



MHCPDP: multi-source heterogeneous cross-project defect prediction via multi-source transfer learning and autoencoder

Jie Wu¹ · Yingbo Wu¹ · Nan Niu² · Min Zhou¹

Accepted: 11 March 2021 / Published online: 27 April 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

Heterogeneous cross-project defect prediction (HCPDP) is aimed at building a defect prediction model for the target project by reusing datasets from source projects, where the source project datasets and target project dataset have different features. Most existing HCPDP methods only remove redundant or unrelated features without exploring the underlying features of cross-project datasets. Additionally, when the transfer learning method is used in HCPDP, these methods ignore the negative effect of transfer learning. In this paper, we propose a novel HCPDP method called multi-source heterogeneous cross-project defect prediction (MHCPDP). To reduce the gap between the target datasets and the source datasets, MHCPDP uses the autoencoder to extract the intermediate features from the original datasets instead of simply removing redundant and unrelated features and adopts a modified autoencoder algorithm to make instance selection for eliminating irrelevant instances from the source domain datasets. Furthermore, by incorporating multiple source projects to increase the number of source datasets, MHCPDP develops a multi-source transfer learning algorithm to reduce the impact of negative transfers and upgrade the performance of the classifier. We comprehensively evaluate MHCPDP on five open source datasets; our experimental results show that MHCPDP not only has significant improvement in two performance metrics but also overcomes the shortcomings of the conventional HCPDP methods.

Keywords Autoencoder · Heterogeneous cross-project defect prediction · Multi-source transfer learning · Modified autoencoder

✉ Yingbo Wu
wyb@cqu.edu.cn

¹ School of Software Engineering, Chongqing University, Chongqing, China

² Department of Electrical Engineering and Computer Science, University of Cincinnati, Cincinnati, USA

1 Introduction

The software is a collection of large datasets with thousands or millions of lines of code. How to ensure software quality has become a key issue. Bugs inevitably exist in software projects, even though experienced developers using powerful development platforms develop these projects (Xiaoting et al., 2020).

Within-project defect prediction (WPDP) (Lee et al., 2011; Menzies et al., 2007b; Nam et al., 2015) first constructs classification models based on a sufficient amount of historical labeled software entities from a project and then uses the models to predict the defect labels of new entities within the same project (Xu et al., 2018). However, in some situations where we do not have enough datasets with labeled defect information, WPDP does not necessarily work well.

In order to solve this problem, many researchers continue to explore open source software, and then, they share datasets such as NASA (Menzies et al., 2007a), Relink (Wu et al., 2011), Promise (Jureczko & Madeyski, 2010), and AEEEM (D'Ambros et al., 2010). This makes cross-project defect prediction (CPDP) possible. CPDP (He et al., 2012; Ma et al., 2012; Rahman et al., 2012; Turhan et al., 2009) is the study of how to transfer knowledge related to the target project from the source project (Hosseini et al., 2019). This topic has attracted a lot of attention lately in the literature.

However, most existing CPDP methods are based on the assumption that both source and target projects have the identical features or share similar features. Unfortunately, it is not always feasible when source and target projects have little or no shared features (Turhan et al., 2008), because different projects may be developed with distinct programming languages and the features may be collected at different levels of granularity using various tools. Jing et al. (Hosseini et al., 2019) has showed that NASA and Relink, as well as Softlab and Relink, have few common metrics, and there are no common metrics between the remaining datasets. In this case, the effect obtained by CPDP can be limited. Therefore, the heterogeneous cross-project software defect prediction (HCPDP) method was proposed.

The main challenge of HCPDP is how to narrow the difference between two different feature spaces of source projects and target projects. Besides, the relationship between source projects and target projects influences the effectiveness of transfer learning. Resource utilization that is not related to the target may degrade the performance of the classifier and result in a negative transfer. One strategy to reduce negative transfer is to increase the number of sources.

In this paper, we propose a novel multi-source heterogeneous cross-project defect prediction (MHCPDP) method. To narrow the gap between source datasets and target datasets, MHCPDP uses an autoencoder to extract the intermediate features from the original datasets instead of simply removing redundant and unrelated features and adopts a modified autoencoder algorithm to make instance selection for eliminating irrelevant instances from the source datasets.

Furthermore, to reduce the impact of negative transfer in transfer learning, MHCPDP incorporates multiple source projects to solve the problem of insufficient source projects and defines a multi-source transfer-learning algorithm to reduce the impact of negative transfer.

To evaluate our approach, we conducted large-scale experiments on five public datasets. This paper mainly focuses on answering the following three research questions.

- 1 Is MHCPDP better than the existing HCPDP methods?
- 2 How does MHCPDP compare to unsupervised learning?
- 3 What is the result of MHCPDP when compared with WPDP?

