Mathematical Biosciences and Engineering

MBE, 17(2): 1020–1040.
DOI: 10.3934/mbe.2020054
Received: 03 May 2019
Accepted: 08 October 2019
Published: 11 November 2019

http://www.aimspress.com/journal/MBE

Research article

Heterogeneous cross-project defect prediction with multiple source projects based on transfer learning

Xinglong Yin, Lei Liu, Huaxiao Liu* and Qi Wu

Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, College of Computer Science and Technology, Jilin University, Changchun 130012, China

*** Correspondence:** liuhuaxiao@jlu.edu.cn; Tel: +86043185166810; Fax: +86043185166810.

Abstract: Cross-project defect prediction (CPDP) aims to predict the defect proneness of target project with the defect data of source project. Existing CPDP methods are based on the assumption that source and target projects should have the same metrics. Heterogeneous cross-project defect prediction (HCPDP) builds a prediction model using heterogeneous source and target projects. Existing HCPDP methods just focus on one source project or multiple source projects with the same metrics. These methods limit the scope of getting the source project. In this paper, we propose Heterogeneous Defect Prediction with Multiple source projects (HDPM) which can use multiple heterogeneous source projects for defect prediction. HDPM based on transfer learning which can learn knowledge from one domain and use it to help with other domain. HDPM constructs a projective matrix between heterogeneous source and target projects to make the distributions of source and target projects similar. We conduct experiments on 14 projects from four public datasets and the results show that HDPM can achieve better performance compared with existing CPDP methods, and outperforms or is comparable to within-project defect prediction method. The use of multiple heterogeneous source projects for defect prediction can effectively extend the data acquisition range of defect prediction and make software defect prediction better applied to software engineering.

Keywords: defect prediction; heterogeneous metrics; multiple heterogeneous source projects; transfer learning

1. Introduction

Software defect prediction (SDP) is one of the most active research areas in software engineering

which predicts defect proneness of new modules using historical defect data [1–4]. If software developers can perform defect prediction before software release, it can effectively help in reducing cost of development, shortening production cycle and improving software quality. So far, a number of software defect prediction methods are based on machine learning technique to build a prediction model using historical defect data in software dataset [1,5–11]. The defect data usually consists of software metrics and defect labels. Here, software metrics commonly used for defect prediction are complexity metrics (such as lines of code, Halstead metrics [12] and McCabe metrics [13]), object-oriented metrics [14] and process metrics [15], and defect labels are used to record the defect proneness of the software modules.

Early defect prediction methods predict defect proneness of a new project using historical defect data from the same project, which is called within-project defect prediction (WPDP) [16–18]. However, WPDP method is not applicable for a new project or a project that only has limited historical defect data due to lacking of sufficient data to build an accurate prediction model.

To address the limitation of WPDP method, researchers have proposed cross-project defect prediction (CPDP) [1,5–9], which predicts defects for a new project using historical defect data from other projects. However, most existing CPDP methods are based on the assumption that source and target projects should have the same software metrics, while the metrics from different projects are usually heterogeneous in practice problems. For example, Table 1 shows the public defect datasets that are widely used in defect prediction including SOFTLAB, NASA, AEEEM and ReLink, which have 29, 37, 61 and 26 metrics respectively. Although there are some CPDP methods can handle heterogeneous source and target projects, they only use the common metrics which less exist between two projects and it is not sufficient to build a prediction model with such small number of metrics. Such as in Table 2, NASA and ReLink datasets only have 3 common metrics, and NASA and AEEEM have no common metrics. Therefore, the existing CPDP methods are not applicable for heterogeneous source and target projects.

Recently, there have researches about heterogeneous cross-project defect prediction (HCPDP) [10,11], which builds a prediction model using source and target projects with heterogeneous software metrics. Existing HCPDP methods predict defects using one source project or multiple source projects with the same metrics. Studies have shown that the performance of using multiple source projects for defect prediction is better than using only one source project [11]. But finding multiple projects with exactly the same metrics as source projects is a challenge. In this paper, we propose Heterogeneous Defect Prediction with Multiple source projects (HDPM) which can use multiple heterogeneous source projects for defect prediction and it can be an effective solution for the above issues. The main idea of HDPM is to construct a projective matrix between heterogeneous source and target projects which converts source project to target project space, and then use the classifier to predict the defect proneness of target project.

We use transfer learning to obtain projective matrix from multiple projects. Transfer learning is a method that allowed the domains, tasks, and distributions used in the training data and test data to be different [6]. When predicting defects in a project, there may be only a small scale of accessible data, and even no relevant data especially for a new project. In such condition, the predication would become a difficult task and many existing methods could not achieve good results. Transfer learning can complete a classification task by using heterogeneous data source in a different feature space or follow a different data distribution [19], so it can use the data from multiple heterogeneous source projects to predict defects in the target project. Furthermore, there is a phenomenon in software

development that many projects copy the same project [20], which means they may share same library or documents. This makes transfer learning can be more suitable for applying to the defects predication of these projects. Specifically, the transfer learning provides a solution for converting source project to target project space, it can convert different features of heterogeneous projects to the vectors in the same length, and then those vectors can be used as input of the classifier for the prediction. Compared to existing CPDP methods, our method can utilize more data so it works well in defect prediction.

To evaluate the effectiveness of HDPM, we perform an experiment using four public datasets (SOFTLAB [21], NASA [21,22], AEEEM [23] and ReLink [24]) that contain 14 projects in total. And we evaluate our approach against CPDP methods including TNB [6], NN-filter [5], TCA+ [1], CCA+ [11] and WPDP method. The results show that HDPM can achieve better performance of defect prediction compared with existing CPDP methods, and outperforms or is comparable to WPDP. And the performance of defect prediction for using multiple source projects outperforms using one source project.

In this paper, we answer the following two research questions:

RQ1: Is our approach HDPM helpful for heterogeneous cross-project defect prediction?

RQ2: Whether using multiple heterogeneous source projects is applicable to software defect prediction?

The contributions of this paper are concluded as the following two points:

1) We for the first time introduce the transfer learning method Multiple Outlook MAPping algorithm (MOMAP) into defect prediction for making the distribution of source and target projects similar, and it makes full use of all the metrics of source and target projects.

2) We give a method to predict defects for one source project, and for the first time use multiple heterogeneous source projects for defect prediction to extend the data acquisition range based on this method, making software defect prediction can be better applied to software engineering.

The remainder of this paper is organized as follows: Section 2 reviews the related work of defect prediction. Section 3 introduces our approach HDPM. Section 4 describes our experimental setup and the results. The conclusion and future work are presented in Section 5.

Table 1. The number of metrics of four public datasets.

Dataset	SOFTLAB	NASA	AEEEM	ReLink
Number of metrics	29	38	61	26

Table 2. The number of common metrics between different datasets.

Dataset A∩Dataset B	SOFTLAB ∩NASA	SOFTLAB ∩AEEEM	SOFTLAB ∩ReLink
Number of common metrics	28	0	3
Dataset A∩Dataset B	NASA∩ AEEEM	NASA∩ ReLink	AEEEM∩ ReLink
Number of common metrics	0	3	0

2. Related work

Software defect prediction is an active area in software engineering. There have been a number

of researches on defect prediction [1,5–11,16–18,25–29]. Most of these methods use machine learning technique and the same project defect data which is called within-project defect prediction (WPDP). For instance, Flish et al. [16] used support vector machines (SVM) to predict defect-prone software modules at four NASA datasets. Andreou et al. [17] applied a machine learning technique called fuzzy decision trees to acquire accurate and reliable costs that involved in software development. Bettenburg et al. [18] designed MARS that has local consideration and is a hybrid between global and local models to improve the performance of defect prediction. However, there are many new projects that have not sufficient historical defect data to build a prediction model.

