IET Software

Research Article

Multiple-components weights model for cross-project software defect prediction

ISSN 1751-8806
Received on 18th May 2017
Revised 1st March 2018
Accepted on 29th March 2018
E-First on 15th May 2018
doi: 10.1049/iet-sen.2017.0111
www.ietdl.org

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: qiushaojian@outlook.com*

Abstract: Software defect prediction (SDP) technology is receiving widely attention and most of SDP models are trained on data from the same project. However, at an early phase of the software lifecycle, there are little to no within-project training data to learn an available supervised defect-prediction model. Thus, cross-project defect prediction (CPDP), which is learning a defect predictor for a target project by using labelled data from a source project, has shown promising value in SDP. To better perform the CPDP, most current studies focus on filtering instances or selecting features to weaken the impact of irrelevant cross-project data. Instead, the authors propose a novel multiple-components weights (MCWs) learning model to analyse the varying auxiliary power of multiple components in a source project to construct a more precise ensemble classifiers for a target project. By combining the MCW model with kernel mean matching algorithm, their proposed approach adjusts the source-instance weights and source-component weights to jointly alleviate the negative impacts of irrelevant cross-project data. They conducted comprehensive experiments by employing 15 real-world datasets to demonstrate the advantages and effectiveness of their proposed approach.

1 Introduction

Software quality assurance plays an important role in software life cycle. Potential and unfound defects will affect the quality of software. If the distribution of software defects can be detected at an early stage, it will help the quality assurance team accelerate the search for potential issues, reasonably allocating test resources [1]. In recent years, more and more researchers have paid close attention to software defect prediction (SDP) technology and tried to detect defect-prone modules or files using machine-learning methods. Most existing studies [2–5] of SDP on training a within-project defect prediction (WPDP) model, which attempts to leverage historical defects to predict future defects within a given software project. Nevertheless, in real-world scenarios, there are little to no within-project training data at an early phase of the software lifecycle to learn an available supervised defect-prediction model. Fortunately, there are enough labelled data derived from public data repositories provided by different companies and organisations (e.g. AEEEM [5], PROMISE [6]). From these public repositories, many researchers have proposed several approaches to train predictors via cross-project data.

Cross-project defect prediction (CPDP) is a special application of transfer learning used because labelled data in a target project are difficult to obtain, while abundant labelled data are available in related source projects. By implementing transfer learning, researchers can use the related auxiliary information from a source project for a target project. Existing CPDP approaches [7–10] concentrate on using only cross-project data to build an available target prediction model, such as transfer component analysis (TCA+) [9] approach and combined defect predictor (CODEP) [10]. In these approaches, however, large irrelevant cross-project data usually makes it difficult to build a prediction model with high performance.

To overcome this challenge, many researchers focus on filtering source instances or features that are irrelevant for CPDP tasks. Among these methods, researchers commonly use the nearest neighbour filter (NN filter) [11]. It makes source and target projects more similar in distribution by removing those instances of the source project, that do not appear in the nearest neighbours to target project data. Yu *et al.* [12] select appropriate features to train a CPDP model by applying correlation-based feature selection (CFS), which is a feature subset selection approach that evaluates the individual predictive ability of each feature and redundancy between features. Ma *et al.* [13] propose an approach named transfer naïve Bayes (TNB) that first reweights source instances via a data gravitation (DG) method [14] to weaken the impact of irrelevant source data, and then builds a naïve Bayes classifier on these reweighted source data. Recently, some studies have demonstrated that if there is a certain ratio of labelled data in the target project, it may help to improve the performance of CPDP (e.g. 5% within-project data is not enough to perform WPDP, but it is useful for CPDP). Double transfer boosting (DTB) [15] is an up-to-date approach that employs a small ratio of labelled data in the target project. In DTB, the DG method is also used to initialise the weight of source project data. Then it builds a prediction model using TrAdaboost [16], which applies a limited amount of labelled data in the target project to reweight source data.

The solutions mentioned above, however, only consider the direct impacts of instances or features. In this paper, we advocate that various components, which are composed of similar instances in the source project, have different auxiliary powers for the target project. If the result is materially affected by low auxiliary power components that contain much useless and irrelevant data, it is easy to bring a negative transfer. A toy example can be used to illustrate the situation. In Fig. 1, circles, triangles, and rhombuses represent the instances in a source project; squares represent the instances in a target project; solid shapes represent defective instances, and hollow shapes represent clean instances. As we can see to the left in Part 1, the source project should contain some clusters (hereinafter referred to as components) which are composed of similar instances; each component could, respectively, train a classifier. By using these classifiers to predict the instances of the target project, the accuracies to the right Part 2 show that different component classifiers have various predictive powers for the target project.

To address the above situation, we propose a multiple components weights (MCWs) method whose main idea is to analyse the auxiliary powers of source components by using a

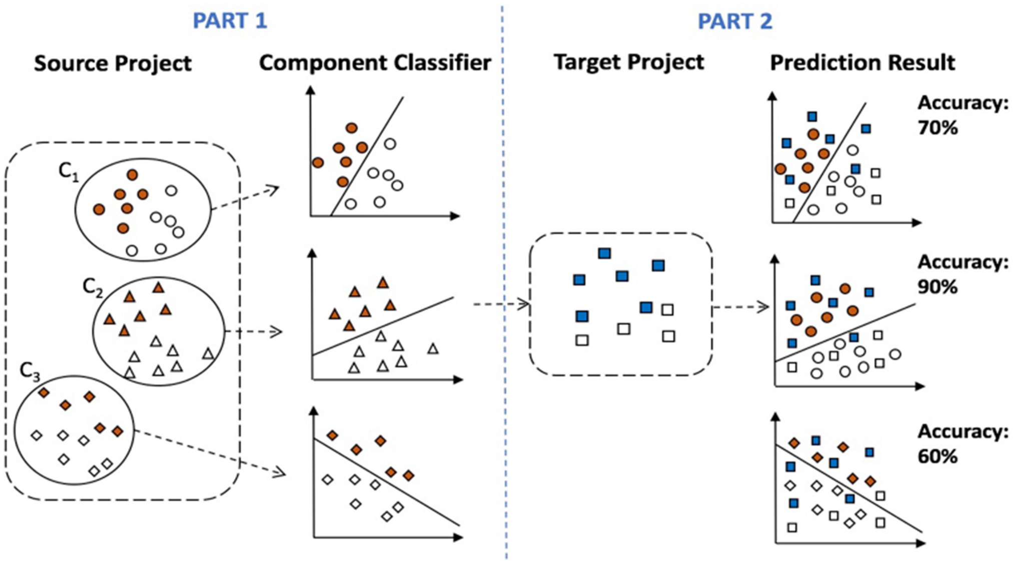


Fig. 1 *Illustration of different components' different auxiliary powers for target project*

small ratio of within-project data. The goal of our proposed approach is to promote the weight of high-auxiliary-power components and cut down the impact of low-auxiliary-power components. In this method, firstly, we separate the source project into multiple components by clustering processes. Secondly, we adopt the kernel mean matching (KMM) [17] algorithm to reduce the distribution divergence between source components and target project. Thirdly, the weights of components' classifiers are adjusted by optimisation manners. Finally, we build an assembled classifier to perform CPDP task. As both KMM and MCW are integrated, we use the name KMM-MCW for our proposed approach.

To assess the KMM-MCW approach, we explore the following research questions.

RQ1: Can KMM-MCW present a better performance than the conventional WPDP method?

RQ2: How effective is KMM-MCW approach compared to other CPDP methods?

The main contributions of this paper are:

(1) After considering the different auxiliary powers of various components in the source project, we propose a novel MCW transfer learning model to reduce the impact of irrelevant cross-project data.
(2) Unlike most previous transfer defect learning methods, we adopt the KMM algorithm to reduce the distribution divergence between source components and target project.
(3) We validate the effectiveness of our proposed methods, KMM-MCW, by conducting extensive experiments on 15 real-world datasets.

This paper is organised as follows: In Section 2, we review some relevant works. In Section 3, we introduce the high-level framework of KMM-MCW and explain its steps in detail. In Section 4, we provide the experimental results of two research questions to validate the effectiveness of our proposed approach. After that, we discuss different approach settings and threats to validity in Section 5. Section 6 concludes our work and outlines the possibilities for future work.

2 Related work

Works related to this paper mainly include CPDP approaches and transfer learning methods. CPDP approaches are divided into two main types: using only the data in a source project or using data from the source project in addition to a small ratio of labelled data from the target project.

