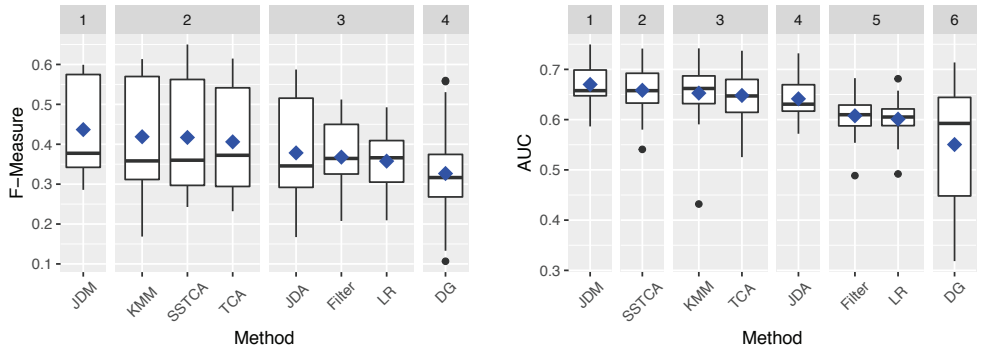
Regarding the PROMISE repository: **1)** The average F-measure of JDM is 0.380, which respectively outperforms TCA (0.369), KMM (0.365), SSTCA (0.362), Filter (0.347), LR (0.346), JDA (0.333) and DG (0.289) by 3.1%, 4.2%, 5.1%, 9.4%, 10.0%, 14.1%, and 31.6%. **2)** The average AUC of JDM is 0.682, which outperforms TCA (0.667), KMM (0.662), SSTCA (0.66), Filter (AUC: 0.641), LR (0.639), JDA (AUC: 0.622) and DG (AUC: 0.562) by 2.2%, 2.9%, 3.3%, 6.3%, 6.7%, 9.6%, and 21.2%, respectively.

4.4.2 ***Performance Comparison when using RF classifier as the base classifier:*** Using RF as the base classifier, Figure 3 shows the comparative results of eight approaches for 20 combinations for five AEEEM projects and 90 combinations for 10 PROMISE projects.

In the AEEEM repository, our approach JDM can achieve minor improvement over other seven examined methods in most combinations. **1)** The average F-measure of JDM is 0.428, which respectively outperforms KMM (0.418), SSTCA (0.417), DG (0.416), RF (0.415), Filter (0.415), TCA (0.414) and JDA (0.396) by 2.5%, 2.8%, 3.0%, 3.1%, 3.1%, 3.5%, and 8.2%. **2)** The average AUC of JDM is 0.663, which outperforms DG (0.652), KMM (0.650), Filter (0.650), RF (0.649), SSTCA (0.644), TCA (0.644) and JDA (0.635) by 1.7%, 1.9%, 2.0%, 2.2%, 2.9%, 2.9%, and 4.4%, respectively.

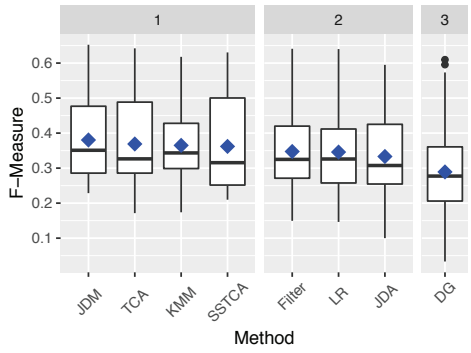
In most combinations of the PROMISE repository, the performance are better by using our approach JDM. **1)** The average F-measure of JDM is 0.377, which respectively outperforms KMM (0.367), DG (0.360), SSTCA (0.357), Filter (0.356), TCA (0.351), RF (0.346) and JDA (0.330) by 2.6%, 4.8%, 5.7%, 5.8%, 7.5%, 9.0%, and 14.2%. **2)** The average AUC of JDM is 0.671, which outperforms KMM (0.664), DG (0.662), Filter (0.656), SSTCA (AUC: 0.652), RF (0.650), TCA (AUC: 0.648) and JDA (AUC: 0.638) by 1.0%, 1.3%, 2.4%, 3.0%, 3.3%, 3.5%, and 5.2%, respectively.