This paper makes the following main contributions:

- 1 We propose the MHCPDP method for HCPDP. MHCPDP adopts the autoencoder and a modified autoencoder algorithm for dimensionality reduction and instance selection to reduce the feature space gap between the source project and target project. Furthermore, we develop a multisource transfer-learning algorithm to address the negative transfer effects of transfer learning in HCPDP, and overcome the shortcoming of insufficient source projects.
- 2 We have comprehensively evaluated MHCPDP by using two widely used performance indicators. The results show that MHCPDP not only is comparable with the performance of two of the most advanced HCPDP but also overcomes the shortcomings of the conventional HCPDP methods.
- 3 We have conducted extensive experiments on our method. The results show that MHCPDP has the advantages and competitiveness when it compares with the WPDP scheme and the state-of-the-art unsupervised learning method.

The rest of the paper is organized as follows. Section 2 introduces the related work. After presenting our method MHCPDP in Sect. 3, we detail the experimental setup in Sect. 4. Section 5 shows experimental results and statistical test. Section 6 discusses the parameter sensitivity to MHCPDP, the effectiveness of MHCPDP, and the impact of different classifiers on our method, followed by threats to validity in Sect. 7. Finally, Sect. 8 concludes our work.

2 Related work

2.1 Cross-project defect prediction method

Cross-project defect prediction is to build a defect prediction model for the target project with existing project datasets. In order to make the model effect and accurate, scholars have carried out a series of researches on this.

Briand et al., (2002) were the first to study CPDP where they proposed a new use of exploratory analysis techniques (namely, multivariate adaptive regression splines) to establish this failure propensity model. The results show that models built on one system can accurately classify classes within another system based on fault propensity.

Zimmermann et al. (2009) conducted a series of large-scale studies on cross-project defect prediction models. For 12 real-world applications, they ran 622 cross-project forecasts. The results show that cross-project prediction is a serious challenge, that is, using only models from the same field or projects with the same process does not lead to accurate predictions.

Nam et al. (2015) proposed that the performance of cross-project defect prediction is generally poor, mainly due to the difference in feature distribution between the source project and the target project. In the paper, they apply a state-of-the-art transfer learning

method—TCA (Pan et al., 2011) to make the distribution of features in source and target projects similar. In addition, they extended a TCA to propose a new transfer defect learning method TCA+. They also conducted experiments on eight open source projects, and the results showed that TCA+ significantly improved cross-project forecasting performance.

However, all these studies assume that the target and source project data share some common features. Therefore, if the cross-project data have heterogeneous feature sets (i.e., HCPDP scenario), these methods are a failure.

2.2 Heterogeneous cross-project defect prediction method

Nam et al. (2013) propose a heterogeneous defect prediction (HDP) method that matches different metrics in different projects. Experiments show that their proposed HDP model is feasible and has achieved good results. In addition, they also studied the lower bounds of the source and target dataset sizes for effective transfer learning in defect prediction. Based on their empirical and mathematical studies, it is possible to display only 50 instances of the category dataset, sufficient to construct defect predictions and apply HDP.

Jing et al. (2015) propose a method for using a unified metric representation (UMR) source project and target project. UMR includes three types of metrics, that is, the source project and the target project are in common, the source project is unique, and the target project is unique. Then a typical correlation analysis (CCA) (Knapp, 1978) was introduced into the method. Their approach is clearly superior to the latest cooperating CPDP method, and it achieves comparable predictive performance.

Besides, Li et al. propose a series of solutions for HCPDP, for example, a novel two-stage ensemble learning (TSEL) approach to HDP (Li et al., 2019), a new cost-sensitive transfer kernel canonical correlation analysis (CTKCCA) approach for HDP (Li et al., 2017a), a multi-source selection based manifold discriminant alignment (MSMDA) approach (Li et al., 2017b), and an empirical study on heterogeneous defect prediction approaches (Chen et al., 2020).

In a word, the prediction performance of HCPDP model is greatly affected by its construction method. Most of the studies have constructed multiple models with different modeling techniques and have made great progress in their comparative performance. The main differences between our approach and existing HCPDP methods are as follows. Firstly, our method adopts the autoencoder and a modified autoencoder algorithm for dimensionality reduction and instance selection to reduce the feature space gap between the source project and target project. Secondly, our method uses multiple heterogeneous source projects to make up insufficient source datasets, and we propose a multi-source transfer-learning algorithm to address the negative transfer effects of transfer learning.

2.3 Transfer learning

With the increasing popularity of machine learning applications in recent years, existing supervised learning tends to require a large amount of annotated data. However, labeling data is a cumbersome task that requires a lot of work, human resources, and financial resources. So transfer learning (Fu-Zhen & Ping, 2015; Pan & Yang, 2010) that requires less data annotation has received attention.