2.1 Defect prediction using only source project data

Researchers use the CPDP to handle situations in which the target project of a prediction task has historical data that is too limited to allow for effective WPDP. Zimmermann *et al.* [7] focused on what factors should be considered when selecting projects to build cross-project models and did research on 12 real-world datasets and 622 combinations to analyse the feasibility of CPDP. However, the predictive results of their method were not good enough for CPDP tasks. He *et al.* [8] proposed an approach that could automatically select suitable data for training the defect predictor of a target project and conducted experiments by employing 34 datasets obtained from ten open-source projects. The results of their experiments indicated that the performance of some CPDP approaches could be comparable to WPDP methods.

To improve the performance of CPDP, Turhan *et al.* [11] investigated the availability of cross-project data for building localised defect predictors using static code features. They proposed the NN filter to select data for building CPDP models. Following this work of NN filter, Peters *et al.* [18] developed a model called a Peters filter which employs a different instance selection mechanism to select training data via the structure of source projects. Both the NN filter and Peter filter make project and target projects more similar in distribution, but the removed data may contain potentially useful information for model training. Following prior research, Ma *et al.* [13] proposed the TNB algorithm to transfer defect knowledge of cross-project data to reweight training data. TNB introduced the DG method to assign the weight for source data and TNB model on these reweighted data. The results of TNB indicated that it is more accurate in terms of AUC. However, TNB was designed only for Naïve Bayes classifier. Considering the generality for various classifiers, Nam *et al.* proposed an approach called TCA+ [9] that could be used for most machine-learning algorithms to facilitate performance of CPDP. TCA + built on a transfer learning approach, TCA [19], with some data-normalising rules. The experimental results for eight open-source projects showed that TCA+ could improve cross-project prediction performance. Panichella *et al.* [10] held that various classifiers had different predictive powers over CPDP tasks. They proposed an approach named CODEP, which combined six classification algorithms for CPDP.

To explore the importance of instance and feature for CPDP, Yu *et al.* [12] experimented with the NN filter and CFS to evaluate the significance of instance and feature for CPDP. The results indicated that feature selection performed better than instance filtering for CPDP, so Yu *et al.* suggested that researchers should pay more attention to the former. The applicability of this suggestion needs further validation, however. Allowing for the usability of CPDP for a large set of projects, Zhang *et al.* [20] attempted to build a universal defect prediction model for a large set of projects (1398 open-source projects obtained from SourceForge and GoogleCode). They first clustered these projects based on the similarity of distribution, then proposed a context-aware rank transformation method to pre-process them. After rank transformation, they built a universal model using varying software metrics. The results showed that the method was effective and had potential for application to industrial practice but that it usually required sufficient source projects to be successful.

2.2 Defect prediction using source project data in addition to a small ratio of labelled data from the target project

The approaches mentioned in Section 2.1 only use cross-project data to train target classifiers, but in recent years, as scholars have found that a small amount of within-project data is helpful for CPDP tasks and reasonably priced, more and more researchers have begun to focus on how to make full use of limited within-project data [15, 21–23].

Turhan *et al.* [21] investigated the feasibility of using mixed-project data for defect prediction. They used the data obtained from 73 versions of 41 public projects and compared the performances using within-project data and mixed-project data. The results showed that limited within-project data helped improve the performance of CPDP, which is comparable to WPDP. Chen *et al.* [15] proposed the DTB model, which integrated two layers of data transfer. Firstly, it used the DG method to reshape the source project's data distribution to fit that of the target project. Secondly, transfer Aaboost (TrAaboost) [16] method was employed with a small amount of labelled within-project data to weaken negative transfer. The DTB method improved predictive performance by using labelled within-project data, but the potential value of these data could be further mined.

In recent years, researchers have applied the CPDP method with a small ratio of labelled within-project data to multi-source CPDP tasks. Yu *et al.* [22] proposed a multi-source TrAaboost (MSTrA) approach that employed a small ratio of labelled within-project data to weaken the impact of irrelevant cross-project data. Yu *et al.* imported transferring knowledge not from one project but from multiple sources to avoid negative transfer. Xia *et al.* [23] introduced the hybrid model reconstruction approach (HYDRA) for CPDP, which includes two steps: a genetic algorithm (GA) step and an ensemble learning step. HYDRA first built a composite GA predictor on multiple sources and assigns different weights to each inner underlying classifier by GA. It then repeats the GA step many times, creating a number of GA classifiers. Finally, HYDRA uses Adaboost [24] to assigns a weight to each GA predictor. Both the MSTrA and HYDRA methods avoid the bias of a single source, but they require either labelled target data, or multiple source projects for consensus learning.

2.3 Transfer learning

Most machine-learning approaches can achieve good performance when training data is sufficient [25]. However, in most real-world scenarios, sufficient training data for a classification task is hard to obtain due to the high expense of human labelling. Researchers are expected to leverage some auxiliary source data to help the learning in the target domain, but the distributions of source domain and target domain are different. To solve this issue, transfer learning [25] is proposed to extract common knowledge across domains so that a model trained on one domain can be applied effectively to others. In the past decade, an increasing number of scholars have worked to develop transfer learning methods in machine learning area. In many real-world applications (e.g. sentiment analysis [26] and image classification [27]), transfer learning has proven to be promising.

A crucial research issue in transfer learning is how to reduce the difference between source and target domains while preserving original data properties. Previous research on transfer learning can be classified into two types: feature-representation-based and instance-based transfer learning. The feature-representation-transfer approach aims at finding proper feature representations to minimise domain divergence. Pan *et al.* [28] proposed a dimensionality reduction approach by discovering the latent feature space. Then Pan *et al.* use supervised learning algorithms to train classification models and obtain satisfying results. After that, Pan *et al.* also proposed the TCA [19] algorithm that attempts to minimise the distributions divergence between domains in a reproducing-kernel Hilbert space.

A very fruitful and intuitively appealing transfer learning method for CPDP is the instance-based approach. Although researchers cannot directly use the source domain data, certain relevant parts of the data can be reused to train a model for a target project. Huang et al. [17] presented the KMM method, a non-parametric method that produces resampling weights without distribution estimation. The method works by matching distributions between training and testing sets in a projecting space. Dai *et al.* [16] proposed the TrAdaoost, which is an extension of the Adaboost [24] algorithm to address transfer learning problems. TrAdaboost assumes that source and target domain data use exactly the same set of features and labels but that the distributions of the data in these two domains are different.

As mentioned above, the problem setting of CPDP is related to the transfer learning setting. Transfer learning addresses issues by transferring knowledge extracted from a related, but different domain. In CPDP, it can be regarded as transferring defect knowledge from the source project to build predictive models for the target project. In this paper, our goal is to find a way that effectively leverage limited labelled target data to analyse the different auxiliary powers of multiple components of the source project.

3 Proposed approach

In this section, we present the KMM-MCW approach in detail. Frequently used notations are shown in Table 1. At this end of this section, we have included the whole pseudo-code of KMM-MCW.