In summary, whether using LR or RF as the base classifier, our approach JDM tends to yield better CPDP performance than the other seven CPDP techniques we examined. This result suggests that reducing the divergences in both the marginal and conditional distributions between source and target projects can usually improve the performance of CPDP. We believe that this improvement would provide more effective help to quality assurance teams for detecting potential software issues and reasonably assigning test resources at an early phase of the software lifecycle.

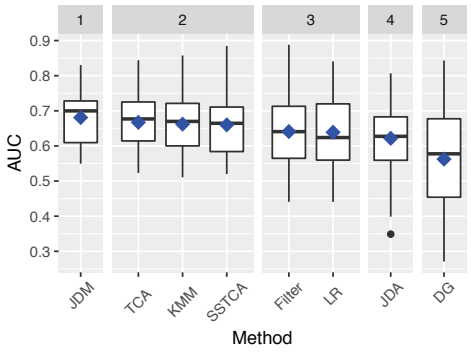


(a) F-measure comparison in the AEEEM repository

(b) AUC comparison in the AEEEM repository

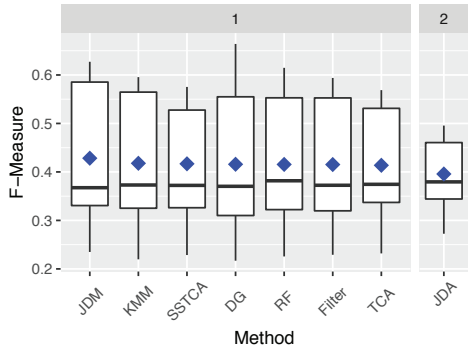


(c) F-measure comparison in the PROMISE repository

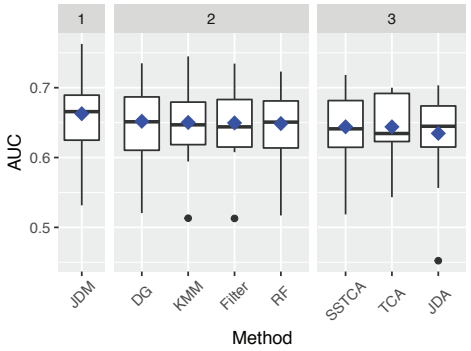


(d) AUC comparison in the PROMISE repository

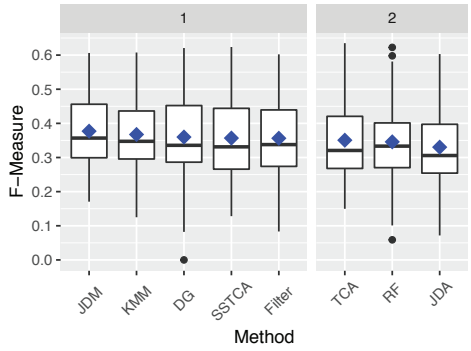
Fig. 2: The ScottKnott ESD ranking of 8 different CPDP approaches (using LR as the base classifier) across the AEEEM and PROMISE repositories. The blue diamond indicates the average metric of our studied method.



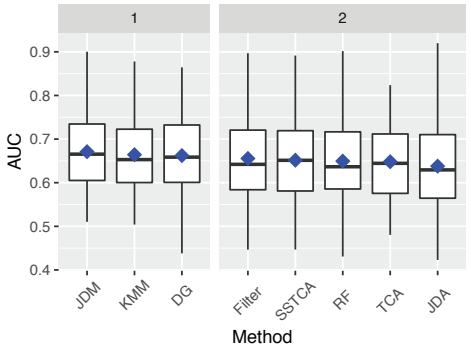
(a) F-measure comparison in the AEEEM repository



(b) AUC comparison in the AEEEM repository



(c) F-measure comparison in the PROMISE repository



(d) AUC comparison in the PROMISE repository

Fig. 3: The ScottKnott ESD ranking of 8 different CPDP approaches (using RF as the base classifier) across the AEEEM and PROMISE repositories. The blue diamond indicates the average metric of our studied method.