Some researchers attempt to use unsupervised defect prediction that without requiring source data. Zhong et al. [25] proposed a clustering and expert based method that applying k-means and Neural-Gas in defect prediction. Bishnu et al. [26] applied a Quad Tree to find the initial cluster centers to k-means algorithm for predicting faults in software modules. Abaei et al. [27] proposed self-organizing map (SOM) to build a prediction model that uses thresholds but not experts for labeling modules. Nam et al. [28] proposed novel approaches called CLA and CLAMI using the magnitude of metric values to label an unlabeled dataset. Zhang et al. [29] built a prediction model using a connectivity-based unsupervised classifier via spectral clustering. Although unsupervised defect prediction is the best method for defect prediction theoretically for it only uses own data to predict defects, there exist a limited number of researches on it. One important reason is the ability of unsupervised learning methods usually underperform supervised ones resulting in the prediction power of unsupervised classifiers is relatively low.

Other researchers propose a number of cross-project defect prediction (CPDP) or cross-company defect prediction (CCDP) methods about supervised ones [1,5–9]. Turban et al. [5] proposed the nearest neighbor (NN) filter which builds a prediction model by source modules that remove irrelevancies with target modules to avoid the high false-positive rate. Ma et al. [6] proposed Transfer Naïve Bayes (TNB) which estimates the distribution of target data and transfers cross-company data information into the weights of source data, and using these weighted data to build a prediction model. Canfora et al. [7] proposed a novel multi-objective defect prediction approach using genetic algorithm based on a multi-objective logistic regression model to achieve a compromise between effectiveness and cost. Similar to the work by Turban, Peter et al. [8] proposed Peters filter that selects source data via the structure of training data set and test data. Nam et al. [1] applied a state-of-the-art transfer learning approach called transfer component analysis (TCA) to make feature distribution similar between source and target projects, and extended TCA to TCA+ that choosing different normalization for preprocessing to achieve the best performance. Chen et al. [9] proposed Double Transfer Boosting (DTB) which reshapes the distribution of cross-company data to fit within-company data using data gravitation method and eliminates negative instances in CC data using labeled WC data. However, these CPDP or CCDP methods are based on the assumption that source and target data should have the same software metrics. When it is hard to find a project that has the same metric with target project, the existing CPDP or CCDP methods cannot be used in the setting. But there are a number of projects that have heterogeneous metrics in dataset, so finding a new method for heterogeneous cross-project defect prediction becomes urgent.

So far, there are researches about heterogeneous cross-project defect prediction (HCPDP) [10,11]. Nam et al. [10] proposed heterogeneous defect prediction (HDP) which conducts metric selection and metric matching to predict defects across projects with heterogeneous metrics set. Jing et al. [11]

proposed a unified metric representation (UMR) and applied canonical correlation analysis (CCA) to make the data distribution of source and target similar. The main differences between our approach and existing HCPDP methods are as follows. First, our approach can use multiple heterogeneous source projects to predict defect which is the first time used in software defect prediction. Second, our approach uses all the metrics of source and target projects for defect prediction that maintains the whole information of datasets.

3. Approach

In this section, we describe details on our approach HDPM, which includes one source project HDPM and multiple heterogeneous source projects HDPM. Compared to predict defects using one source project, HDPM use multiple heterogeneous source projects to predict a target project. In our method, the multiple source projects can be completely different, and the target project not could be related to the source projects.

3.1. Problem formulation and overview

Suppose there are Q labeled source projects and an unlabeled target project, the software metrics of source and target projects are different, that is, source and target projects are heterogeneous. The purpose of defect prediction is to use source projects to predict defect proneness of target project. Let $\left\{X_s^{(k)}\right\}_{k=1}^{Q}$ denote Q source projects where $X_s^{(k)} = \left\{x_s^1, x_s^2, \cdots, x_s^{N_{sk}}\right\}$, and $X_t = \left\{x_t^1, x_t^2, \cdots, x_t^{N_t}\right\}$ denote target project. Here, x_s^i denotes the i^{th} module in $X_s^{(k)}$ and x_t^i denotes the i^{th} module in X_t, N_{sk} and N_t are the numbers of modules in $X_s^{(k)}$ and X_t. A module in source project can be represented as $x_s^i = \left\{m_s^{i1}, m_s^{i2}, \cdots, m_s^{id_s}\right\}$ and a module in target project can be represented as $x_t^i = \left\{m_t^{i1}, m_t^{i2}, \cdots, m_t^{id_t}\right\}$. Here, m_s^{ij} and m_t^{ij} respectively represent j^{th} metric value of x_s^i and x_t^i, d_s and d_t are the number of metrics in source and target projects. In general, the metrics that used in source and target projects are different, namely $d_s \neq d_t$. $Y_s^{(k)} = \left\{y_s^1, y_s^2, \cdots, y_s^{N_{sk}}\right\}$ represents defect proneness of k^{th} source project, $y_s^i = 1$ when x_s^i has defects and $y_s^i = 0$ when x_s^i is clean. The defect proneness of target project $Y_t = \left\{y_t^1, y_t^2, \cdots, y_t^{N_t}\right\}$ is obtained by our approach HDPM.

The target of HDPM is using transfer learning to obtain projective matrix for predicting defects. Figure 1 shows the three main phases of HDPM: (1) Firstly, features are extracted from the projects, and the data from source projects are preprocessed. (2) Then, we obtain the projective matrix by using the method of PCA and SVD, so that the heterogeneous source projects can be converted to target project space. (3) Finally, we obtain the normalized vectors of projects based on projective matrix, and train a classifier to predict the defect proneness of target project.

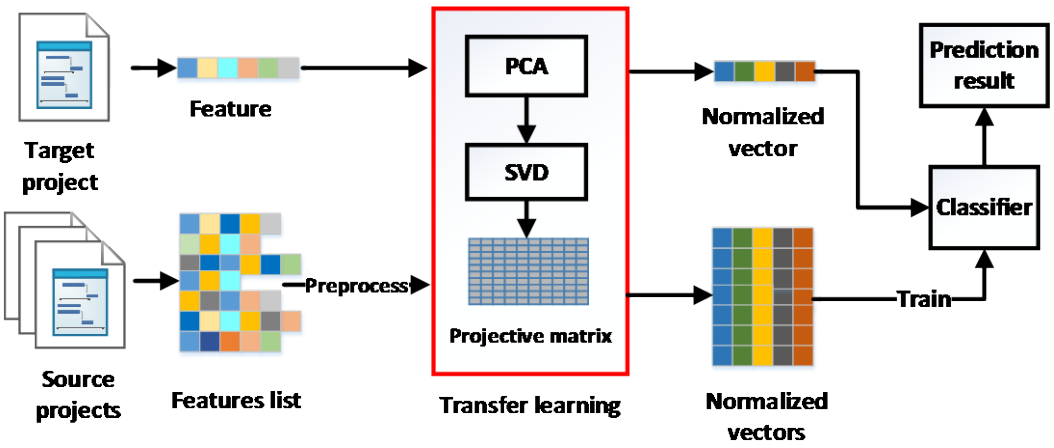

Figure 1. The overview of approach.

3.2. Preprocessing

Typically, the defect data is class-unbalanced data that the number of defective modules in a project is much less than the number of clean modules. For example, the rate of defects of PC1 in NASA is only 8.65%. Mahmood et al. [3] proposed the performance of defect prediction is usually low when the data is unbalanced. That is, as the data becomes more and more balanced, the performance of defect prediction increases gradually. Therefore, we use random multi-sampling to preprocess source project, by increasing the number of defective modules to balance the data, so that all the information of source project can be retained.

Since the values of software metrics vary between different projects, it is necessary to use normalization to make the metric values of source and target projects in the same range. Normalization gives all metrics of data an equal weight and is useful for classification algorithms [30]. Similarly, Graf et al. [31] also confirmed that normalization can improve the performance of defect prediction. Therefore, we employ min-max normalization method to preprocess source and target projects which makes metric values range from zero to one. In other words, the minimum and maximum values of source and target projects are transformed into zero and one, respectively.

Algorithm 1 HDPM Approach

Input: Source project X_s, target project X_t and source labels Y_s.