A common assumption of traditional machine learning algorithms is that the probability distributions of training datasets and test datasets are the same. Under this assumption, when presenting a set of new datasets with different distributions, we need to collect and

label new training samples to learn new classifiers. However, if we use the useful information in the existing labeled dataset to train the classifier, we can save time. Transfer learning is to apply the knowledge or model learned in a certain domain task to different but related domains. However, when the probability distributions between the source domain and target domain vary greatly, it may lead to serious negative transfer. How to solve the impact of negative transfer in HCPDP is a main motivation of this paper.

3 Method

3.1 Framework

As shown in Fig. 1, the framework of MHCPDP consists of five steps: data collection, data pre-processing, feature processing and instance selection, multi-source transfer learning, and defect prediction.

Step 1. Data collection: The first step of the framework is data collection. In this paper, we collect the datasets that come from five public datasets, i.e., AEEEM, MORPH, NASA, Relink, and SOFTLAB.

Step 2. Data pre-processing: The second step is data pre-processing. After the data collection, it is necessary to pre-process datasets because they may have several irregular data. The pre-processing step in this paper includes oversampling and datasets standardization. For over-sampling, we adopt smote method, and z-score for dataset standardization.

Step 3. Feature representation and instance selection: The third step is feature representation and instance selection. In this step, we firstly adopt the autoencoder to obtain

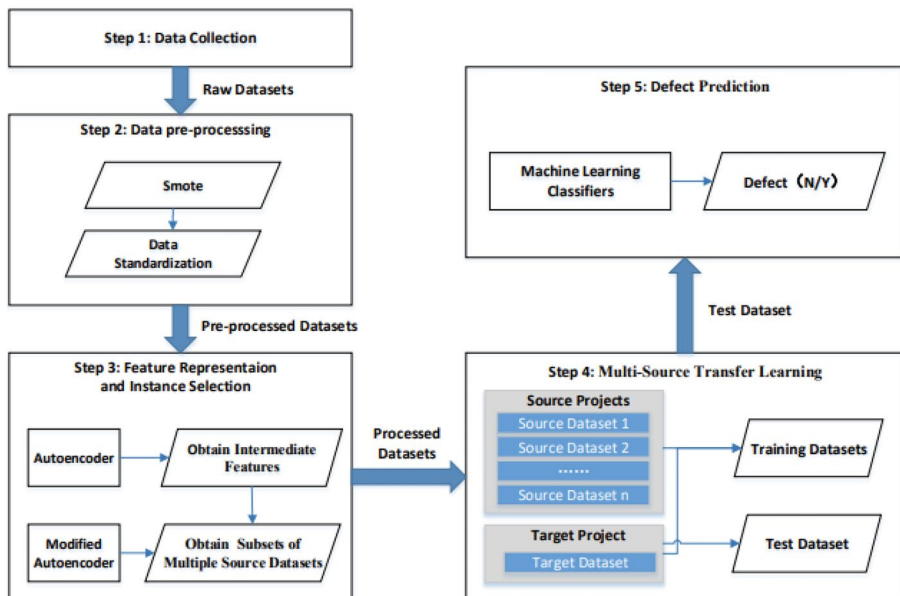


Fig. 1 Framework of MHCPDP

the intermediate features of source datasets. Then, we design a modified autoencoder for instance selection to narrow the gap between the source datasets and the target dataset. The supplementary procedure of instance selection is described in Sect. 3.3.

Step 4. Multi-source transfer learning: The fourth step is data pre-processing. The project (i.e., target project) that is needed to perform defect prediction may not have enough labeled defect for training the classification model, while other projects (i.e., source projects) may have a large amount of label data. Transfer learning can capture useful information from the source project as additional training data for the target project's prediction. In this step, we propose a multi-source transfer learning algorithm to solve the negative transfer and further narrow the gap between the source datasets and the target datasets. The supplementary procedure of the multi-source transfer learning algorithm is presented in Sect. 3.4.

Step 5. Defect prediction: The fifth step is defect prediction. Machine learning classifiers are used to build the defects prediction model in this step. To compare the performance of different classifiers, four machine learning classifiers are used, including decision tree, Naive Bayes, logistic regression, and random forest. The evaluation metrics used in this paper are AUC and g-measure.

3.2 Problem definition

Let the target data matrix be $T = [t_1, t_2, \dots, t_m]$, where $t_i = [t_{i1}, t_{i2}, \dots, t_{ip}]^T \in R^p$; m and p represent the number of target project entities and features, respectively. T_{ij} is the j th features of i th target entity. Similarly, let the source project data matrix be $S = [s_1, s_2, \dots, s_n]$, where $s_i = [s_{i1}, s_{i2}, \dots, s_{iq}]^T \in R^p$; n and q represent the number of source entities and features, respectively. A task τ is made of a label space $\gamma = [+1, -1]$, and a boolean function $\mathcal{F}: x \rightarrow y$. s_{ij} indicates the j th feature of the i th target entity. $v_S = [v_{s1}, v_{s2}, \dots, v_{sn}]^T, v_{si} \in \{0, 1\}$, and v_{si} represents the i th instance in the source project. When $v_{si}=1$, the instance is selected; otherwise, it is not selected. In addition, the $f_e(\cdot)$ is encoding functions and the $f_d(\cdot)$ is decoding function.