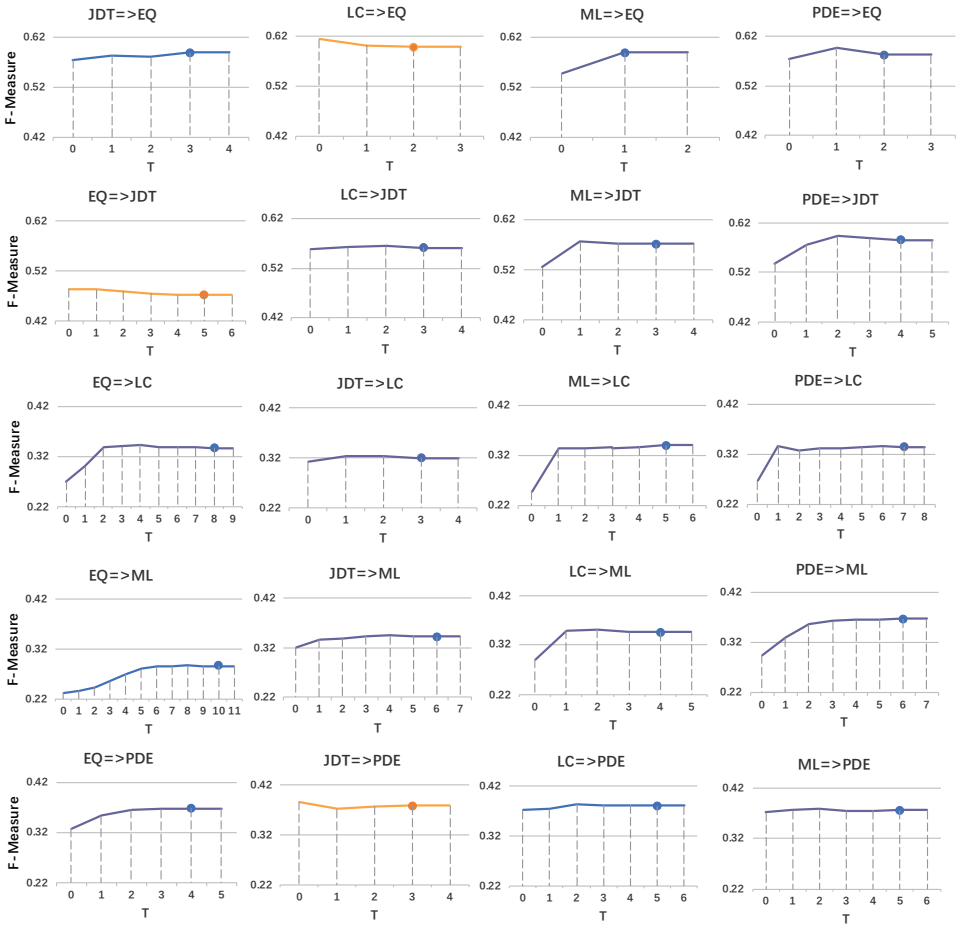


Fig. 4: The convergence process of JDM in the AEEEM repository (20 combinations of source and target projects).

5 Discussion

5.1 *The Convergence Property of JDM*

In this subsection, we investigate the convergence property of JDM. As presented in the Methodology, in most cases, the pseudo-labels are obtained by the initial predictor, then the JDM can be iteratively conducted to improve the labeling quality until convergence (all the predicting labels are not changing in the next iteration).

Figure 4 shows the results of 20 combinations of JDM on the AEEEM repository. In this section, we use LR as the base classifier to illuminate the convergence property of our approach because the randomness of RF will cause floating convergence times. Figure 4 displays the F-measures of JDM in each iteration and the filled circle in each subfigure represents the converging iteration. Note that if the iterative number is 0 (T = 0), it means the algorithm only uses initial predictor as the final classifier without JDM process. As known from the calculation, the min.max.avg. iteration numbers for convergence of 20 combinations are 2/10/5.4.

Figure 4 also presents that 1) most combinations can be converged in 10 iterations; 2) in 17 combinations (all subgraphs with blue breakline), the F-measures improve when JDM is conducted, whether converged or not; and 3) of the 17 improved combinations, eight have better final F-measures under algorithm converging than when T = 1 (e.g., EQ=>LC and PDE=>ML) and that the remaining nine combinations perform similarly.

In summary, JDM can improve the performance of CPDP with only one iteration in most combinations. The pseudo-label refinement procedure mentioned in the Methodology may further enhance CPDP performance in these already-improved models. Since each combination has differences both in instance and label distributions, the convergence times of JDM varies by combination. Note that not all combinations can improve performance through the JDM process. One possible reason for this is that the conditional probability distribution used in JDM is calculated by pseudo-labels rather than real labels, so the calculation may be biased, thus causing a certain degree of predictive performance degradation. Finding out how to reduce the impact of deviations is an important part of our future work.