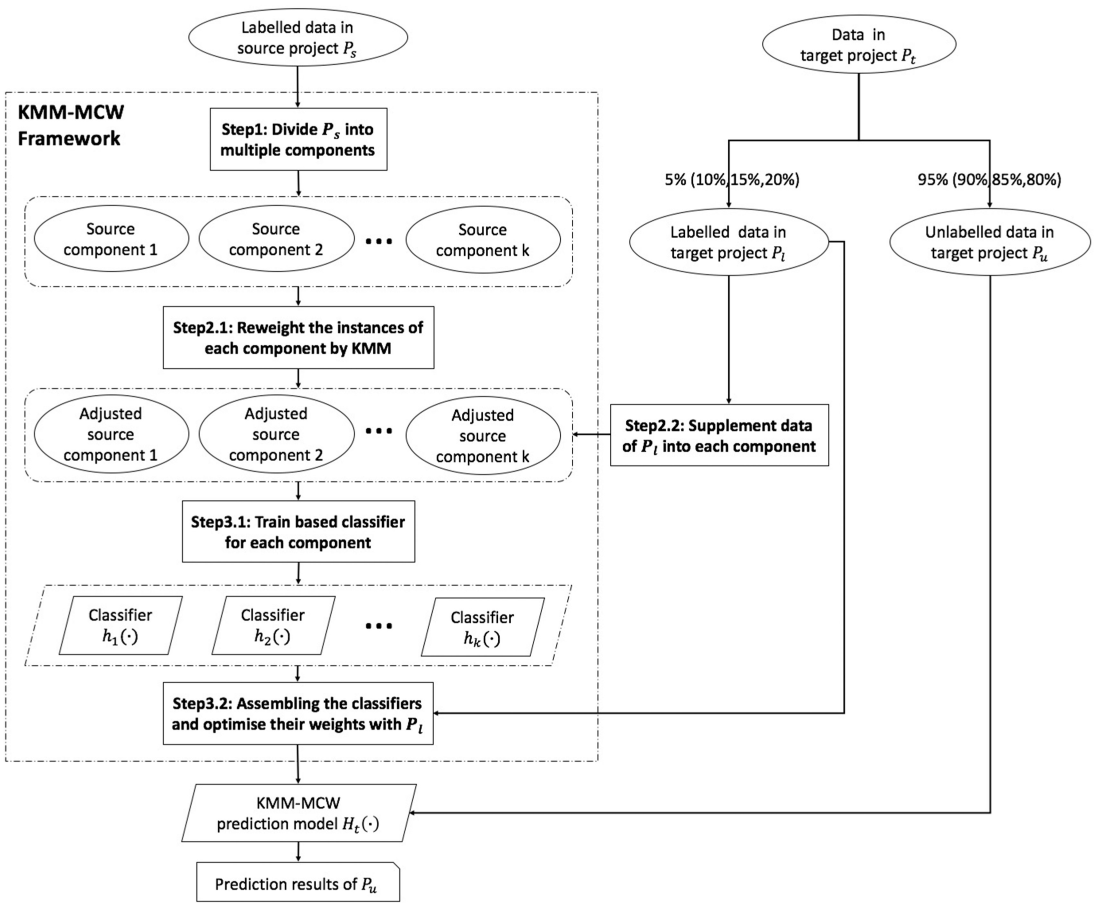
3.1 Problem definition

Given a target project P_t and target project P_s under the assumption that their feature spaces $\mathcal{X}_t = \mathcal{X}_s$ and binary label sets $\mathcal{Y}_t = \mathcal{Y}_s = \{0, 1\}$. An instance is clean ($y = 0$) if it has no bug. Otherwise, it is defective ($y = 1$). Let a sequence of labelled instances $\{(x_i, y_i) | i = 1, 2, \dots, n^s\}$ denote the instances of P_s. Note that the CPDP task we focus on in this paper is the defect prediction using both data in a source project and a small ratio of labelled data in its target project. As a result, we, respectively, use P_l and P_u to denote labelled data and unlabelled data of the target project, where $P_l \cup P_u = P_t$, $P_l = \{(x_i, y_i) | i = 1, 2, \dots, n^l\}$ and $P_u = \{(x_i) | i = 1, 2, \dots, n^u\}$.

In our work, we assume that the small ratio of labelled data P_l is not enough to train a good within-project defect-predictor. The goal of a CPDP task is to extract shared defect prediction knowledge from P_s to solve defect prediction problems in P_t. However, in general, large irrelevant cross-project in P_s may weaken the auxiliary power of P_s and even bring negative transfer, so we cannot apply the source-labelled instances directly.

Table 1 Summary of frequently used mathematical notations

Notations	Mathematical meanings
P_t,P_s	the target/source project
n^t,n^s	the number of instances in the target/source project
P_l,P_u	the labelled/unlabelled instances in target project
k	the number of components in the source project
m_c	the number of instances in the cth source component
$\boldsymbol{\alpha}_c$	the weight vector for the instances of the cth source component
$h_c(\cdot)$	the classifier trained by the cth source component
ω_c	the weight for the $h_c(x)$
$H_t(\cdot)$	the target ensemble classifier, $H_t(x) = \sum_{c=1}^{k} \omega_c h_c(x)$.



Labelled data in source project P_s
Data in target project P_t
KMM-MCW Framework
Step1: Divide P_s into multiple components
5% (10%,15%,20%)
95% (90%,85%,80%)
Source component 1
Source component 2
Source component k
Labelled data in target project P_l
Unlabelled data in target project P_u
Step2.1: Reweight the instances of each component by KMM
Adjusted source component 1
Adjusted source component 2
Adjusted source component k
Step2.2: Supplement data of P_l into each component
Step3.1: Train based classifier for each component
Classifier $h_1(\cdot)$
Classifier $h_2(\cdot)$
Classifier $h_k(\cdot)$
Step3.2: Assembling the classifiers and optimise their weights with P_l
KMM-MCW prediction model $H_t(\cdot)$
Prediction results of P_u

Fig. 2 *Framework of KMM-MCW*

To address this challenging scenario, we designed the KMM-MCW approach, which constructs a precise ensemble classifier for the target project by estimating the auxiliary power of multiple components in the source project.

3.2 Overall framework of KMM-MCW

Fig. 2 shows the overall framework of KMM-MCW. As illustrated in Fig. 2, this approach contains three main steps to alleviate the potential impacts of irrelevant cross-project data. The steps are introduced as following:

Step 1: Dividing multiple components: In this step, spectral clustering [29] is used to partition source data into k clusters. In this paper, these clusters are referred to as components. Given a set of source instances $X_s = \{x_1, x_2, \ldots, x_{n^s}\}$, the goal is to cluster these instances into k clusters, where k is an input parameter. The main idea of the spectral clustering algorithm is to find a partition of the graph that satisfied two conditions: different clusters' points are dissimilar and the same clusters' points are similar.

Step 2: Adjusting the weights of instances in components: For reducing sample distribution divergence between source components and the target project, our approach used KMM algorithm to calculate the weights of components' instances. After that, to make the most of the known defect knowledge in target project, we supplemented the small ratio of target-labelled instances P_l into each component.

Step 3: Learning the weights of components: In this step, we train classifiers for each adjusted component and utilise the labelled in P_l to calculate classifiers' predictive power (i.e. accuracy). Then we convert the predictive powers of classifiers into the initial weights of components. At last, a target project is built by assembling these component classifiers, and their corresponding weights begin learning by optimisation tools.

When the above three steps are carried out, k components are captured in the source project and corresponding k classifiers are trained. The goal of KMM-MCW becomes the assembly of k classification models to construct the target classifier H_t. In our proposed learning task, the target classification model can be formulated as follows:

$$H_t(x) = \sum_{c=1}^{k} \omega_c h_c(x) \tag{1}$$

where $h_c(x)$ is the classifier trained by the cth source component. ω_c is the predictive weight for $h_c(x)$.

As we can see in (1), the key challenge of our approach is how to effectively train classifier $h_c(x)$ by adjusted instances in each component (in relation to Step 2) and learn its transferring weight vector $\boldsymbol{wc}$ (in relation to Step 3). Next, we will show Steps 2 and 3 in detail.

3.2.1 Adjusting the weights of instances in components: Step 2 of our approach is applying the KMM algorithm, which is a non-parametric method, to directly infer weights for the source project without distribution estimation [17]. In our approach, the KMM algorithm will independently work on each source component. In other word, KMM algorithm will be applied k times.

Every time the KMM is applied, it will utilise both component data and target data (whether they are labelled or not) to build a potential transferring bridge. In general, there is no way to infer a good estimator based on two different joint distributions in source component $Pr_c(x, y)$ and target project $\Pr_t(x, y)$ because of the two distributions could be arbitrarily far apart [30]. Due to this unsolvable estimation problem, KMM assumes that the conditional probability of these two distributions are fixed, meaning that $\Pr_c(y|x) = \Pr_t(y|x)$. Based on the above assumption and the theorem stating that $\Pr(x, y) = \Pr(y|x) \times \Pr(x)$, $\Pr_c(x, y)$ and $\Pr_t(x, y)$ are only affected by $\Pr_c(x)$ and $\Pr_t(x)$, respectively. In this

```
Input:
$P_s$: Labelled data in source project
$P_l$: Labelled data in target project
$P_u$: Unlabelled data in target project
$k$: Component number
Output:
Ensemble classifier $H_t(x, \omega)$
1: Divide $P_s$ into $k$ components using spectral clustering method;
2: for $c = 1, 2, \dots, k$ do
3:     Compute $\alpha_c$ to reweight the instances of $c$th component by solving (3);
4:     Add $P_l$ into each component;
5:     Learn base classifier $h_c$ by using the data of the $c$th component;
6:     Calculate the prior-accuracies $\mathrm{acc}_c$ of $h_c$ using $P_l$;
7: end for
8: Convert vector acc to $\omega^*$ by (4);
9: Initialize $\omega$ using $\omega^*$;
10: Solve (6) to obtain the optimal $\omega$;
11: Assemble classification models learned from each component with
    optimal $\omega$ and construct the target classifier $H_t(x, \omega) = \sum_{c=1}^{k} \omega_c h_c(x)$;
```

Fig. 3 *Algorithm 1: KMM-MCW defect prediction model*

case, the estimation problem is changed to reduce the marginal distribution divergence between $\Pr_c(x)$ and $\Pr_t(x)$.