Output: Target labels Y_t.

1. Process class-unbalanced data of source project.
2. Use the min-max normalization to preprocess X_s and X_t.
3. Translate the means of each metric of source and target projects to zero, $\widetilde{X_s} = X_s - \mu_s$ and $\widetilde{X_t} = X_t - \mu_t$.
4. Use PCA to construct the utilization matrices D_s and D_t.
5. Use SVD to decompose matrix $D_s D_t^T = USV^T$.
6. Obtain the projective matrix $R = VZU^T$.
7. Based on the obtained $\widehat{X_s} = R\widetilde{X_s} + \mu_t$ and X_t, using the classifier to predict defects of target project and obtaining the prediction result Y_t.

3.3. One source project HDPM

After preprocessing, we introduce transfer learning method Multiple Outlook MAPping algorithm (MOMAP) [32] into defect prediction to make the distributions of source and target projects similar. By using this method, our approach constructs a projective matrix between heterogeneous source and target projects, in this way the source projects can be converted into target project space and the maximum correlations between them are established.

The mapping of source and target projects can be obtained by translation and rotation. The algorithm of our approach is described in Algorithm 1.

First, we remove the mean of each metric in source and target projects to translate the means to zero. Assume the means of each metric in source and target data construct the vectors μ_s and μ_t respectively. The following two equations are used to obtain a new representation of source and target projects after removing the means of each metrics.

$$\widetilde{X_s} = X_s - \mu_s \tag{1}$$

$$\widetilde{X_t} = X_t - \mu_t \tag{2}$$

Next, we obtain the projective matrix R using the principal direction of source and target projects, the rotation matching of source and target projects can be done by solving the following optimization problem:

$$\begin{cases} \arg\min_R \|RD_s - D_t\|_F^2 \\ s.t.\ R^T R = I \end{cases} \tag{3}$$

where $\|\cdot\|_F$ is the Frobenius norm that is used to represent distance, $R \in R^{d_t \times d_s}$ is the projective matrix, $D_s \in R^{d_s \times h}$ and $D_t \in R^{d_t \times h}$ are the utilization matrices that are formed by the h principal directions of source and target projects, here, $\mathrm{h} \leq \min(d_s, d_t)$.

After rotation, source project processed by the projective matrix is added with the means of target project to make source project transform to target project space finally. Then we can use the converted source project and target project to train any classifier, such as k-nearest neighbors, SVM and Naïve Bayes, for defect prediction.

In Eq (3), we use principal directions of source and target projects to obtain the projective matrix. The principal directions can be obtained by principal component analysis (PCA). The main idea of PCA is to recombine the original metrics and form a new set of unrelated metrics. The newly formed metrics can represent the main features of the original data and can better distinguish different types of modules. Therefore, it is reasonable to obtain the projective matrix through establishing a mapping relationship between the principal directions of source and target projects. In this paper, we use PCA to extract the first h principal directions of source and target projects. In addition, we changed the value of h varies between 2 and 10 in the experiment to get better results [33]. The principal directions of source and target projects can be obtained by the following steps: We first perform singular value decomposition (SVD) on source and target projects separately, and then select h eigenvectors corresponding to the h largest eigenvalues as the first h principal directions.

The expansion of optimization problem in Eq (3) is $\|RD_s - D_t\|_F^2 = tr(D_s^T R^T R D_s - 2D_t^T R D_s + D_t^T D_t)$, here, $D_s^T R^T R D_s$ and $D_t^T D_t$ are constants. So Eq (3) can be transformed to the following problem:

$$\begin{cases} \arg\max_R tr(D_t^T R D_s) \\ s.t.\ R^T R = I \end{cases} \tag{4}$$

where D_s and D_t consist of h principal directions, that is $D_s = [v_s^1, \cdots, v_s^h]$ and $D_t = [v_t^1, \cdots, v_t^h]$ where v_s^l and v_t^l $(l = 1, \cdots, h)$ are the l^{th} principal directions of source and target projects, respectively. Equation (4) can be intuitively transformed to:

$$\begin{cases} \arg\max_R \sum_{l=1}^h v_t^{lT} R v_s^l \\ s.t.\ R^T R = I \end{cases} \tag{5}$$

Equation (3) is equivalent to minimization of the first h principal directions angles of source and target projects, and Eq (5) in turn implies maximization of the sum of inner products between the principal directions of source and target projects. For the solutions of Eq (4) we refer Procrustes Analysis technique from [34]. We refer Procrustes Analysis technique for solutions of Eq (4) [35]. Since $\text{tr}(D_t^T R D_s) = tr(R D_s D_t^T)$, Eq (4) is equivalent to:

$$\begin{cases} arg\max_R tr(R D_s D_t^T) \\ s.t.\ R^T R = I \end{cases} \tag{6}$$

Let USV^T be the singular value decomposition (SVD) of $D_s D_t^T$. Define $Z = V^T R U$, then,

$$\text{tr}(R D_s D_t^T) = \text{tr}(RUSV^T) = \text{tr}(ZS) = \sum_{k=1}^m z_{kk}\, \sigma_k \le \sum_{k=1}^m \sigma_k \tag{7}$$

where σ_k is the k^{th} singular value of $D_s D_t^T$.

Since $tr(R D_s D_t^T)$ can reach its max value when $Z = I$, we can obtain the projective matrix $R = VZU^T$.

3.4. Multiple heterogeneous source projects HDPM

In this sub-section, we present the algorithm to use multiple source projects $\left\{X_s^{(k)}\right\}_{k=1}^Q$ for defect prediction, where multiple source projects can have heterogeneous metrics. We transform all the source projects to target project space. Similar to Algorithm 1, we first process class-unbalanced data of source projects and use min-max normalization to preprocess $\left\{X_s^{(k)}\right\}_{k=1}^Q$ and X_t. Next, we translate the means of multiple source projects and target project to zero, and then construct the utilization matrices $\left\{D_s^{(k)}\right\}_{k=1}^Q$ of source projects and D_t of target project. By the Eq (8), all source projects are converted to target project space.

$$\begin{cases} \arg\min_{\left\{R^{(k)}\right\}_{k=1}^Q} \sum_{k=1}^Q \left\| R^{(k)} D_s^{(k)} - D_t \right\|_F^2 \\ s.t.\ R^{(k)T} R^{(k)} = I \end{cases} \tag{8}$$

The method of getting projective matrix $R^{(k)}$ is similar to that of R in Algorithm 1. Let $U^{(k)} S^{(k)} V^{(k)T}$ be the singular value decomposition (SVD) of $D_s^{(k)} D_t^T$, and the projective matrix of k^{th} source project is $R^{(k)} = V^{(k)} U^{(k)T}$. The final k^{th} source project can be represented as:

$$X_s^{(k)} = (X_s^{(k)} - \mu_s^{(k)})R^{(k)T} + \mu_t \tag{9}$$

Finally, we combine all the source projects and use the classifier to build a prediction model to predict defect proneness of target project.

4. Experiments

In this section, we present the experimental setup in detail to evaluate the performance of our approach. First, we introduce the benchmark datasets and evaluation methods which are commonly used in defect prediction. Then, we perform experiments of HDPM with one source project, followed by the experiment of HDPM with multiple heterogeneous source projects. Last, we would discuss the value of parameters that are used in the paper.

4.1. Benchmark datasets

In our experiments, we examine 14 projects containing 7511 modules from four public datasets that are commonly used in defect prediction including SOFTLAB, NASA, AEEEM and ReLink. Table 3 shows the datasets we used in experiment, and a brief description on each dataset is presented as follows.

Table 3. Details of dataset used in the experiment.