5.2 *Impact of Different Initial Pseudo-Label Predictor*

In the Methodology, we mentioned that our JDM uses a model combined with a TCA (or KMM) model and a base classifier as the predictor for the initial pseudo-label. In fact, our JDM approach also involves a tunable step regarding the initial pseudo-label predictor. Herein, we investigate how different initial predictors affect the performance of JDM+LR. In our experiments, we select the pseudo-label predictor from (TCA+LR, KMM+LR, DG+LR, Filter+LR, and LR only). We use the parameter selection of each model in Table 7 to train the initial predictor.

With various initial pseudo-label predictors, Table 8 presents the corresponding average F-measure, AUC, and convergence times of JDM with different initial pseudo-label predictors. The experimental results show that different initial predictors bring a slight difference in prediction performance and convergence speed: 1) on the average F-measure, the maximum difference does not exceed 0.07; 2) in terms of average AUC, the biggest difference is only 0.1; and 3) the difference in the average convergences times is no more than 1. When using JDM, if better known classifiers are available (e.g., TCA and KMM with LR), we recommend using them. Better initial pseudo labels could help JDM calculate the conditional distribution more accurately, so it is more likely to achieve faster convergence and better performance.

Table 8 Average F-measure, AUC and convergence times in the AEEEM and PROMISE repositories with different initial pseudo-label predictor

Repositories	Metric	TCA+LR	KMM+LR	DG+LR	Filter+LR	LR only
AEEEM	Average F-Measure	**0.437**	0.435	0.435	0.433	0.436
	Average AUC	**0.67**	0.667	0.668	0.666	**0.67**
	Average convergence times	**5.4**	5.6	5.8	5.5	5.6
PROMISE	Average F-Measure	0.378	**0.38**	0.373	0.377	0.376
	Average AUC	0.676	**0.681**	0.671	0.677	0.675
	Average convergence times	5.3	**4.8**	6.8	5.3	5.3

5.3 Threats to Validity

Experimental results might not be generalizable. In this paper, our experiments are conducted on 15 real-world projects included in the AEEEM and PROMISE repositories. It cannot make sure that the JDM and corresponding setting could bring similar improvement for other open-source and closed-source projects. More experimental validation should be performed in the future.

F-measure and AUC might not be the only appropriate measures. In our work, we use F-measure and AUC to evaluate predictive power of model. There are more metrics (e.g., accuracy, recall and G-measure) in the machine-learning problem that could be used for performance evaluation. In fact, F-measure and AUC, as comprehensive measures, are commonly applied to evaluate CPDP models [12-14, 40] .

Parameter selection does not take all options into account. In our experiment design, we have tuned the parameters for each method. Because assessing all possible combinations in the parameter spaces is impractical, we have evaluated several combinations of parameters within a certain range based on experience. There may be better combinations of parameters that can improve each model's performance.

LR and RF are used as the base classifiers. In our experiments, we employ the LR and RF as the base classifiers to examine the CPDP models. It is possible that LR and RF have better (or worse) performance than other known classification methods. In CPDP area, LR has been applied as a base classifier by many literatures [13,16], and RF has been widely used in recent years and is considered to have excellent performance [9,15].

6 Conclusion

When there is no historical labeled data in a given target project, it is challenging but meaningful to construct a CPDP model by using the labeled data of a source project. Most current studies adopt instances selection and classifiers adjustment methods to perform the CPDP. However, these approaches ignore to consider the joint distribution divergence between source and target projects that may weaken the performance of CPDP.

In this paper, we put forward a novel JDM approach to conduct the distribution-adaptation-based CPDP. JDM aims to jointly reducing the marginal distribution discrepancy and conditional distribution discrepancy across source and target projects. By integrating the JDA and KMM procedures, we achieve a reweighting vector for the instances of the source project to conduct the CPDP task. We compare our method to related seven distribution-adaptation-based CPDP methods. The experiment results show that JDM outperforms these approaches on 15 open-source projects derived from AEEEM and PROMISE repositories.

Several issues remain to be investigated in future work. One example is how to extend our JDM method with state-of-the-art solution [15] for class imbalance problem. In addition, we plan to apply the proposed JDM to solve the defect prediction across multiple projects.