3.3 Feature processing and instance selection

3.3.1 Feature processing

Inspired by the work from GE Hinton and RR Salakhutdinov (Hinton & Salakhutdinov, 2006), MHCPDP performs the feature processing based on an autoencoder instead of simply removing redundant and unrelated features. We use autoencoder to mine potential features and the connections between features.

The autoencoder is an unsupervised neural network model, which contains two processes of encoding and decoding. The encoding process can learn the implicit features of the input data. The decoding process is to reconstruct the original input data by using the learned new features. Through such processing, the intermediate features can be considered as feature extraction of the original data and are advanced features with robustness.

By training, we make the output x' close to the input $x^i \in R^n$. When we add some restrictions to the autoencoder neural network, such as limiting the number of hidden neurons, we can find some interesting structures from the input data. For example, the data dimension we input is $n=64$, and there are 25 hidden neurons in the hidden layer L_2 .

Since there are only 25 hidden neurons, we force the autoencoder neural network to learn the compressed representation of the input data. In other words, it must be reconstructed a 64-dimensional input value from the 25-dimensional hidden neuron activation degree vector $a^{(2)} \in R^{25}$. If some specific structures are hidden in the input data, for example, some input features are related to each other, and then, this algorithm can well find the correlation of the input data. After the network is trained, $a^{(2)} \in R^{25}$ of layer L_2 corresponding to each input x^i is equivalent to the reduced-dimensional data.

In this paper, the autoencoder is used for feature extraction, and the number of features in the five public datasets after extraction is same.

3.3.2 Instance selection

After dimensionality reduction, we modified the autoencoder algorithm for instance selection. This idea comes from distant domain transfer learning (Tan et al., 2017). The goal is to narrow the gap between the source and target datasets.

The specific steps of Algorithm 1 as follows:

Step 1: We take the source dataset, the target dataset, and the number of iterations as input data.

Step 2: Initializing the parameters, $V_s = 1$ of the source domain dataset means instances in the source domain dataset are selected.

Step 3: Iterate N times and update V_s by adjusting the parameters of the autoencoder to minimize the loss function.

Step 4: Finally, remove the source domain $V_s = 0$ and a subset of the source datasets as output data.

The loss function is as follows:

$$\min_{f_e, f_d} \sum_n^{i=1} \|x' - x^i\|_2^2 + l(y^i, f_c(f_e(x^i))) \quad (1)$$

y^i is label of x^i . $l(\cdot)$ is loss function. $f_c(\cdot)$ is a classification function to output classification probabilities.

The original autoencoder reconstructs the output through input. Through the original input, we found the instance with a large difference between the source datasets and the target datasets and marked it as zero. When finally outputting, these instances do not output no longer. The final output instances are a subset of the source datasets, and the gap between this subset and the target datasets is small. After we have dealt with the source datasets, we reconstruct the datasets. Then, use multi-transfer learning for modeling.

3.4 Multi-source transfer learning

Transferring learning has been widely used since it was introduced into software defect prediction, but the effects of negative transfer have been rarely mentioned. We use Algorithm 2 to solve this problem and further narrow the gap between the source datasets and the target datasets.

Algorithm 2 Multi-Source Transfer Learning

Input: filter subset of S_1, S_2, \dots, S_N , target dataset T , and the maximum number of iterations M

Output: Target classifier function $F_t: x \rightarrow \gamma$

1 Set $\alpha_s = \frac{1}{2} \ln(1 + \sqrt{2 \ln \frac{n_s}{M}})$, where $n_s = \sum_k n_{S_k}$

2 Initialize the weight vector $(w^{S_1}, \dots, w^{S_N}, w^T)$, where $w^{S_k} = (w_1^{S_k}, \dots, w_{n_{S_k}}^{S_k})$, and $w^T = (w_1^T, \dots, w_{n_T}^T)$ to the desired distribution

for $t \leftarrow 1$ to M do

3 Empty the set of candidate weak classifiers, $\mathcal{F} \leftarrow \emptyset$

4 Normalize to 1 the weight vector $(w^{S_1}, \dots, w^{S_N}, w^T)$
for $k \leftarrow 1$ to N do

5 Find the candidate weak classifier $h_t^k: x \rightarrow y$ that minimizes the classification error over the combined set $S_k \cup T$, weight according to (w^{S_k}, w^T)