Based on these considerations, KMM introduces the maximum mean discrepancy (MMD) [31], which is used to measure the similarity of two distributions. The main idea of KMM is to calculate appropriate weights for each instance of source component to minimise MMD between $\Pr_c(x)$ and $\Pr_t(x)$ in a reproducing-kernel Hilbert space. We denote x_i^c as the ith sample in the cth source component and x_i^t as the ith sample in the target project. After space mapping, the task of minimising MMD between the cth source component and target project can be formulated as follows:

$$\min_{\alpha_c} \left\| \frac{1}{m_c} \sum_{i=1}^{m_c} \alpha_c \phi(x_i^c) - \frac{1}{n^t} \sum_{i=1}^{n^t} \phi(x_i^t) \right\|_{\mathscr{H}}^2 \tag{2}$$

where m_c and n^t, respectively, are the number of instances in the cth source component and target project, α_c is an m_c dimensional weight vector for the instances of cth component, and $\phi(\cdot)$ represents a feature map on a reproducing-kernel Hilbert space.

Formula (2) is a constrained quadratic programming problem [17]. To make sure that $\alpha_c \Pr_c(x)$ is close to $\Pr_t(x)$, three constraints are added to the formulation: lower bound of α, the upper bound of α and $|(1/m_c)\sum_{i=1}^{m_c} \alpha_i - 1| \le \epsilon$. In this paper, we set $\mathrm{LB} = 0$ and $\mathrm{UB} = 1$. For ϵ, a good choice would be $O(\mathrm{UB}/\sqrt{m_c})$. Now, the formula can be rewritten as

$$\begin{aligned} &\min_{\alpha_c} \frac{1}{2} \alpha_c^{\top} R \alpha_c - \mathscr{R}^{\top} \alpha_c \\ &\mathrm{s.t.}\ \left| \frac{1}{m_c} \sum_{i=1}^{m_c} \alpha_i - 1 \right| \le \epsilon, \mathrm{LB} \le \alpha_i \le \mathrm{UB} \end{aligned} \tag{3}$$

where $\mathscr{R}_i = n^t m_c \sum_{j=1}^{m_c} \exp\left(-||x_i^c - x_j^t||^2 2\sigma^2\right)$ and $R_{ij} = \exp\left(-||x_i^c - x_j^c||^2 2\sigma^2\right)$, x_i^c represents the ith instance of cth component and x_j^t is the jth instance of the target project, σ is the parameter of function width.

Formula (3) is a standard quadratic problem that could be solved by a number of algorithms and tools. In this paper, we use the ‘quadprog’ function in MATLAB to get the optimal solution. After this step, the instances of k components are reweighted.

In practice, the members in some components are so few that they cannot be used to train a model. To solve this issue, labelled data from the target project are, respectively, added into each component, and the weight of these labelled instances is 1. Doing this allows for the components that have not enough members to be supplemented and the remaining components to be corrected.

3.2.2 Learning the weights of components:

At the beginning of this step, the based classifier is trained for each component and their prior-accuracies are calculated via known labelled data in the target project. Here we denote these prior-accuracies as $\mathrm{acc}: \mathrm{acc} \in \mathbb{R}^{k*1}$ and $\mathrm{acc} = \{\mathrm{acc}_1; \mathrm{acc}_2; \dots \mathrm{acc}_c; \dots; \mathrm{acc}_k\}$. Then acc is used as the evaluation of predictive power to initialise transfer weight of each component. We denote the initial weight vector as $\omega^*: \omega^* \in \mathbb{R}^{k*1}$ and $\omega^* = \{\omega_1^*; \omega_2^*; \dots \omega_c^*; \dots; \omega_k^*\}$

$$\omega_c^* = \frac{\mathrm{acc}_c}{\sum_{i=1}^{k} \mathrm{acc}_i} \tag{4}$$

So far, our approach has trained k classification models $\{h_1, h_2, \dots, h_c, \dots, h_k\}$ and their corresponding initialised weight $\{\omega_1^*, \omega_2^*, \dots \omega_c^*, \dots, \omega_k^*\}$. As mentioned before, the classification models are assembled with each component to construct the target classifier H_t. We rewrite the H_t as follows:

$$H_t(x, \omega^*) = \sum_{c=1}^{k} \omega_c^* h_c(x) \tag{5}$$

Next we will show how to optimise weights of each classifier for the sake of achieving a more precise target classifier. By utilising the data in P_l, the task is to solve the following optimisation problem:

$$\begin{aligned} &\min_{\omega} \sum_{i=1}^{n_l} ||\hat{\mathbf{y}}_i \times \omega - y_l^i|| + \lambda ||\omega - \omega^*|| \\ &\mathrm{s.t.} \sum_{i=1}^{k} \omega_i = 1, \omega_i \ge 0 \end{aligned} \tag{6}$$

where $\omega \in \mathbb{R}^{k*1}$ and $\omega = \{\omega_1; \omega_2; \dots; \omega_c; \dots; \omega_k\}$ is the weight vector to be adjusted, n_l is the number of labelled data in the target project, $\hat{\mathbf{y}}_i \in \mathbb{R}^{1*k}$ represents the predictive label vector formed by $\{h_1(x_l^i), h_2(x_l^i), \dots, h_c(x_l^i), \dots, h_k(x_l^i)\}$. The result of $\hat{\mathbf{y}}_i \times \omega$ represents the predicted value of x_l^i, y_l^i is the true label of x_l^i. Besides, $||\cdot||$ denotes the L^2-norm of matrix, and λ is a non-negative parameter to control the regularisation term. In this paper, we set $\lambda = 1$.

Now the minimisation problem is a standard optimisation problem and could be solved by applying many existing solvers. In this paper, our approach uses the CVX toolbox [32] to tackle the optimisation process. Algorithm 1 (see Fig. 3) presents the pseudo-code of the KMM-MCW approach to performing CPDP.

4 Experiments

This section illustrates the experiments in detail. The description mainly contains the experiment dataset, processing of data imbalances, evaluation measures, experiment design and results.

4.1 Experiment dataset

To assess KMM-MCW, we selected 15 different open-source projects from the PROMISE repository which collected by Jureczko and Madeyski [33]. Each project consists of a collection of Java classes that includes two parts: 20 static code attributes and a label attribute (defective or clean). Table 2 presents these 15 projects' essential information, including project names, project versions, number of instances and defect rate. Table 3 shows the attributes for these datasets that are widely used by SDP models [15, 23, 33]. Note that, to verify the generality, the dataset is composed of several projects with different sizes and defective rates.

4.2 Processing of data imbalances

As we can see in Table 2, some projects (e.g. 'elearn' and 'tomcat') had a low defect rate, and some projects (e.g. 'log4j' and 'xerces') had a high defect rate. These values indicate that there is a great imbalance between the clean and defective classes in the PROMISE datasets.

In fact, unbalanced distribution is a typical characteristic of software defects [15]. If we conduct classification algorithms on a highly skewed dataset, the class imbalance problem may bring difficulties for learning because the trained classifiers tend to prefer the majority class and have a weaker ability to classify the minority class. Some studies [34, 35] have shown that the performance of such CPDP model is also susceptible to class imbalance problems. To alleviate this issue, techniques of class imbalance learning are widely used.

Table 2 Fifteen datasets selected form the PROMISE repository [6], sorted in order of the name (*defect rate*: the percentage of defective instances)

Project name	Project version	Instance count	Defect rate, %
ant	1.7	745	22.3
arc	1	234	11.5
camel	1.6	965	19.5
elearn	1	64	7.8
jedit	4.3	492	2.2
log4j	1.2	205	92.2
lucene	2.4	340	59.7
poi	3.0	1077	63.6
prop	6	660	10.0
redaktor	1	176	15.3
synapse	1.2	256	33.6
tomcat	6.0	858	9.0
velocity	1.6	229	34.1
xlan	2.7	909	98.8
xerces	1.4	588	74.3

In the present experiments, we used the synthetic minority oversampling technique (SMOTE) to pre-process the training data before implementing the CPDP models. SMOTE [36] is an over-sampling technique that can synthetically create new sample with minority class to provide more balanced class distribution for learning task.

4.3 Evaluation measures

To assess prediction performance, we used accuracy and *F*-measure as the evaluation metrics. Accuracy is the common evaluation measure for traditional transfer learning tasks [26, 27] and *F*-measure is the harmonic mean of precision and recall that is frequently used in CPDP in recent years [9, 13, 23]. These two evaluation measures are defined below.