7 Acknowledgment

We thank Qian Sun for providing us the source code of KMM, and Sinno Pan for providing us the source code of TCA. This work is supported in part by the National Nature Science Foundation of China (No. 61370103), Guangdong Province Application Major Fund (2015B010131013), Guangzhou Produce & Research Fund (201802020006) and Zhongshan Produce & Research Fund(2017A1014). The source code of JDM can be downloaded from: https://github.com/kevinqiu1990/JDM.

8 References

[1] Menzies, T., Greenwald, J., Frank, A.: 'Data mining static code attributes to learn defect predictors', IEEE Transactions on Software Engineering, 2007, 33, (1), pp.2-13

[2] Hassan, A. E.: 'Predicting faults using the complexity of code changes'. Proc. Thirty-First Int. Conf. on Software Engineering, Vancouver, Canada, May 2009, pp. 78-88

[3] Menzies, T., Milton, Z., Turhan, B., et al.: 'Defect prediction from static code features: current results, limitations, new approaches', Automated Software Engineering, 2010, 17, (4), pp. 375-407

[4] Lee, T., Nam, J., Han, D. G., et al.: 'Micro interaction metrics for defect prediction'. Proc. Nineteenth ACM SIGSOFT symposium and Thirteenth European Conf. on Foundations of Software Engineering, Szeged, Hungary, September 2011, pp. 311-321

[5] Kamei Y, Shihab E, Adams B., et al.: 'A large-scale empirical study of just-in-time quality assurance', IEEE Transactions on Software Engineering, 2013, 39, (6), pp. 757-773

[6] Laradji I H, Alshayeb M, Ghouti L., et al.: 'Software defect prediction using ensemble learning on selected features', Information and Software Technology, 2015, 58, pp. 388-402

[7] D'Ambros, M., Lanza, M., Robbes, R.: 'An extensive comparison of bug prediction approaches'. Seventh IEEE Working Conf. on Mining Software Repositories, Cape Town, South Africa, May 2010, pp. 31-41

[8 'The Promise Repository of Empirical Software Engineering Data', http://openscience.us/repo. accessed 30 July 2018

[9] Herbold S, Trautsch A, Grabowski J.: 'A Comparative Study to Benchmark Cross-project Defect Prediction Approaches', IEEE Transactions on Software Engineering, 2017, (99), pp. 1-25

[10] Turhan, B., Menzies, T., Bener, A, B., et al.: 'On the relative value of cross-company and within-company data for defect prediction', Empirical Software Engineering, 2009, 14, (5), pp. 540-578

[11] He, Z., Shu, F., Yang, Y., et al.: 'An investigation on the feasibility of cross-project defect prediction', Automated Software Engineering, 2012, 19, (2), pp. 167-199

[12] Ma, Y., Luo, G., Zeng, X., et al.: 'Transfer learning for cross-company software defect prediction', Information and Software Technology, 2012, 54, (3), pp. 248-256

[13] Nam, J., Pan, S. J., Kim, S.: 'Transfer defect learning'. Proc. Thirty-fifth Int. Conf. on Software Engineering, San Francisco, California, May 2013, pp. 382-391

[14] Ryu, D., Choi, O., Baik. J.: 'Value-cognitive boosting with a support vector machine for cross-project defect prediction', Empirical Software Engineering, 2016, 21, (1), pp. 43-71.

[15] Jing X Y, Wu F, Dong X. J.: 'An improved SDA based defect prediction framework for both within-project and cross-project class imbalance problems', IEEE Transactions on Software Engineering, 2017, 43, (4), pp. 321-339.

[16] Xia, X., Lo, D., Pan, S, J., et al.: 'HYDRA: Massively compositional model for cross-project defect prediction', IEEE Transactions on Software Engineering, 2016, 42, (10), pp. 977-998

[17] Chen L, Fang B, Shang Z.., et al.: 'Negative samples reduction in cross-company software defects prediction.', Information and Software Technology, 2015, 62, (1), pp. 67-77

[18] Ryu D, Jang J I, Baik J., et al.: 'A transfer cost-sensitive boosting approach for cross-project defect prediction', Software Quality Journal, 2017, 25, (1), pp. 235-272

[19] Qiu S J, Lu L, Jiang S J ., et al.: 'A multiple components weights model for cross-project defect prediction', IET Software, 2018