6 Compute the error of h_t^k on T :

$$\epsilon_t^k = \sum_j \frac{w_j^T [y_j^T \neq h_t^k(x_j^T)]}{\sum_i w_i^T}$$

7 $\mathcal{F} \leftarrow \mathcal{F} \cup (h_t^k, \epsilon_t^k)$

8 Find the weak classifier $h_t: x \rightarrow y$ such that
 $(h_t, \epsilon_y) = \arg \min_{(h, \epsilon) \in \mathcal{F}} \epsilon$

9 Set $\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$, where $\epsilon_t < \frac{1}{2}$

10 Update the weight vector

$$w_i^{S_k} \leftarrow w_i^{S_k} e^{-\alpha_t |h_t(x_i^{S_k}) - y_i^{S_k}|}$$

$$w_i^T \leftarrow w_i^T e^{\alpha_t |h_t(x_i^T) - y_i^T|}$$

return $F_t: x \rightarrow \gamma = \text{sign}(\sum_t \alpha_t h_t(x))$

Source projects that have little to do with the target project in transfer learning may reduce the performance of the classifier and cause negative transfer. An effective way to solve the negative transfer is to use multiple sources for transfer learning (Yao & Doretto, 2010). In research, single-source transfer learning is often used, and the resulting negative transfer is inevitable. This paper introduces multi-source transfer learning to solve the impact of negative transfer in HCPDP.

Algorithm 2 is a multi-source transfer learning algorithm. Our inspiration comes from Yi Yao's multi-source transfer learning. In the multi-source transfer learning algorithm, we take the filtered subset of S_1, \dots, S_N , target dataset T , and the maximum number of iterations M as input. We take the target classifier function $F_t: x \rightarrow y$ as output. Learning the task τ for the target domain T , in traditional machine learning, from the given source datasets according to certain

criteria, amounts to estimating a classifier function $F_t : x \rightarrow y$. By adjusting parameters, it can reduce data that is different from the target dataset, thereby limiting the effects of negative transfers. We use this algorithm to solve the impact of negative transfer in HCPDP, narrow the gap between the source data domain and the target data domain, and increase the number of sources.

4 Empirical setup

4.1 Benchmark datasets

We selected five public datasets as the benchmark datasets, namely AEEEM, MORPH, NASA, Relink, and SOFTLAB. These five datasets are widely used for defect prediction (D'Ambros et al., 2012; Herzig et al., 2013a; Herzig et al., 2013b; Jing et al., 2015; Nam et al., 2013; Nam et al., 2015; Tantithamthavorn et al., 2015; Xu et al., 2016; Pingclasai, xxxx). Table 1 reports the detailed information of the items in the five datasets, including the name of the items (dataset), the number of features (metrics), the number of entities (entities), and the number of defective entities (Buggy).

Table 1 Statistic of benchmark datasets

Group	Datasets	Entities	Buggy	Metrics
AEEEM	EQ	324	129	61
	JDT	997	206	61
	LC	691	64	61
	ML	1862	245	61
	PDE	1292	209	61
Relink	Apache	194	98	26
	Safe	56	22	26
	Zxing	399	118	26
MORPH	ant-1.3	125	20	20
	arc	234	27	20
	camel1.0	339	13	20
	poi-1.5	237	141	20
	Redktor	176	27	20
	skarbonka	45	9	20
	tomcat	858	77	20
	xerces1.2	440	71	20
NASA	cm1	344	42	37
	mw1	264	27	37
	pc1	759	61	37
	pc3	1125	140	37
	pc4	1399	178	37
	kc3	200	36	39
SOFTLAB	ar1	121	9	29
	ar3	63	8	29
	ar4	107	20	29
	ar5	36	8	29
	ar6	101	15	29

These datasets may share some of the same features and certainly differ from each other. NASA and SOFTLAB contain proprietary datasets from NASA and a Turkish software company, respectively (Turhan et al., 2009). D'Ambros et al. (2010) collected the AEEEM dataset. Features in the AEEEM dataset include changing indicators, source code indicators, entropy of source code indicators, and drain of source code indicators (Appendix). Wu et al. (2011) collected the Relink dataset, and the items they analyzed were Apache HTTP Server, Safe, and Zxing. The dataset considers 26 metrics, all of which focus on code complexity. The MORPH group contains defect datasets of several open source projects used in the study about the dataset privacy issue for defect prediction (Peters & Menzies, 2012).

4.2 Performance Indicators

In this paper, we chose two commonly used metrics as indicators to measure the performance of our experiment.

4.2.1 AUC

Area under ROC curve (AUC) (Jiang et al., 2009) is the area under the ROC curve (Fawcett, 2006). The full name of the ROC curve is the receiver's working characteristic curve. AUC is an evaluation index to measure the pros and cons of a two-class model, which indicates the probability that the predicted positive example is ranked in front of the negative example. The calculation of this indicator has nothing to do with the threshold setting. The value ranges from 0 to 1. The higher the value, the better the model's performance.