There are four possible outcomes for test instances in a classification task: a truly defective instance, classified as defective (true positive); a clean instance, classified as defective (false positive); predicting a truly defective instance as clean (false negative); and predicting a truly clean instance as clean (true negative). Based on these four outcomes, the accuracy and *F*-measure are defined below:

Accuracy: the proportion of instances that are correctly labelled among all test instances

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precision: the proportion of the number that are correctly labelled as defective to the number of those truly defective instances

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

Recall: the proportion of defective instances that are correctly labelled

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

In fact, there is a trade-off between precision and recall. The trade-off causes the difficulties to compare the performance of prediction models by using only precision or recall. In this paper, we use *F*-measure, which is the balanced evaluation metrics of precision and recall, to avoid the trade-off issue.

Table 3 List of attributes from PROMISE [33]. The descriptions of attributes are referred to [23]

Attribute	Description
dit	the maximum distance from a given class to the root of an inheritance tree
noc	number of children of a given class in an inheritance tree
cbo	number of classes that are coupled to a given class
rfc	number of distinct methods invoked by code in a given class
lcom	number 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	number of public methods in a given class
loc	number of lines of code in a given class
dam	the ratio of the number of private/protected attributes to the total number of attributes in a given class
moa	number of attributes in a given class which are of user-defined types
mfa	number of methods inherited by a given class divided by the total number of methods that can be accessed by the member methods of the given class
cam	the ratio of the sum of the number of different parameter types of every method in a given class to the product of the number of methods in the given class and the number of different method parameter types in the whole class
ic	number of parent classes that a given class is coupled to
cbm	total number 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 number of classes that de- pends upon a given class
ce	efferent coupling, which measures the number 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

F-measure: a dual assessment of both precision and recall. It evaluates if either precision or recall is low, according to which evaluation the F-measure is decreased or increased

$$F\text{-measure} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

4.4 Experiment design and results

To evaluate our proposed approach, we conducted the experiments on the 15 selected PROMISE datasets according to the two research questions mentioned in the Introduction. Following are the design details and experimental results for these two research questions:

RQ1: Can KMM-MCW demonstrate a better performance than the conventional WPDP method?

To assess the performance of WPDP method, we randomly selected 5% (10, 15 and 20%) of within-project instances as the training data to learn a predictive classifier for labelling remaining 95% (90, 85 and 80%) of testing instances.

In this RQ1, we were interested in discovering whether our KMM-MCW approach, which leverages labelled data from other projects, could achieve comparable result with WPDP when there was a small ratio of labelled within-project data available. To evaluate the performance of KMM-MCW, we used both the 5% (10, 15 and 20%) of within-project instances and the all labelled data in the source project as the training data to perform the prediction.

In these comparison experiments, we conducted m-fold cross-validation on KMM-MCW and WPDP methods, respectively. Here, $m = 100\%/l\%$, where $l \in \{5, 10, 15, 20\}$ is the ratio of labelled data in the target project. For example, if $l = 10$, $m = 100\%/10\% = 10$, so that ten-fold cross-validation experiments were conducted. Considering the process of data selection involves a degree of randomness, we ran the m-fold cross-validation 20 times, recording the results of accuracy and F-measure. For underlying classifier, we choose logistic regression. Its implementation and parameters settings were consistent for KMM-MCW and WPDP. When we performed KMM-MCW, we set the components number at 4.

Tables 4 and 5, respectively, present the mean accuracies and F-measures of KMM-MCW and WPDP methods with 5, 10, 15 and 20% of within-project data. The better results of comparison are in bold, and the penultimate rows of these tables show the win/tie/loss counts of the KMM-MCW versus the WPDP method. Each predictive result of the KMM-MCW entered in Tables 4 and 5 is the average performance of 14 KMM-MCW models, as calculated by the following procedure. First, given a target project, we, respectively, evaluated the performance of models that trains by one source project. Across 14 source projects, we got 14 prediction models for the given target project. Second, we computed the mean performance of these 14 models and entered it into table.

Across the experiment on 15 datasets, Tables 4 and 5 show that the average results of KMM-MCW outperformed the WPDP method with 5, 10, 15 and 20% within-project data by 4.1,3.6, 2.5 and 1.9% on accuracy and 9.5, 6.1, 2.3 and 1.9% on F-measure. Comparing WPDP methods, the penultimate rows of Tables 4 and 5 show that KMM-MCW could win the most performance comparisons with WPDP method. By observing the comparative result, we can found that the advantage of KMM-MCW is more outstanding with fewer labelled data in the target project.

As the result shown in Tables 4 and 5, the performances of KMM-MCW outperform the WPDP method with same ratio of target-labelled data. It proves that our approach could be used to assist the learning of the target project prediction model. In addition, in some target datasets, the performances of KMM-MCW with 5% (10%, 15%) target-labelled data are better than the WPDP method with 10% (15%, 20%) target-labelled data. It means that the work of SDP can start earlier in the software life cycle by using our approach. In conclusion, we believe the improvements would provide more effective help for the quality assurance team to detect the software defects.

RQ2: How effective is KMM-MCW approach compared to other CPDP methods?

To confirm whether the KMM-MCW approach could perform better than other CPDP approaches, we compared it with seven CPDP approaches on the 15 datasets from the PROMISE repository. These seven CPDP approaches including: (1–2) two CPDP approaches designed with a limited amount of labelled within-project data (DTB [15] and TrAdaboost [16]); (3–5) three CPDP methods using only cross-project data (TNB [13], TCA + [9] and NN filter [11]); (6) the KMM [17] method (it was adopted because it is a main step in KMM-MCW); (7) a baseline using an underlying classifier trained by the source project to directly predict the defects of the target project. The source codes of KMM, TCA + and TrAdaboost were provided by their author. We re-implemented the DTB, TNB and NN filter. Noted that, to make a fair comparison, for TNB, TCA+, NN filter, KMM and baseline, we also supplemented the same ratio of labelled target data into training data to perform the CPDP.

For these CPDP methods, by default, 5% instances were randomly selected in the target project to construct the target

Table 4 Accuracy performance comparison of our approach (KMM-MCW) compared with the WPDP that using different percentage labelled data (5, 10, 15, 20%).

Ratio	5%		10%		15%		20%	
datasets	KMM-MCW	WPCP	KMM-MCW	WPCP	KMM-MCW	WPCP	KMM-MCW	WPCP
ant	0.723	0.700	0.755	0.743	0.781	0.775	0.797	0.791
arc	0.742	0.759	0.759	0.705	0.769	0.743	0.776	0.758
camel	0.697	0.686	0.774	0.761	0.785	0.776	0.793	0.789
elearn	0.824	0.842	0.820	0.840	0.840	0.776	0.834	0.798
jedit	0.900	0.874	0.948	0.950	0.936	0.929	0.931	0.925
log4j	0.818	0.792	0.842	0.775	0.845	0.817	0.825	0.806
lucene	0.586	0.535	0.606	0.586	0.640	0.623	0.662	0.656
poi	0.658	0.573	0.680	0.658	0.710	0.687	0.731	0.716
prop	0.782	0.769	0.805	0.788	0.854	0.840	0.874	0.857
redaktor	0.677	0.670	0.718	0.651	0.759	0.732	0.771	0.729
synapse	0.625	0.565	0.626	0.586	0.664	0.647	0.637	0.617
tomcat	0.836	0.819	0.841	0.834	0.871	0.862	0.888	0.881
velocity	0.606	0.570	0.626	0.551	0.658	0.651	0.672	0.662
xalan	0.957	0.935	0.964	0.956	0.973	0.968	0.975	0.969
xerces	0.726	0.642	0.808	0.776	0.831	0.798	0.847	0.832
W/T/L	13/0/2		13/0/2		15/0/0		15/0/0	
AVG	0.744	0.715	0.771	0.744	0.794	0.775	0.801	0.786

The penultimate row presented the win/tie/loss counts of KMM-MCW versus WPDP approach and the last row showed the average accuracies. The better accuracies are in bold.

Table 5 *F*-measure performance comparison of our approach (KMM-MCW) compared with the WPDP that using different percentage labelled data (5, 10, 15, 20%).