[20] Schein A I, Popescul A, Ungar L H., et al.: 'Methods and metrics for cold-start recommendations'. Proc. Twenty-fifth annual international ACM SIGIR conference on Research and development in information retrieval, New York,USA , August 2002, pp. 253-260

[21 Peng, L., Yang, B., Chen, Y., et al.: 'Data gravitation based classification', Information Sciences, 2009, 179, (6), pp. 809-819

[22] Pan, S. J., Tsang, I. W., Kwok, J. T., et al.: 'Domain adaptation via transfer component analysis', IEEE Transactions on Neural Networks, 2011, 22, (2), pp. 199-210

[23] Long M, Wang J, Ding G., et al.: 'Transfer feature learning with joint distribution adaptation'. Proc. Fourteenth IEEE International Conference on Computer Vision, Santiago, Chile, 2014, pp. 2200-2207

[24] Huang, J., Smola, A. J., Gretton, A., et al.: 'Correcting sample selection bias by unlabeled data', Advances in Neural Information Processing Systems, 2007, 19, pp. 601-608

[25] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B., et al.: 'Cross-project defect prediction:a large scale experiment on data vs. domain vs. process'. Proc. Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Software Engineering, Amsterdam, the Netherlands, August, 2009, pp.91-100

[26] Huang, J., Smola, A. J., Gretton, A., et al.: 'Data mining: practical machine learning tools and techniques with java implementations', Acm Sigmod Record, 2005, 31, (1), pp.76-77

[27] Chang C C, Lin C J et al.: 'LIBSVM: A library for support vector machines', ACM, 2011

[28] Pan, S. J., Yang, Q.: 'A survey on transfer learning', IEEE Transactions on Knowledge and Data Engineering, 2010, 22, (10), pp. 1345-1359

[29] Hall P.: 'On kullback-leibler loss and density estimation.', Annals of Statistics, 1987, 15, (4), pp. 1491-1519

[30] Borgwardt, K. M., Gretton, A., Rasch, M. J., et al.: 'Integrating structured biological data by kernel maximum mean discrepancy', Bioinformatics, 2006, 22, (14), pp. e49-e57

[31] Sugiyama, M., Suzuki, T., Nakajima, S., Kashima, H., Bunau, P. V., Kawanabe, M: 'Direct importance estimation for covariate shift adaptation', Annals of the Institute of Statistical Mathematics, 2008, 60, (4), pp. 699-746

[32] Yu, Q., Jiang, S., Qian, J.: 'Which Is More Important for Cross-Project Defect Prediction: Instance or Feature?'. Proc. 2016 International Conf. on Software Analysis, Testing and Evolution, Kunming, China, November 2016, pp. 90-95

[33] Nam, J., Wei, F., Kim, S., Menzies, T., Lin, T.: 'Heterogeneous defect prediction', IEEE Transactions on Software Engineering, 2015, (99), pp. 1-1

[34] J Han, K Micheline.: 'Data mining: Concepts and techniques', Data Mining Concepts Models Methods and Algorithms Second Edition, 2011, 5, (4), pp. 1-18

[35] Fawcett T.: 'An introduction to roc analysis', Pattern Recognition Letters, 2006, 27, (8), pp. 861-874

[36] Tantithamthavorn, C., Mcintosh, S., Hassan, A. E., Matsumoto, K.: 'Automated Parameter Optimization of Classification Techniques for Defect Prediction Models'. Proc. thirty-eighth International Conference on Software Engineering, Austin, USA, 2016, 76, pp. 321-332

[37] Tantithamthavorn, C., Mcintosh, S., Hassan, A. E., Matsumoto, K.: 'The impact of automated parameter optimization on defect prediction models', IEEE Transactions on Software Engineering, 2018

[38] Fu, W., Menzies, T., Shen, X.: 'Tuning for software analytics: is it really necessary?', Information and Software Technology, 2016, 76, pp. 135-146

[39] Tantithamthavorn, C., Mcintosh, S., Hassan, A. E., Matsumoto, K.: 'An empirical comparison of model validation techniques for defect prediction models', IEEE Transactions on Software Engineering, 2017, 43, (1), pp. 1-18

[40] Zhang, F, Keivanloo, I., Zou, Y.: 'Data transformation in cross-project defect prediction', Empirical Software Engineering, 2017, 22, (6), pp. 3186-3218