4.2.2 G-measure

There is a contradiction between the precision (It can also be called the probability of false alarm.) and the recall rate (It can also be called the probability of detection.). In these occasions, g-measure can be used to effectively balance the two indicators. The calculation formula is as follows:

$$g - \text{measure} = \frac{2 \times pd \times (1 - pf)}{pd + (1 - pf)} \quad (2)$$

Among them, pf is the probability of false alarm, which refers to the proportion of all non-defective modules that are predicted to be defective. In addition, pd is the probability of detection. It returns the proportion of all defective modules that are predicted to be defective. G-measure is the harmonic mean of pd and (1-pf).

4.3 Prediction model

In this work, we train a decision tree model with the mapped source project dataset and then apply it to the mapped target project datasets. The decision tree model is widely used for software defect prediction.

4.4 Experimental design

At the beginning of the experiment, some parameters need to be set. The parameters will be adjusted with multiple experiments. Finally, a relatively optimal parameter is selected as the final parameter setting. The initial values of the parameters are shown in the figure below.

Following the steps of MHCPDP that are presented in Sect. 3.1, we carry out the experiments. To alleviate the bias, we run MHCPDP 30 times and report the average indicator values. For the parameter values, we empirically set $x=100$ (EPOCHS: it is defined as a single training iteration of all batches in the forward and backward propagation.) and $y=128$ (BATCH_SIZE: it is number of samples selected for one training.) as the basic setting through extensive experiments in autoencoder. The number of iterations is an important parameter in transfer learning. We set the number of iterations to 50. We will discuss the impact of different parameter settings on the discussion in Sect. 5.

5 Experimental results and statistical test

5.1 Experimental results

Common classification methods are logistic regression (Hosmer et al., 2013), random forest, Bayesian network, and decision tree. Whether there is a difference in these classification methods in software defect prediction is also a significant problem. Lessmann et al. have explored this problem in the early days. They did experiments on the NASA datasets, and they compare the research performance of 22 different classification methods. The results showed that the difference is not significant (Ghotra et al., 2015). Later, other scholars did the corresponding research and got the same answer. However, Ghotra et al. (Lessmann et al., 2008) did get a different answer. We discussed the impact of the classifier on our experiments in Sect. 5. We choose the decision tree classifier as our classifier to experiment.

Next, we will start with the following three questions and compare the performance of our model with AUC value and g-measure value (in 5-A, 5-B, and 5-C).

A. RQ1. How does MHCPDP compare to existing HCPDP methods?

1) Motivation

When we make software defect prediction, we often encounter situations where the dataset is insufficient or the existing dataset is different from the target dataset. Although many methods have studied this, there are still some shortcomings in these methods. We compared our approach to two top-level heterogeneous defect prediction methods (CCA + and HDP methods), hoping to achieve better results while overcoming their shortcomings.

2) Methods

We choose AUC and g-measure to measure our experiments and comparative experiments. To simulate HCPDP scenario, we select the source project and the target project from different datasets. We selected 16 datasets as the target dataset. Since neither CCA + nor HDP involves any randomness, we only run it once and

Table 2 Hyper-parameter settings

Hyper-parameters	Values	Explanation
ENCODING_DIM_INPUT		Dimensions of the original dataset
ENCODING_DIM_OUTPUT	10	Dimension after dimensionality reduction
EPOCHS	50	It is defined as a single training iteration of all batches in the forward and backward propagation
BATCH_SIZE	258	It is the number of samples selected for one training

record the results. The former is the representative method of HCPDP. Please note that all ranges of the two indicators used in this work are between 0 and 1. For the CCA + method, when it is applied to some cross-project pairs, and the elements in the mapped target project data are all zero, we treat this situation as a failure. For the HDP method, when the matching scores of all feature pairs are below the threshold, we will not be able to select any features. In this case, we will also treat them as failures, and we will not do much research on failed experiments.

Table 2 shows the comparison between our method and the HDP method, as well as the CCA + method when selecting the AUC values and g-measure values.

3) Results

We selected sixteen datasets in AEEEM, NASA, MORPH and Relink, and Softlab as target datasets.