Dataset	Project	Number of metrics	Number of total modules	Percentage of defective modules
SOFTLAB	AR3	29	63	12.7%
	AR4	29	107	18.69%
	AR5	29	36	22.22%
NASA	CM1	37	327	12.84%
	MW1	37	253	10.67%
	PC1	37	705	8.65%
AEEEM	EQ	61	324	39.81%
	JDT	61	997	20.66%
	LC	61	691	9.26%
	ML	61	1862	13.16%
	PDE	61	1497	13.96%
ReLink	Apache	26	194	50.52%
	Safe	26	56	39.29%
	ZXing	26	399	29.57%

The SOFTLAB and NASA datasets were collected from a Turkish software company and numerous NASA contractors, respectively [21]. For the SOFTLAB dataset, we use three projects AR3, AR4 and AR5 which are embedded controller software in the PROMISE repository. We use three NASA projects CM1, MW1 and PC1 which have the same metrics in the PROMISE repository. Shepperd et al. [22] find that the NASA dataset contains conflict and inconsistent cases, therefore, we use NASA datasets cleaned by them in this study. There are 28 common metrics between NASA

and SOFTLAB which are Halstead and McCabe's cyclomatic complexity metrics.

The AEEEM dataset was collected by D'Ambros et al. [23], and contains 61 metrics that are the most number of metrics in public defect datasets. The whole metrics include source code metrics, previous-defect metrics, entropy-of-change metrics, entropy-of-source-code metrics and churn-of-source-code metrics. The AEEEM dataset has no common metrics with the other three datasets.

The ReLink dataset was collected by Wu et al. [24] to improve the performance of defect prediction by manually verifying and correcting the dataset to increase its quality. The three projects that are used in the paper have 26 code complexity metrics which are widely used in defect prediction. The ReLink dataset has only three common metrics with NASA and SOFTLAB including lines of code, lines of blank and lines of comment.

4.2. Performance measure

To evaluate the performance of defect prediction, we use recall (pd), false-positive (pf) and F-measure as evaluation measures because they are commonly used in defect prediction. The measures are defined as follow: the number of defective modules that are predicted as defective (true positive, TP); the number of clean modules that are predicted as defective (false positive, FP); the number of defective modules that are predicted as clean (false negative, FN); the number of clean modules that are predicted as clean (true negative, TN). The four kinds of defect prediction outcomes can be more clearly showed in Table 4.

Table 4. Four kinds of defect prediction outcomes.

	Predict as defective	Predict as clean
Defective modules	TP	FN
Clean modules	FP	TN

Recall: The ratio of modules that are correctly classified as defective to those defective modules.

$$recall = TP/(TP + FN)$$

False-positive: The ratio of modules that are wrongly classified as defective to those clean modules.

$$false\_positive = FP/(FP\ + TN)$$

Precision: The ratio of modules that are correctly classified as defective to those classified as defective.

$$precision = TP/(TP + FP)$$

F-measure: A harmonic mean of precision and recall.

$$\text{F\_measure} = \frac{2 \times precision \times recall}{precision + recall}$$

A good prediction model intends to find out defective modules as much as possible and few wrongly prediction. So a good prediction model desires to achieve high value of recall and precision, and low value of false-positive. However, there is a trade-off between recall and precision and it is difficult to compare the performance of several prediction models by using only recall or precision. Therefore, we choose F-measure which is a harmonic mean of recall and precision. The recall, false-positive and F-measure evaluation measures range from 0 to 1. Obviously, an ideal defect prediction model should have high values of recall and F-measure, and low value of false-positive.

4.3. Experiment design

To evaluate the effectiveness of our approach, we compare HDPM with two cross-company defect prediction methods including TNB [6] and NN-filter [5], two cross-project defect prediction methods TCA+ [1] and CCA+ [11], and within-project defect prediction method WPDP. For TCA+, we use the source code provided by the author. We carefully re-implement TNB, NN-filter and CCA+ according to their papers. In these compared methods, TNB, NN-filter and TCA+ methods require the source and target projects should have the same metrics, so we use the common metrics in heterogeneous source and target projects. NN-filter attempts to select suitable training modules to construct predictors, and we select 5 nearest neighbors for each target module to construct the training set. We use K-Nearest Neighbor (KNN) as classifier in our approach, where Euclidean distance and the top 5 nearest neighbors are set as parameters of KNN.

We design the following two experiments to evaluate our approach:

1) One-to-one HDPM. We use all modules in one project as source project to conduct heterogeneous cross-project defect prediction. The source and target projects are from different datasets. For example, EQ→AR4, CM1→JDT, PC1→Apache, etc. The left of "→" represents the source project and the right of "→" represents the target project.

2) Multiple-to-one HDPM. We use all modules in multiple projects which are from different datasets as multiple heterogeneous source projects. We use five-fold crossvalidation method to evaluate our approach: Eighty percent data is used as source projects, while other 20% is used as target projects for evaluating the performance. The source and target projects are also heterogeneous. For example, {CM1, Apache, EQ}→AR3, {AR3, Apache, JDT}→CM1, etc.

Since our approach involves a degree of randomness that we process class-unbalanced data of source project in preprocessing, we run HDPM 100 times of each pair of source and target projects and add up these experiment results. We set a parameter *threshold* that will introduce in 4.4.1, if the number of experiments that predict a module to be defective is larger than the threshold, the module is predicted to be defective eventually. In 4.4.1, the result shows that our approach can get better performance of defect prediction when the value of *threshold* varies from 30 to 45. Therefore, in the following experiments, we choose *threshold* to 30 which means if more than 30 percent of the experiments predict a module to be defective, the module is predicted to be defective eventually.

4.4. One-to-one HDPM

For one-to-one HDPM, we choose one project in a dataset as source project and one project in another dataset as target project, that is, source and target projects are heterogeneous. Since existing CCDP or CPDP methods use common metrics in heterogeneous source and target projects, HDPM can compare with these methods including TNB, NN-filter and TCA+ method when there are common metrics in source and target projects. For source and target projects with no common metrics, such as AEEEM and the other three datasets have no common metrics, HDPM can only compare with CCA+ method.

Table 5 shows the Pd and Pf values of our approach and other compared methods when source and target projects exist common metrics. Table 7 shows the Pd and Pf values of our approach and CCA+ method when source and target projects exist no common metrics. In these two tables, the numbers which are presented with boldface indicate the best results in these methods. The Pd and Pf

values in these two tables are the average values calculated from 100 repeated runs. The average Pd and Pf values of HDPM are 0.65 and 0.07 in Table 5, and the average Pd and Pf values of HDPM are 0.61 and 0.09 in Table 7. From Tables 5 and 7, we can see that HDPM can obtain better Pd and Pf values compared with TNB, NN-filter and TCA+ in most prediction scenes, and is similar to CCA+. The average values of Pd and Pf are similar in Tables 5 and 7 whether there are common metrics in source and target projects.

Table 5. Pd and Pf values of one-to-one HDPM with common metrics.