Ratio	5%		10%		15%		20%	
datasets	KMM-MCW	WPCP	KMM-MCW	WPCP	KMM-MCW	WPCP	KMM-MCW	WPCP
ant	0.421	0.386	0.448	0.429	0.468	0.455	0.475	0.458
arc	0.181	0.166	0.268	0.265	0.216	0.187	0.250	0.234
camel	0.284	0.279	0.281	0.282	0.260	0.249	0.230	0.224
elearn	0.166	0.102	0.202	0.105	0.219	0.194	0.175	0.186
Jedit	0.072	0.020	0.064	0.020	0.095	0.088	0.076	0.061
log4j	0.899	0.879	0.914	0.865	0.916	0.897	0.904	0.889
lucene	0.645	0.593	0.663	0.638	0.689	0.671	0.713	0.706
Poi	0.726	0.635	0.742	0.719	0.775	0.756	0.793	0.779
Prop	0.176	0.175	0.156	0.157	0.116	0.132	0.118	0.120
redaktor	0.279	0.188	0.337	0.298	0.356	0.346	0.375	0.367
synapse	0.465	0.422	0.457	0.445	0.493	0.487	0.460	0.441
tomcat	0.213	0.242	0.225	0.228	0.255	0.250	0.276	0.265
velocity	0.452	0.410	0.455	0.399	0.490	0.492	0.516	0.506
xalan	0.980	0.966	0.983	0.977	0.988	0.984	0.989	0.984
xerces	0.802	0.722	0.863	0.835	0.883	0.860	0.895	0.884
W/T/L	14/0/1		12/0/3		13/0/2		13/0/2	
AVG	0.451	0.412	0.471	0.444	0.481	0.470	0.483	0.474

The penultimate row presented the win/tie/loss counts of KMM-MCW versus WPDP approach and the last row showed the average F-measures. The better F-measures are in bold

Table 6 Accuracy results (average accuracies of 14 source prediction models to predict 1 target project) of different CPDP approaches (KMM-MCW, DTB, TrAdaboost, TNB, TCA + , NN filter, KMM and baseline) on the 15 PROMISE datasets.

Datasets	KMM-MCW	DTB	TrAdaboost	TNB	TCA+	NN filter	KMM	Baseline
ant	0.723	0.649	0.642	0.722	0.539	0.558	0.703	0.558
arc	0.742	0.641	0.638	0.682	0.548	0.581	0.627	0.577
camel	0.697	0.647	0.648	0.714	0.553	0.593	0.691	0.592
elearn	0.824	0.638	0.624	0.639	0.598	0.608	0.676	0.622
jedit	0.900	0.670	0.658	0.874	0.422	0.515	0.715	0.513
log4j	0.818	0.657	0.662	0.757	0.536	0.568	0.598	0.573
lucene	0.586	0.590	0.588	0.558	0.566	0.571	0.565	0.567
poi	0.658	0.684	0.684	0.668	0.621	0.641	0.657	0.640
prop	0.782	0.716	0.716	0.797	0.502	0.554	0.742	0.554
redaktor	0.677	0.598	0.594	0.637	0.438	0.533	0.571	0.500
synapse	0.625	0.578	0.588	0.580	0.540	0.562	0.577	0.556
tomcat	0.836	0.800	0.798	0.826	0.582	0.703	0.778	0.696
velocity	0.606	0.562	0.559	0.546	0.552	0.533	0.575	0.532
xalan	0.957	0.871	0.872	0.937	0.673	0.743	0.893	0.743
xerces	0.726	0.733	0.738	0.801	0.558	0.632	0.714	0.628
W/T/L		12/0/3	12/0/3	11/0/4	15/0/0	15/0/0	15/0/0	15/0/0
AVG	0.744	0.669	0.667	0.715	0.548	0.593	0.672	0.590

The penultimate row presented the win/tie/loss counts of KMM-MCW versus other approaches and the last row showed the average accuracies. The best accuracies are in bold

training data. We followed the randomness setting of the experiments in RQ1, conducting *m*-fold cross-validation on KMM-MCW and other seven CPDP methods 20 times.

We also chose logistic regression as the underlying classifier and its parameters settings are consistent for each time and each approach. Tables 6 and 7 present the mean accuracies and *F*-measures of KMM-MCW compared with DTB, TrAdaboost, TNB, TCA+, NN filter, KMM and baseline, respectively. The best accuracies and *F*-measures are in bold, and the penultimate rows of these tables show the win/tie/loss counts of KMM-MCW versus other approaches.

Table 6 shows the mean values of accuracy results. The accuracies of KMM-MCW varied from 0.586 to 0.957. The penultimate row of Tables 6 shows that KMM-MCW won the most of performance comparisons on the 15 datasets. The average accuracy of KMM-MCW was 0.744. It outperforms DTB, TrAdaboost, TNB, TCA+, NN filter, KMM and baseline by 11.2, 11.5, 4.1, 35.8, 25.5, 10.7 and 26.1%.

The mean values of *F*-measures results are shown in Table 7. As we can see, the *F*-measures of KMM-MCW varied from 0.072 to 0.980. In the experiment on 15 datasets, the penultimate row of Table 7 shows that KMM-MCW can win most of the performance comparisons. The average *F*-measures of KMM-MCW is 0.451. It outperforms DTB, TrAdaboost, TNB, TCA + , NN filter, KMM and baseline by 4.4, 3.2, 7.1, 17.8, 14.8, 6.9 and 13.9%, respectively.

The experiments undertaken to find the response to RQ2 show that KMM-MCW performs better than the other seven CPDP models.

5 Discussion

5.1 Impact of the number of clusters in the KMM-MCW

In this paper, we advocate that various components in the source project have different auxiliary power for target project. The main idea of MCW is to cluster multiple components of the source project and assemble their power to predict the defects of the target project. In the above experiments, by default, KMM-MCW's component number was set to be 4. Here, the number of components was, respectively, changed from 1 to 8 to verify the

Table 7 *F*-measure results (average *F*-measure of 14 source prediction models to predict one target project) of different CPDP approaches (KMM-MCW, DTB, TrAdaboost, TNB, TCA+, NN filter, KMM and baseline) on the 15 PROMISE datasets.

Datasets	KMM-MCW	DTB	TrAdaboost	TNB	TCA+	NN filter	KMM	Baseline
ant	0.421	0.394	0.396	0.407	0.394	0.370	0.413	0.373
arc	0.181	0.199	0.216	0.138	0.233	0.218	0.197	0.227
camel	0.284	0.295	0.304	0.260	0.263	0.260	0.288	0.262
elearn	0.166	0.143	0.160	0.121	0.173	0.131	0.117	0.190
jedit	0.072	0.043	0.048	0.044	0.049	0.050	0.047	0.050
log4j	0.899	0.765	0.771	0.854	0.607	0.659	0.715	0.661
lucene	0.645	0.650	0.647	0.625	0.571	0.597	0.616	0.592
poi	0.726	0.753	0.754	0.738	0.639	0.679	0.720	0.678
prop	0.176	0.166	0.173	0.160	0.195	0.176	0.168	0.176
redaktor	0.279	0.236	0.239	0.190	0.201	0.207	0.242	0.203
synapse	0.465	0.462	0.465	0.388	0.451	0.458	0.448	0.444
tomcat	0.213	0.208	0.205	0.188	0.198	0.180	0.196	0.180
velocity	0.452	0.438	0.438	0.390	0.427	0.414	0.435	0.412
xalan	0.980	0.929	0.929	0.967	0.760	0.831	0.943	0.831
xerces	0.802	0.805	0.811	0.860	0.583	0.666	0.786	0.660
W/T/L	—	10 / 0 / 5	9 / 1 / 5	13 / 0 / 2	12 / 0 / 3	13 / 1 / 1	13 / 0 / 2	12 / 1 / 2
AVG	0.451	0.432	0.437	0.421	0.383	0.393	0.422	0.396

The penultimate row presented the win/tie/loss counts of KMM-MCW versus other approaches and the last row showed the average *F*-measures. The best *F*-measures are in bold