Firstly, according to Table 3, we see that our method is superior to HDP by comparing the AUC values. Our AUC values are all above 0.6, and some projects even reach 0.875. The average value of AUC is also above 0.712, while the HDP mean is only 0.700, which is slightly lower than our method. It can be seen that HDP does

Table 3 Comparison of AUC values and g-measure values between MHCPDP, HDP, and CCA + methods

Target	AUC			G-measure		
	HDP	CCA +	MHCPDP	HDP	CCA +	MHCPDP
EQ	0.776	0.738	0.718	0.616	0.673	0.630
ML	0.692	0.575	0.676	0.582	0.558	0.624
JDT	0.767	0.648	0.663	0.654	0.632	0.571
PDE	0.692	0.604	0.609	0.633	0.691	0.564
Apache	0.720	0.707	0.756	0.586	0.625	0.628
safe	0.837	0.762	0.812	0.672	0.721	0.708
zxing	0.650	0.611	0.700	0.546	0.557	0.528
arc	0.701	0.704	0.674	0.603	0.603	0.646
poi-1.5	0.706	0.612	0.781	0.604	0.604	0.786
Redktor	0.528	0.609	0.875	0.571	0.595	0.753
tomcat	0.737	0.617	0.640	0.681	0.585	0.631
xerces-1.2	0.497	0.530	0.747	0.464	0.447	0.552
pc1	0.754	0.716	0.737	0.642	0.641	0.586
pc4	0.681	0.678	0.714	0.635	0.626	0.612
cm1	0.720	0.661	0.604	0.593	0.604	0.524
mw1	0.745	0.697	0.687	0.656	0.661	0.701
average	0.700	0.654	0.712	0.609	0.608	0.622

not perform well in some datasets, such as xerces-1.2, whose AUC value is only 0.497 which is lower than the random guess. The results of our method are better, and AUC values are above 0.6. The AUC values of many projects are around 0.7, and the overall average is higher than HDP. Although the test results are not ideal on some projects, they are generally better than HDP.

Secondly, when comparing AUC values with CCA + method, our method achieves better performance. The CCA + method relies heavily on many features common between the source project and the target project. The source datasets we selected during the comparative experiment and our target datasets have many common features, so the experimental results are better. However, even if the CCA + method selects a large number of datasets between the source project and the target project, its AUC value is not very high: the average value is only 0.654, and the AUC value of the ML and xerces-1.2 items is about 0.5, and the highest AUC value is 0.716. The AUC value of most projects is above 0.6, and only a few of them reach 0.7 or more.

Thirdly, when comparing g-measure values, the mean of our method is higher than the CCA + and HDP methods. The maximum value and the minimum value are better than the first two methods. Most of them change from 0.6 to 0.65; only a few are slightly lower than 0.6, which is better than the former two as a whole.

Lastly, when we do experiments, there are not as many constraints as the first two methods, and the experiment success rate is higher than the first two methods. CCA + works better when the source and target projects have features that are more common. When there are only a few or no common features, the experimental results do not perform well. The HDP method must consider the consistency of the defect propensity of the source project and the target project when experimenting, but often, we do not know this propensity. In contrast, our method achieved better results than they did when we overcome the shortcomings of the two.

In summary, our method overcomes some shortcomings of HDP method and CCA + method in addition to achieving comparable performance with HDP.

B. RQ2. What is the result of MHCPDP compared with unsupervised learning?

1) Motivation

Due to insufficient source datasets, unsupervised learning methods are used to solve CPDP problems. However, there are also some shortcomings in unsupervised defect prediction. Firstly, in software defect prediction, many unsupervised methods require manual labeling, and the cost of manual labeling is too high. Secondly, unsupervised learning methods generally solve problems by cluster analysis, but cluster analysis methods also have certain defects. When doing software defect prediction, cluster analysis methods are based on the assumption that the value of the metric element of the defective module has a higher value than the metric element of the non-defective module. However, we encountered the problem may not be met, and the effectiveness of the unsupervised method may not be guaranteed.

Therefore, HCPDP research is still essential. Next, we compare our method with excellent unsupervised methods.

Table 4 Comparison of AUC values and g-measure between MHCPDP, UDP, and WPDP methods

Target	AUC			G-measure		
	WPDP	UDP	MHCPDP	WPDP	UDP	MHCPDP
EQ	0.801	0.737	0.718	0.670	0.563	0.630
ML	0.710	0.630	0.676	0.630	0.588	0.624
JDT	0.817	0.733	0.663	0.741	0.636	0.571
PDE	0.731	0.646	0.609	0.615	0.610	0.564
Apache	0.757	0.0.754	0.756	0.634	0.582	0.628
safe	0.829	0.773	0.0.812	0.577	0.645	0.708
zxing	0.626	0.664	0.700	0.511	0.514	0.528
arc	0.726	0.615	0.674	0.600	0.591	0.646
poi-1.5	0.717	0.720	0.781	0.650	0.532	0.786
Redktor	0.719	0.489	0.875	0.600	0.580	0.753
tomcat	0.814	0.725	0.640	0.677	0.740	0.631
xerces-1.2	0.504	0.456	0.747	0.510	0.405	0.552
pc1	0.814	0.693	0.737	0.636	0.612	0.586
pc4	0.850	0.681	0.714	0.712	0.610	0.612
cm1	0.741	0.675	0.604	0.595	0.586	0.524
mw1	0.726	0.680	0.687	0.651	0.645	0.701
average	0.743	0.667	0.712	0.626	0.590	0.622

2) Methods

The unsupervised learning method CLAML proposed by Nam and Kim et al. It is an automatic method to serve the process of expert annotation in unsupervised learning methods (Nam & Kim, 2015). Their method is the leader in unsupervised learning methods in the field of software defect prediction. One of the best ways, we use AUC values and g-measure values to evaluate the performance of the CLAML model and the MHCPDP model.