Source→ Target	Measure	HDPM	TNB	NN-filter	TCA+	CCA+	Source→ Target	Measure	HDPM	TNB	NN-filter	TCA+	CCA+
CM1→	Pd	0.72	**0.74**	0.58	0.60	0.70	Apache→	Pd	**0.59**	0.26	0.14	0.40	0.44
AR4	Pf	**0.01**	0.65	0.09	0.32	**0.01**	MW1	Pf	**0.06**	0.20	0.10	0.19	0.21
AR4→	Pd	0.70	0.76	0.15	0.59	**0.78**	PC1→	Pd	0.78	0.61	0.54	0.13	**0.83**
CM1	Pf	**0.02**	0.52	**0.02**	0.40	0.03	Safe	Pf	**0.01**	0.55	0.20	0.08	0.25
PC1→	Pd	**0.68**	0.50	0.60	0.30	0.60	Safe→	Pd	0.59	0.70	0.12	0.54	**0.72**
AR4	Pf	0.02	0.36	0.23	0.36	**0.00**	PC1	Pf	0.27	0.47	0.10	0.38	**0.08**
AR4→	Pd	0.66	0.65	0.40	0.47	**0.85**	CM1→	Pd	0.64	0.52	0.60	0.60	**0.68**
PC1	Pf	**0.00**	0.21	0.15	0.23	0.04	Apache	Pf	**0.02**	0.41	0.28	0.35	0.10
MW1→	Pd	0.55	0.50	**0.75**	0.32	0.60	AR3→	Pd	0.71	0.53	0.75	0.17	**0.92**
AR4	Pf	0.15	0.38	0.18	**0.10**	0.02	Apache	Pf	**0.04**	0.47	0.62	0.07	0.50
PC1→	Pd	0.73	**0.75**	**0.75**	0.37	**0.75**	Apache→	Pd	**0.75**	0.12	0.25	0.37	**0.75**
AR3	Pf	0.02	0.21	0.10	0.16	**0.01**	AR3	Pf	**0.02**	0.10	0.17	0.14	0.30
PC1→	Pd	0.60	0.50	**1.00**	0.37	0.62	AR4→	Pd	0.42	0.23	0.11	0.47	**0.48**
AR5	Pf	0.03	0.37	0.25	0.03	**0.00**	ZXing	Pf	**0.05**	0.21	0.16	0.36	0.23
CM1→	Pd	**0.55**	0.42	0.50	0.50	0.46	ZXing→	Pd	0.68	0.27	0.26	0.20	**0.75**
ZXing	Pf	**0.01**	0.31	0.28	0.25	0.25	AR4	Pf	0.15	0.24	**0.07**	0.13	0.24
ZXing→	Pd	**0.64**	0.45	0.45	0.55	0.56	AR5→	Pd	**0.72**	0.42	0.53	0.37	0.65
CM1	Pf	**0.01**	0.24	0.13	0.21	0.25	Safe	Pf	0.29	0.30	0.42	**0.19**	0.24
MW1→	Pd	0.63	0.42	0.35	0.38	**0.72**	Safe→	Pd	0.64	0.22	0.35	0.47	**0.66**
Apache	Pf	**0.10**	0.15	0.22	0.26	0.35	AR5	Pf	**0.05**	0.12	0.19	0.16	0.18
Average	Pd	0.65	0.48	0.46	0.41	**0.67**	-	-	-	-	-	-	-
	Pf	**0.07**	0.32	0.20	0.22	0.16	-	-	-	-	-	-	-

Table 6 shows the F-measure values of HDPM, TNB, NN-filter, TCA+ and CCA+ method when source and target projects exist common metrics, and Table 8 shows the F-measure values when source and target projects exist no common metrics. Both of these tables are the average values calculated 100 repeated runs. The average F-measure value of HDPM is 0.63 in Table 6, and the average F-measure values of HDPM is 0.61 in Table 8. From Tables 6 and 8, we can see that HDPM can obtain better F-measure values in most cases compared with existing CPDP methods. The reason is that our approach uses all the metrics for defect prediction rather than just common metrics, so HDPM can retain all the information of source and target projects. When comparing Table 6 with Table 8, the performance of our approach is not affected significantly whether the source and target projects have common metrics. The reason is that we use principle directions of source and target projects to construct the projective matrix that the principle directions recombine all the metrics to

make them irrelevant and can represent the main information of source and target projects, so HDPM is not affected by the number of common metrics. Above all, we can conclude that HDPM is helpful for HCPDP and can achieve better performance for one-to-one HDPM.

We perform the Wilcoxon rank-sum test [35] at a confidence level of 95% to statistically analyze the F-measure results in Tables 4 and 6, and the p-values are shown in the last row of Tables 4 and 6. As the p-values are all below 0.05, it illustrates that HDPM can significantly improve the existing CPDP methods.

Table 6. F-measure values of one-to-one HDPM with common metrics.

Source→Target	HDPM	TNB	NN-filter	TCA+	CCA+	Source→Target	HDPM	TNB	NN-filter	TCA+	CCA+
CM1→AR4	0.67	0.33	0.60	0.40	**0.80**	Apache→MW1	**0.57**	0.34	0.27	0.43	0.27
AR4→CM1	0.56	0.29	0.24	0.28	**0.78**	PC1→Safe	**0.80**	0.48	0.60	0.22	0.73
PC1→AR4	0.70	0.33	0.55	0.37	**0.75**	Safe→PC1	0.50	0.21	0.11	0.19	**0.56**
AR4→PC1	0.58	0.34	0.27	0.23	**0.74**	CM1→Apache	0.70	0.54	0.65	0.62	**0.76**
MW1→AR4	0.48	0.32	0.59	0.38	**0.70**	AR3→Apache	**0.53**	0.53	0.63	0.28	0.32
PC1→AR3	0.68	0.31	0.61	0.46	**0.80**	Apache→AR3	**0.78**	0.14	0.21	0.31	0.38
PC1→AR5	0.62	0.36	0.50	0.51	**0.76**	AR4→ZXing	**0.57**	0.26	0.15	0.40	0.47
CM1→ZXing	**0.61**	0.49	0.44	0.53	0.54	ZXing→AR4	**0.76**	0.23	0.33	0.22	0.52
Zxing→CM1	**0.69**	0.31	0.30	0.40	0.39	AR5→Safe	0.46	0.33	**0.47**	0.31	**0.47**
MW1→Apache	**0.70**	0.30	0.51	0.33	0.69	Safe→AR5	**0.70**	0.20	0.38	0.32	0.55
Average	**0.63**	0.33	0.42	0.36	0.60	P-values	-	$3.3e^{-7}$	$2.2e^{-4}$	$7.2e^{-7}$	0.0017

4.5. Multiple-to-one HDPM

For multiple-to-one HDPM, the experiments were conducted on a 32-bit window machine with Intel Core2 Duo E8400 3.0 GHz and 3.0 GB memory. We run the code on MATLAB. We select one project from SOFTLAB, NASA, AEEEM and ReLink datasets, respectively, and three of them as heterogeneous source projects and the other project as target project. Since the source projects are heterogeneous among themselves and they are also heterogeneous with target project, there are no common metrics in source and target projects, therefore, HDPM can only compare with within-project method (target→target) for multiple heterogeneous source projects.

Table 9 shows the Pd and Pf values of HDPM and WPDP method when using multiple heterogeneous source projects. Table 10 shows the F-measure values of HDPM and WPDP method. The Pd, Pf and F-measure values of these tables are the average values calculated from 100 repeated runs, and the numbers which are presented with boldface indicate the best results. The average Pd and Pf values of HDPM are 0.70 and 0.09 and the average F-measure values of HDPM is 0.71. From Tables 9 and 10, we can see that HDPM can obtain better performance of defect prediction compared with WPDP method. The p-value is 6.5×10^{-8} which indicates HDPM makes a statistically significant improvement in comparison with WPDP method. As compared with one-to-one HDPM, the Pd and F-measure values are increased by about 10% and the Pf value is similar. The reason is that using multiple heterogeneous source projects contains more useful information and can build a more accurate prediction model than using only one source project, so defect prediction for multiple heterogeneous source projects outperforms one source project. Above all, we can conclude that using

multiple heterogeneous source projects is applicable to HCPDP and extend the data acquisition range of defect prediction and make software defect prediction better applied to software engineering.

Table 7. Pd and Pf values of one-to-one HDPM with no common metrics.