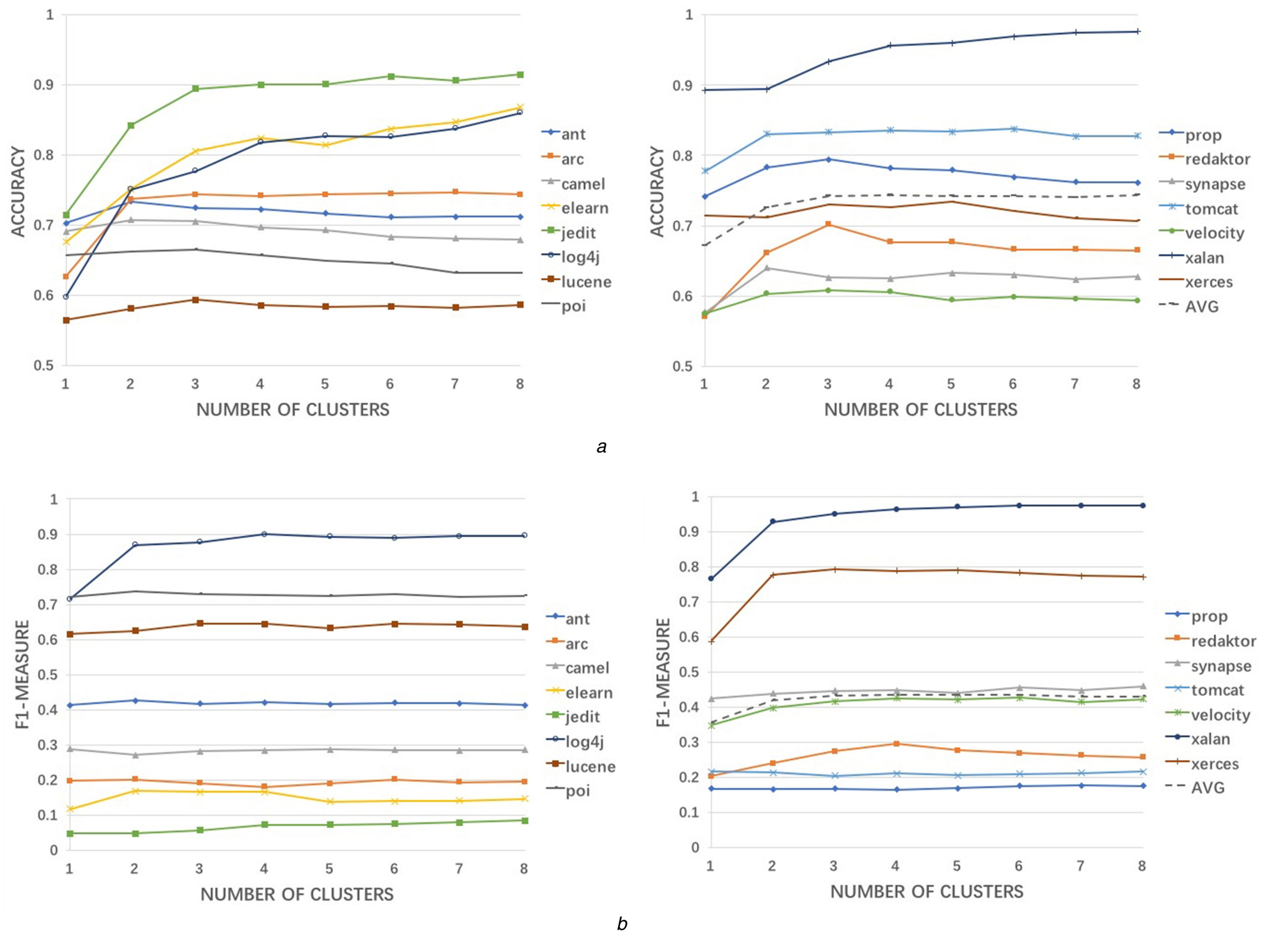


Fig. 4 *KMM-MCW performances with different numbers of components in the source project*
(a) Accuracy performances (dotted line shows AVG performance), ***(b)*** *F*-measure performances (dotted line shows AVG performance)

effectiveness of KMM-MCW. Note that, if set the number as 1, it means there is only one component in the source project, so its result represents the performance of the method that only uses KMM steps without MCW.

Fig. 4 shows the average accuracies and *F*-measures of KMM-MCW on 15 datasets with different number of clusters. It is clear that the performance of KMM-MCW was significantly improved when the number of clusters was larger than 1. When our approach only used KMM steps without MCW, the average accuracy was 0.672 and *F*-measure is 0.422. These scores were significantly lower than those of KMM-MCW with 3–8 clusters. Thus, by combining the KMM and MCW steps, the performance of CPDP could be enhanced. Moreover, the result was basically stable when the number of components was changed from 3 to 8. Thus, in practice, four components were applied in this paper because more clusters translate to higher run-time costs.

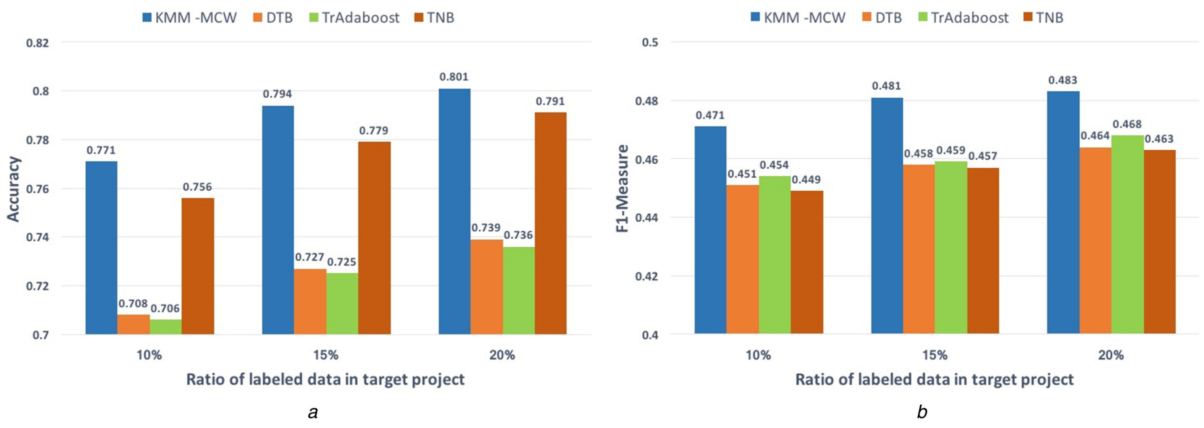


Fig. 5 *KMM-MCW, DTB, TrAdaboost and TNB method performances with 10, 15 and 20% within-project data*
(a) Accuracy performances, ***(b)*** *F*-measure performances

5.2 Impact of the percentages of within-project labelled instances for KMM-MCW, DTB, TrAdaboost and TNB

A small number of labelled data from the target project are necessary for KMM-MCW because within-project data is indispensable for initialising and adjusting the assembling weight. By default, the ratio of instances in the target data was set to be 5% in the above experiments to compare the different CPDP models. It will be important to confirm whether KMM-MCW is still better than other CPDP models (e.g. DTB, TrAdaboost and TNB) with different percentages of instances in the target data. We randomly selected 10, 15 and 20% within-project data as training data with cross-project data, and the remaining data were taken as test data. Based on the randomness setting of the experiment section, we, respectively, conducted *m*-fold cross-validation on these approaches and repeated the process 20 times.

Fig. 5 shows the results of KMM-MCW, DTB, TrAdaboost and TNB on accuracy and *F*-measure. In this figure, on average, our model continues maintaining an advantage in comparison with DTB, TrAdaboost and TNB with 10, 15 and 20% of within-project data.

5.3 Threats to validity

Threats to construct validity: include the suitability of evaluation measures [23]. In this paper, we use accuracy and *F*-measure to evaluate models' predictive power. Accuracy is a common evaluation metric for traditional transfer learning issues [26, 27]. *F*-measure is a dual assessment of both precision and recall that has been used in recent studies of CPDP [9, 13, 23]. In addition, the object-oriented metrics we used have been widely applied in existing CPDP studies [15, 23, 33]. However, it may be different from static code metrics (e.g. complexity metrics or size metrics) and change metrics in other paper. It may be a potential threat to the validity of results.

Threats to internal validity: include sample selection bias [15]. In our experiments, the process of data selection involved a degree of randomness. To avoid the threat of data selection bias, we repeated our experiments multiple times using cross-validation. In addition, for fair comparability, we used the 'rng' function of MATLAB to make the random seed orderly so that different approaches could adopt the same random data by using a given loop number.

Threats to external validity: include the general effectiveness of our model. We conducted our experiments on 15 real-world projects collected from the PROMISE repository [6]. However, various software projects have their own properties, so our approach might not be generalisable to other datasets. In addition, Jimenez *et al.* [37] provide evidence that predictions made on randomly selected training and evaluation datasets differ significantly from those that are made based on time. Our reported results do not mean that we can accurately assess the performance of our model in identifying future defects. More validation on various projects and situations should be performed in the future.

6 Conclusion and future work

It is meaningful to construct a CPDP model when there are not enough local historical data in a given target project. However, many studies have shown that CPDP is a challenging task. One of the key issues is that a large amount of irrelevant data in source project will bring about negative transfer learning. Most current solutions for this issue only consider the direct impacts of instances and features.