Source→Target	Measure	HDPM	CCA+	Source→Target	Measure	HDPM	CCA+	Source→Target	Measure	HDPM	CCA+
EQ→AR5	Pd	**0.73**	0.66	MW1→EQ	Pd	0.56	**0.62**	JDT→ZXing	Pd	0.61	**0.68**
	Pf	**0.09**	0.18		Pf	**0.22**	**0.22**		Pf	**0.07**	0.43
AR5→EQ	Pd	0.59	**0.62**	LC→CM1	Pd	**0.58**	0.56	ZXing→JDT	Pd	0.67	**0.69**
	Pf	0.28	**0.22**		Pf	**0.02**	0.25		Pf	**0.00**	0.17
ML→AR4	Pd	0.61	**0.66**	CM1→LC	Pd	**0.67**	0.60	LC→Safe	Pd	0.63	**0.65**
	Pf	**0.02**	0.23		Pf	**0.00**	0.20		Pf	0.27	**0.24**
AR4→ML	Pd	0.53	**0.58**	JDT→PC1	Pd	0.73	**0.86**	Safe→LC	Pd	0.64	**0.70**
	Pf	**0.02**	0.20		Pf	**0.10**	0.29		Pf	**0.20**	0.20
PDE→AR4	Pd	0.51	**0.66**	PC1→JDT	Pd	0.50	**0.69**	PDE→ZXing	Pd	0.67	**0.68**
	Pf	**0.15**	0.23		Pf	**0.10**	0.17		Pf	**0.03**	0.43
AR4→PDE	Pd	0.52	**0.77**	EQ→Safe	Pd	0.55	**0.65**	ZXing→PDE	Pd	0.70	**0.77**
	Pf	**0.04**	0.33		Pf	**0.05**	0.24		Pf	**0.03**	0.33
EQ→MW1	Pd	**0.50**	0.44	Safe→EQ	Pd	**0.67**	0.62	Average	Pd	0.61	**0.63**
	Pf	**0.13**	0.21		Pf	**0.01**	0.22		Pf	**0.09**	0.25

Table 8. F-measure values of one-to-one HDPM with no common metrics.

Source→Target	HDPM	CCA+	Source→Target	HDPM	CCA+	Source→Target	HDPM	CCA+
EQ→AR5	**0.79**	0.55	MW1→EQ	**0.80**	0.66	JDT→ZXing	**0.49**	0.47
AR5→EQ	0.51	**0.66**	LC→CM1	**0.59**	0.39	ZXing→JDT	**0.78**	0.67
ML→AR4	**0.72**	0.59	CM1→LC	**0.67**	0.45	LC→Safe	**0.72**	0.63
AR4→ML	**0.49**	0.37	JDT→PC1	**0.56**	0.51	Safe→LC	0.30	**0.45**
PDE→AR4	**0.60**	0.56	PC1→JDT	**0.67**	0.54	PDE→ZXing	**0.57**	0.49
AR4→PDE	**0.60**	0.42	EQ→Safe	0.59	**0.63**	ZXing→PDE	**0.59**	0.47
EQ→MW1	**0.64**	0.27	Safe→EQ	0.55	**0.66**	-	-	-
Average	**0.61**	0.52	P-value	-	0.0159	-	-	-

4.6. Impact of parameters

In the experiments of our approach, there are two parameters may have an effect on the performance of defect prediction, one is *threshold*, the other is the number of repeated runs. In this section, we would discuss the effect of these parameters.

4.6.1. Impact of threshold

In our experiment setup, we run HDPM 100 times and add up these experiment results. When there are more than *threshold* experiments predict a module to be defective, the module is predicted to be defective eventually. Here, we would like to investigate what value of *threshold* can achieve better performance of defect prediction. We take *threshold* between 10 and 70 and record the

F-measure values. Figure 1 shows the boxplot of F-measure when taking different value of *threshold*. From Figure 2, we can see our approach can get better performance of defect prediction when the value of *threshold* varies from 30 to 45. That is, our approach can get better performance when 30–45% of the experiments predict a module to be defective and the module is predicted to be defective eventually. While the value of *threshold* is too low or too high, most of modules are predicted as defective or clean in the experiment, making the experiment results inaccuracy.

Table 9. Pd and Pf values of multiple-to-one HDPM.

Source→Target	Measure	HDPM	WPDP (Target→Target)	Source→Target	Measure	HDPM	WPDP (Target→Target)
{CM1, Apache,	Pd	**0.71**	0.38	{CM1, AR4, ML}→Safe	Pd	**0.79**	0.65
LC}→AR3	Pf	**0.01**	0.14		Pf	**0.11**	0.24
{MW1, ZXing,	Pd	**0.75**	0.38	{MW1, AR3, EQ}→Safe	Pd	0.61	0.65
EQ}→AR3	Pf	0.22	**0.14**		Pf	**0.01**	0.24
{MW1, Apache,	Pd	0.61	**0.66**	{MW1,AR4,LC}→ZXing	Pd	0.65	**0.68**
PDE}→AR4	Pf	**0.21**	0.23		Pf	**0.04**	0.43
{PC1, ZXing,	Pd	**0.75**	0.66	{PC1, AR3, EQ}→ZXing	Pd	0.67	**0.68**
EQ}→AR4	Pf	**0.03**	0.23		Pf	**0.17**	0.43
{CM1, ZXing,	Pd	**0.69**	0.66	{AR3, MW1,	Pd	**0.77**	0.75
ML}→AR5	Pf	0.21	**0.18**	ZXing}→EQ	Pf	**0.19**	0.37
{PC1, Apache,	Pd	**0.74**	0.66	{AR4, CM1, Safe}→EQ	Pd	0.68	**0.75**
PDE}→AR5	Pf	**0.15**	0.18		Pf	**0.02**	0.37
{AR3, Apache,	Pd	0.69	**0.74**	{AR4, CM1,	Pd	0.67	**0.69**
JDT}→CM1	Pf	**0.01**	0.37	ZXing}→JDT	Pf	**0.01**	0.17
{AR4, ZXing,	Pd	**0.80**	0.74	{AR5, MW1,	Pd	0.65	**0.69**
PDE}→CM1	Pf	**0.02**	0.37	Apache}→JDT	Pf	**0.14**	0.17
{AR3, Safe,	Pd	**0.70**	0.44	{AR3, PC1, Safe}→LC	Pd	0.61	**0.71**
JDT}→MW1	Pf	**0.06**	0.21		Pf	0.22	**0.19**
{AR5, ZXing,	Pd	**0.76**	0.44	{AR4, MW1,	Pd	**0.73**	0.71
ML}→MW1	Pf	**0.16**	0.21	Apache}→LC	Pf	**0.02**	0.19
{AR4, Safe,	Pd	0.73	**0.86**	{AR3, PC1,	Pd	**0.71**	0.58
JDT}→PC1	Pf	**0.02**	0.29	ZXing}→ML	Pf	**0.01**	0.20
{AR5, Apache,	Pd	0.73	**0.86**	{AR4, MW1, Safe}→ML	Pd	**0.70**	0.58
JDT}→PC1	Pf	**0.06**	0.29		Pf	**0.01**	0.20
{CM1, AR3,	Pd	0.65	**0.67**	{AR3, MW1,	Pd	0.65	**0.77**
JDT}→Apache	Pf	**0.08**	0.31	Safe}→PDE	Pf	**0.11**	0.33
{PC1, AR4,	Pd	**0.67**	**0.67**	{AR4, CM1,	Pd	0.76	**0.77**
PDE}→Apache	Pf	**0.11**	0.31	Apache}→PDE	Pf	**0.04**	0.33
Average	Pd	**0.70**	0.66	-	Pf	**0.09**	0.26

4.6.2. Impact of different number of repeated runs

To deal with the randomness of our approach, that is using random sampling to process class-unbalanced data of source project, we repeated the experiment 100 times. Here, we would

like to investigate whether the performance of our approach will be affected if we run HDPM with different number of repeated runs. We run HDPM 10–100 times in 40 pairs of heterogeneous source and target projects, and Figure 3 shows F-measure values with different number of repeated runs. We can see that the performance of HDPM is basically stable for different number of repeated runs, and the average F-measure values vary from 0.6083 to 0.6327. Thus, we can conclude that different number of repeated runs has limited impact on the performance of our approach.

Table 10. F-measure values of multiple-to-one HDPM.

Source→Target	HDPM	WPDP (Target→Target)	Source→Target	HDPM	WPDP (Target→Target)
{CM1, Apache, LC}→AR3	**0.81**	0.31	{CM1, AR4, ML}→Safe	0.61	**0.63**
{MW1, ZXing, EQ}→AR3	**0.66**	0.31	{MW1, AR3, EQ}→Safe	**0.75**	0.63
{MW1, Apache, PDE}→AR4	**0.69**	0.49	{MW1,AR4,LC}→ZXing	**0.54**	0.50
{PC1, ZXing, EQ}→AR4	**0.79**	0.49	{PC1, AR3, EQ}→ZXing	**0.69**	0.50
{CM1, ZXing, ML}→AR5	**0.57**	0.55	{AR3, MW1, ZXing}→EQ	**0.87**	0.65
{PC1, Apache, PDE}→AR5	**0.64**	0.55	{AR4, CM1, Safe}→EQ	**0.82**	0.65
{AR3, Apache, JDT}→CM1	**0.62**	0.38	{AR4, CM1, ZXing}→JDT	**0.64**	0.59
{AR4, ZXing, PDE}→CM1	**0.88**	0.38	{AR5, MW1, Apache}→JDT	**0.78**	0.59
{AR3, Safe, JDT}→MW1	**0.69**	0.27	{AR3, PC1, Safe}→LC	**0.67**	0.44
{AR5, ZXing, ML}→MW1	**0.65**	0.27	{AR4, MW1, Apache}→LC	**0.70**	0.44
{AR4, Safe, JDT}→PC1	**0.80**	0.41	{AR3, PC1, ZXing}→ML	**0.83**	0.40
{AR5, Apache, JDT}→PC1	**0.73**	0.41	{AR4, MW1, Safe}→ML	**0.66**	0.40
{CM1, AR3, JDT}→Apache	**0.75**	0.68	{AR3, MW1, Safe}→PDE	**0.62**	0.40
{PC1, AR4, PDE}→Apache	**0.68**	**0.68**	{AR4, CM1, Apache}→PDE	**0.84**	0.40
Average	**0.71**	0.49	P-value	-	6.5×10^{-8}

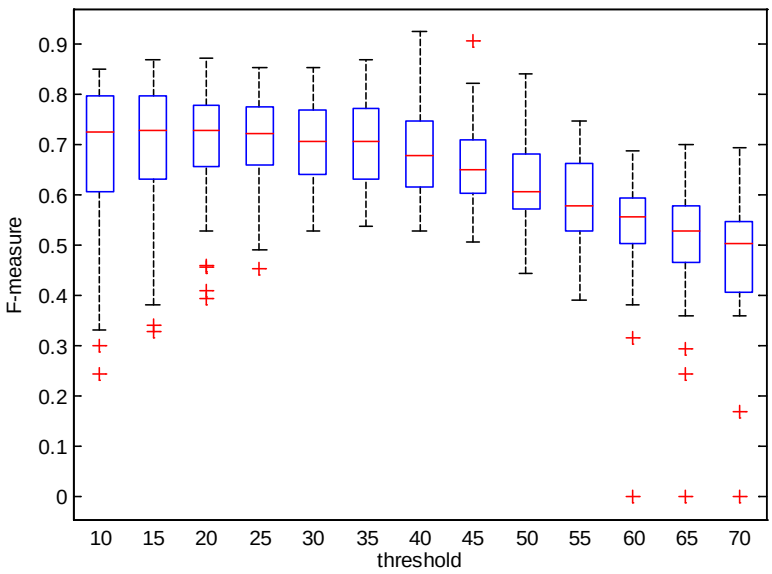

Figure 2. The boxplot of F-measure values of *threshold* from 10 to 70.

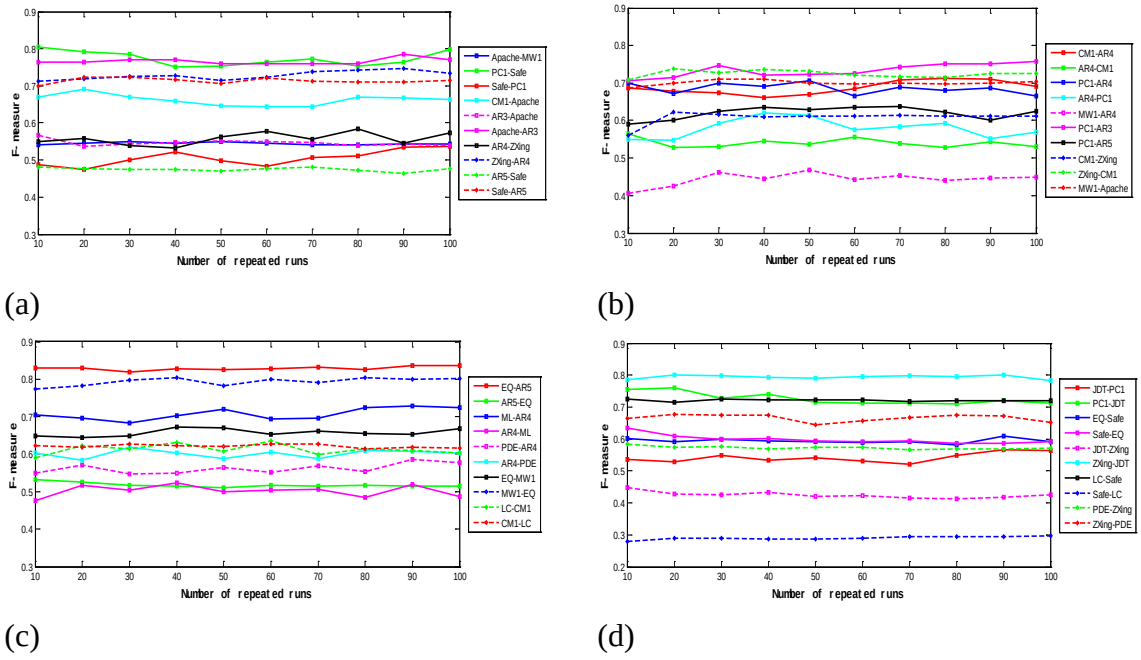

Figure 3. F-measure for HDPM with different number of repeated runs.

4.7. Answers to research questions

RQ1. Is our approach HDPM helpful for heterogeneous cross-project defect prediction?

From Tables 5 and 6 which source and target projects have common metrics, the Pd values of HDPM vary from 0.42 to 0.78 and the F-measure values of HDPM vary from 0.46 to 0.80. The average Pd and F-measure of HDPM are 0.65 and 0.63, respectively. From Tables 7 and 8 which source and target projects have no common metrics, the Pd and F-measure values of HDPM vary from 0.50 to 0.73 and 0.30 to 0.80. The average Pd and F-measure of HDPM are both 0.61. The p-value of Tables 4 and 6 are all below 0.05. Thus, we can conclude that our approach is helpful for heterogeneous cross-project defect prediction.

The existing CPDP methods can only use common metrics and achieve unsatisfactory results when source and target projects contain few common metrics. However, our approach makes full use of all the metrics of source and target projects, and can achieve better performance of defect prediction at most times compared with existing CPDP methods. And HDPM can obtain comparable or even better prediction results compared with within-project defect prediction. The performance of HDPM is similar whether there are common metrics in source and target projects or not. The p-values indicate HDPM can make a statistically significant improvement in comparison with other methods.

RQ2. Whether using multiple heterogeneous source projects is applicable to software defect prediction?

As we for the first time use multiple heterogeneous source projects to predict defects, we also investigate whether using multiple heterogeneous source projects is applicable to HCPDP. From Table 9, the Pd and Pf values of HDPM vary from 0.61 to 0.80 and 0.01 to 0.22, and the average

Pd and Pf of HDPM are 0.70 and 0.09. From Table 10, the F-measure values of HDPM vary from 0.54 to 0.88 and the average F-measure is 0.71. We can see that our approach outperforms within-project defect prediction at most times. The average Pd and F-measure of multiple-to-one HDPM outperforms one-to-one HDPM by 11.11 and 14.52%, respectively. The p-value of Table 8 is 6.5×10^{-8} which is less than 0.05. It indicates that the performance of defect prediction of multiple-to-one HDPM is better than one-to-one HDPM. In addition, all the experiments can get the results in several minutes, and this indicates that the time efficiency of our method is acceptable. Therefore, we can conclude that using multiple heterogeneous source projects is applicable to HCPDP and extend the data acquisition range of defect prediction and make software defect prediction better applied to software engineering.