In this paper, we advocate that various components in the source project have different auxiliary power for target project. Based on this assumption, we proposed a novel CPDP approach named KMM-MCW. This approach mainly includes three steps: dividing multiple components, adjusting the instances in components and learning the weights of components. In the first step, we built components by using spectral clustering. In the second step, we used the KMM algorithm to reweight the instances of each component. In the third step, we learned optimisation weights and built an assembled classifier for the target project. Using 15 real-world datasets, we designed and performed experiments to compare the performance of KMM-MCW, the WPDP method and the other CPDP models. The results showed that our KMM-MCW method achieved better performance than conventional WPDP methods and up-to-date CPDP models with small ratio of within-project data.

In the future, we plan to evaluate KMM-MCW with more datasets collected from different repositories. In addition to considering more practical situations, we will pay more attention to the customised methods of imbalance learning in CPDP tasks.

7 Acknowledgments

We thank Jaechang Nam 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 and Research Fund (201802020006) and Zhongshan Produce and Research Fund(2017A1014).

8 References

[1] Menzies, T., Greenwald, J., Frank, A.: 'Data mining static code attributes to learn defect predictors', *IEEE Trans. Softw. Eng.*, 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', *Autom. Softw. Eng.*, 2010, **17**, (4), pp. 375–407
[4] D'Ambros, M., Lanza, M., Robbes, R.: 'An extensive comparison of bug prediction approaches'. Seventh IEEE Working Conf. Mining Software Repositories, Cape Town, South Africa, May 2010, pp. 31–41
[5] Lee, T., Nam, J., Han, D.G., *et al.*: 'Micro interaction metrics for defect prediction'. Proc. Nineteenth ACM SIGSOFT Symp. Thirteenth European Conf. Foundations of Software Engineering, Szeged, Hungary, September 2011, pp. 311–321
[6] 'The Promise Repository of Empirical Software Engineering Data'. Available at http://openscience.us/repo, accessed 06 June 2014

[7] Zimmermann, T., Nagappan, N., Gall, H., *et al.*: 'Cross-project defect prediction: a large scale experiment on data vs. Domain vs. Process'. Proc. Seventh Joint Meeting of the European Software Engineering Conf. the ACM SIGSOFT Symp. Foundations of Software Engineering, Amsterdam, The Netherlands, August 2009, pp. 91–100

[8] He, Z., Shu, F., Yang, Y., *et al.*: 'An investigation on the feasibility of cross-project defect prediction', *Autom. Softw. Eng.*, 2012, **19**, (2), pp. 167–199

[9] Nam, J., Pan, S.J., Kim, S.: 'Transfer defect learning'. Proc. Int. Conf. Software Engineering, San Francisco, California, May 2013, pp. 382–391

[10] Panichella, A., Oliveto, R., De Lucia, A.: 'Cross-project defect prediction models: L'union fait la force'. Software Evolution Week-IEEE Conf. Software Maintenance, Reengineering and Reverse Engineering, Antwerp, Belgium, February 2014, pp. 164–173

[11] Turhan, B., Menzies, T., Bener, A.B., *et al.*: 'On the relative value of cross-company and within-company data for defect prediction', *Empir. Softw. Eng.*, 2009, **14**, (5), pp. 540–578

[12] Yu, Q., Jiang, S., Qian, J.: 'Which is more important for cross-project defect prediction: instance or feature?'. Int. Conf. Software Analysis, Testing and Evolution, Kunming, Yunnan, November 2016, pp. 90–95

[13] Ma, Y., Luo, G., Zeng, X., *et al.*: 'Transfer learning for cross-company software defect prediction', *Inf. Softw. Technol.*, 2012, **54**, (3), pp. 248–256

[14] Peng, L., Yang, B., Chen, Y., *et al.*: 'Data gravitation based classification', *Inf. Sci.*, 2009, **179**, (6), pp. 809–819

[15] Chen, L., Fang, B., Shang, Z., *et al.*: 'Negative samples reduction in cross-company software defects prediction', *Inf. Softw. Technol.*, 2015, **62**, pp. 67–77

[16] Dai, W., Yang, Q., Xue, G.R., *et al.*: 'Boosting for transfer learning'. Proc. Twenty fourth Int. Conf. on Machine Learning, Corvallis, Oregon, June 2007, pp. 193–200

[17] Huang, J., Smola, A.J., Gretton, A., *et al.*: 'Correcting sample selection bias by unlabeled data', *Adv. Neural Inf. Process. Syst.*, 2007, **19**, pp. 601–608

[18] Peters, F., Menzies, T., Marcus, A.: 'Better cross company defect prediction'. Tenth IEEE Working Conf. Mining Software Repositories, San Francisco, California, May 2013, pp. 409–418

[19] Pan, S.J., Tsang, I.W., Kwok, J.T., *et al.*: 'Domain adaptation via transfer component analysis', *IEEE Trans. Neural Netw.*, 2011, **22**, (2), pp. 199–210

[20] Zhang, F., Mockus, A., Keivanloo, I., *et al.*: 'Towards building a universal defect prediction model'. Proc. Eleventh Working Conf. Mining Software Repositories, Hyderabad, India, May 2014, pp. 182–191

[21] Turhan, B., Misirli, A.T., Bener, A.: 'Empirical evaluation of the effects of mixed project data on learning defect predictors', *Inf. Softw. Technol.*, 2013, **55**, (6), pp. 1101–1118

[22] Yu, X., Liu, J., Fu, M., *et al.*: 'A multi-source tradaboost approach for cross-company defect prediction'. Proc. Twenty-Eighth Int. Conf. Software Engineering and Knowledge Engineering, San Francisco, California, July 2016, pp. 237–242

[23] Xia, X., Lo, D., Pan, S.J., *et al.*: 'HYDRA: massively compositional model for cross-project defect prediction', *IEEE Trans. Softw. Eng.*, 2016, **42**, (10), pp. 977–998

[24] Freund, Y., Schapire, R.E.: 'A decision-theoretic generalization of on-line learning and an application to boosting'. European Conf. Computational Learning Theory, Barcelona, Spain, March 1995, pp. 23–37

[25] Pan, S.J., Yang, Q.: 'A survey on transfer learning', *IEEE Trans. Knowl. Data Eng.*, 2010, **22**, (10), pp. 1345–1359

[26] Bollegala, D., Mu, T., Goulermas, J.Y.: 'Cross-domain sentiment classification using sentiment sensitive embeddings', *IEEE Trans. Knowl. Data Eng.*, 2016, **28**, (2), pp. 398–410

[27] Zhu, Y., Chen, Y., Lu, Z., *et al.*: 'Heterogeneous transfer learning for image classification'. The Association for the Advancement of Artificial Intelligence, San Francisco, California, August 2011

[28] Pan, S.J., Kwok, J.T., Yang, Q.: 'Transfer learning via dimensionality reduction'. The Association for the Advancement of Artificial Intelligence, Chicago, Illinois, July 2008, pp. 677–682

[29] Ng, A.Y., Jordan, M.I., Weiss, Y.: 'On spectral clustering: analysis and an algorithm'. Conf. Workshop on Neural Information Processing Systems, Vancouver, Canada, December 2001, vol. 14, (2), pp. 849–856

[30] Sun, Q., Amin, M., Yan, B., *et al.*: 'Transfer learning for bilingual content classification'. Proc. Twenty First ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, New York, USA, August 2015, pp. 2147–2156

[31] 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

[32] 'CVX: Matlab Software for Disciplined Convex Programming'. Available at http://cvxr.com/cvx, accessed 27 February 2018

[33] Jureczko, M., Madeyski, L.: 'Towards identifying software project clusters with regard to defect prediction'. Proc. Sixth Int. Conf. Predictive Models in Software Engineering, New York, USA, September 2010, p. 9

[34] Ryu, D., Choi, O., Baik, J.: 'Value-cognitive boosting with a support vector machine for cross-project defect prediction', *Empir. Softw. Eng.*, 2016, **21**, (1), pp. 43–71

[35] Yu, X., Zhou, M., Chen, X., *et al.*: 'Using class imbalance learning for cross-company defect prediction'. The 29th Int. Conf. Software Engineering and Knowledge Engineering, Pittsburgh, USA, July 2017, pp. 117–122

[36] Chawla, N.V., Bowyer, K.W., Hall, L.O., *et al.*: 'SMOTE: synthetic minority over-sampling technique', *J. Artif. Intell. Res.*, 2011, **16**, (1), pp. 321–357

[37] Jimenez, M., Papadakis, M., Traon, Y.L.: 'Vulnerability prediction models: a case study on the linux Kernel'. The Int. Working Conf. Source Code Analysis and Manipulation, Raleigh, USA, October 2016, pp. 1–10