4.8. Threats to validity

Although the experiment achieved good results, the generalizability of our results may be limited since the scale of the dataset in the experiment is still small. However, we have analyzed 14 projects from 4 different public defect datasets (SOFTLAB, NASA, ReLink and AEEEM) containing a total of 7511 modules, and the data is from publish defect datasets which are well represented, so it can verify the effectiveness of the method at some extent. In the future, we plan to analyze even more defect data to reduce this threat, especially the data from commercial software projects.

We evaluated our approach in recall, false-positive and F-measure that are commonly used performance measures in defect prediction. However, there are other measures that can be used in defect prediction, such as area under curve (AUC), and G-means which are also comprehensive measures.

For the four compared methods TNB, NN-filter, TCA+ and CCA+, we got the program code of TCA+ from the author, and we carefully implement TNB, NN-filter and CCA+ by following their papers. It may affect the accuracy of the experiment.

5. Conclusion

In this paper, we propose Heterogeneous Defect Prediction with Multiple source projects (HDPM) which can use multiple heterogeneous source projects for defect prediction to solve heterogeneous cross-project defect prediction (HCPDP) problem. HDPM constructs the projective matrix which can transfer source project to target project space, making the distributions of source and target projects are similar. Our approach is applicable for heterogeneous cross-project defect prediction and for the first time to use multiple heterogeneous source projects in software defect prediction, making software defect prediction better applied to software engineering. The results of our experiments show that HDPM can achieve better performance of defect prediction than existing CPDP methods, and can obtain comparable or even better prediction results compared with WPDP method. In addition, the performance of defect prediction for multiple-to-one HDPM outperforms one-to-one HDPM.

In the future, we plan to evaluate HDPM with more datasets to validate the generalization of our approach, and try other transfer learning methods and classifiers to improve the performance of defect prediction further.

Acknowledgements

The presented research is partly funded by the National Key Research and Development Program of China (2017YFB1003103).

Conflict of Interest

The authors declared that they have no conflicts of interest to this work.

References

1. J. Nam, S. J. Pan and S. Kim, *Transfer defect learning*, 2013 35th International Conference on Software Engineering (ICSE), 2013, 382–391. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/6606584.
2. X. Y. Jing, S. Ying, Z. W. Zhang, et al., *Dictionary learning based software defect prediction*, Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, 414–423. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2568320.
3. Z. Mahmood, D. Bowes, P. C. R. Lane, et al., *What is the Impact of Imbalance on Software Defect Prediction Performance?*, Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, ACM, 2015. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2810150.
4. C. Tantithamthavorn, *Towards a better understanding of the impact of experimental components on defect prediction modeling*, 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016, 867–870. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/7883423.
5. B. Turhan, T. Menzies, A. B. Bener, et al., On the relative value of cross-company and within-company data for defect prediction, *Empirical Software Eng.*, **14** (2009), 540–578.
6. Y. Ma, G. Luo, X. Zeng, et al., Transfer learning for cross-company software defect prediction, *Inf. Software Technol.*, **54** (2012), 248–256.
7. G. Canfora, A. De Lucia, M. Di Penta, et al., *Multi-objective cross-project defect prediction*, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, 252–261. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/6569737.
8. F. Peters, T. Menzies and A. Marcus, *Better cross company defect prediction*, Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, 409–418. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2487161.
9. L. Chen, B. Fang, Z. Shang, et al., Negative samples reduction in cross-company software defects prediction, *Inf. Software Technol.*, **62** (2015), 67–77.
10. J. Nam and S. Kim, *Heterogeneous defect prediction*, Proceedings of the 2015 10th joint meeting on foundations of software engineering, ACM, 2015, 508–519. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2786814.
11. X. Jing, F. Wu, X. Dong, et al., *Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning*, Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015, 496–507. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2786813.

12. M. H. Halstead, *Elements of Software Science*, Elsevier Science, New York, 1977.
13. T. J. McCabe, A complexity measure, *IEEE Trans. Software Eng.*, **4** (1976), 308–320.
14. S. R. Chidamber and C. F. Kemerer, A metrics suite for object oriented design, *IEEE Trans. Software Eng.*, **20** (1994), 476–493.
15. T. L. Graves, A. F. Karr, J. S. Marron, et al., Predicting fault incidence using software change history, *IEEE Trans. Software Eng.*, **26** (2000), 653–661.
16. K. O. Elish and M. O. Elish, Predicting defect-prone software modules using support vector machines, *J. Syst. Software*, **81** (2008), 649–660.
17. A. S. Andreou and E. Papatheocharous, *Software cost estimation using fuzzy decision trees*, 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, 371–374. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/4639344.
18. N. Bettenburg, M. Nagappan and A. E. Hassan, *Think locally, act globally: Improving defect and effort prediction models*, 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), 2012, 60–69. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/6224300.
19. S. J. Pan and Q. Yang, A Survey on Transfer Learning, *IEEE Trans. Knowl. Data Eng.*, **22** (2010), 1345–1359.
20. H. F. Chang and A. Mockus, *Constructing universal version history*, Proceedings of the 2006 international workshop on Mining software repositories, 2006, 76–79. Available from: https://dl_acm.xilesou.top/citation.cfm?id=1138002.
21. T. Menzies, B. Caglayan, E. Kocaguneli, et al., The promise repository of empirical software engineering data, **2012** (2012).
22. M. Shepperd, Q. Song, Z. Sun, et al., Data quality: Some comments on the NASA software defect datasets, *IEEE Trans. Software Eng.*, **39** (2013), 1208–1215.
23. M. D'Ambros, M. Lanza, R. Robbes, *An extensive comparison of bug prediction approaches*, 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010, 31–41. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/5463279.
24. R. Wu, H. Zhang, S. Kim, et al., *Relink: Recovering links between bugs and changes*, Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, 2011, 15–25. Available from: https://dl_acm.xilesou.top/citation.cfm?id=2025120.
25. S. Zhong, T. M. Khoshgoftaar and N. Seliya, *Unsupervised Learning for Expert-Based Software Quality Estimation*, HASE, 2004, 149–155. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.1471&rep=rep1&type=pdf.
26. P. S. Bishnu and V. Bhattacherjee, Software fault prediction using quad tree-based k-means clustering algorithm, *IEEE Trans. Knowl. Data Eng.*, **24** (2012), 1146–1150.
27. G. Abaei, Z. Rezaei and A. Selamat, *Fault prediction by utilizing self-organizing Map and Threshold*, 2013 IEEE International Conference on Control System, Computing and Engineering, 2013, 465–470. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/6720010.
28. J. Nam and S. Kim, *CLAMI: Defect Prediction on Unlabeled Datasets (T)*, 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, 452–463. Available from: https://ieeexplore_ieee.xilesou.top/abstract/document/7372033.

29. F. Zhang, Q. Zheng, Y. Zou, et al., *Cross-project defect prediction using a connectivity-based unsupervised classifier*, Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, 309–320.
30. J. Han, J. Pei and M. Kamber, *Data Mining: Concepts and Techniques*, Elsevier, 2012.
31. A. B. A. Graf and S. Borer, *Normalization in support vector machines*, Joint Pattern Recognition Symposium, Springer, Berlin, Heidelberg, 2001, 277–282.
32. M. Harel and S. Mannor, Learning from multiple outlooks, *arXiv preprint arXiv1005.0027*, 2010.
33. L. Yang, L. P. Jing, J. Yu, et al., Heterogeneous transductive transfer learning algorithm, *J. Software*, **26** (2015), 2762–2780 (in Chinese).
34. J. C. Gower and G. B. Dijksterhuis, *Procrustes problems*, Oxford University Press on Demand, 2004.
35. F. Wilcoxon, *Individual comparisons by ranking methods*, Breakthroughs in Statistics, Springer Series in Statistics (Perspectives in Statistics), Springer, New York, 1992, 196–202.

AIMS Press