3) Results

From Table 4, we can see that our method is better than UDP when comparing AUC values and g-measure values.

Firstly, as far as AUC is concerned, our MHCPDP has a higher value than the UDP method in the prediction of 12 target datasets, and the overall mean value is much higher than the UDP method. The minimum value is much higher than the UDP method.

Secondly, as far as g-measure is concerned, our MHCPDP has a higher value than the UDP method in the prediction of 11 target datasets, and the overall mean value is much higher than the UDP method. The minimum value is much higher than the UDP method.

Although there is no heterogeneous problem with unsupervised learning methods, still many other problems still affect the accuracy of the SDP model. Therefore, it is not feasible to solve heterogeneous defect prediction only by unsupervised learning methods. Although our method has its drawbacks, the effect is much better than the UDP method. We will continue to improve it in the future and hope to get a better model.

C. RQ3. What is the result of MHCPDP compared with WPDP?

1) Motivation

In the WPDP scheme, some labeled entities in the target project are used to train the prediction model, and the generated model is used to predict the class labels of other entities. Previous studies have shown that if there is enough labeled training data available, the performance of WPDP will be improved (Lessmann et al., 2008). However, we often encounter the problem of insufficient datasets, that is, we may not have enough data to train the target datasets. Therefore, we often use CPDP to predict software defects, but compared with WPDP, the performance of the previous CPDP method is usually not good enough. Therefore, we are interested in whether our method HDA has better or comparable results than WPDP in terms of two performance indicators.

2) Methods

For WPDP, we randomly select 50% entities in the target datasets as the training datasets and the remaining entities as the test datasets. Since the random selection process for the training datasets and test datasets may affect the prediction performance, we repeat the process 30 times to obtain the average indicator values.

Then, we also select the AUC value and g-measure for performance evaluation. For the parameter settings of these classifiers, we follow the previous studies (Xu et al., 2018).

3) Results

Firstly, the AUC value of our method has nine datasets with AUC values above 0.7 and seven at 0.6 or higher. The AUC value of WPDP has fourteen datasets with AUC values above 0.7, one at 0.6 or above, and one at around 0.5.

Secondly, as far as g-measure is concerned, our MHCPDP has a higher value than the WPDP method in the prediction of seven target datasets, and the overall mean value is close to the WPDP method. The minimum value is much higher than the WPDP method.

WPDP is based on the same dataset to achieve such an effect, and our dataset is based on heterogeneous conditions; the source datasets of the two are very different.

In summary, compared with WPDP methods, our method still has performance comparable with WPDP when implementing heterogeneous defect prediction.

5.2 Statistical test

To statistically analyze the performance values in Tables 3 and 4, we perform the non-parametric Friedman test with the Nemenyi's post hoc test (DemSar, 2006) at significant level 0.05 over the datasets, because the results of the Friedman test show that the p values on all two indicators are all less than 0.05. The Friedman test evaluates whether there exist statistically significant differences among the average ranks of different methods. Since Friedman test is based on performance ranks of the methods, rather than actual performance values, therefore it makes no assumptions on the distribution of performance values and is less susceptible to outliers (Xu et al., 2018).

Use Friedman test to determine whether these algorithms have the same performance. If they are same, their average ordinal values should be equal. Suppose k algorithms on N

datasets, let r_i denote the average order value of the i th algorithm. In order to simplify the discussion, we do not consider the halving value for the time being, then r_i obeys the normal distribution and its mean and variance are $\frac{k+1}{2}$ and $(k^2 - 1)/12$; variable is $\tau_{x^2} = \frac{k-1}{k} \times \frac{12N}{k^2-1} \sum_{i=1}^k \left(r_i - \frac{k+1}{2}\right)^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^k r_i^2 - \frac{k(k+1)^2}{4}\right)$; when both k and N are large, it obeys the x^2 distribution with $k - 1$ degrees of freedom.

If the hypothesis that “all algorithms have the same performance” is rejected, it means that the performance of the algorithms is significantly different. At this time, a “post hoc test” is needed to further distinguish the algorithms. The commonly used algorithm is Nemenyi’s post hoc test. Nemenyi’s post hoc test calculates the critical range of average ordinal difference that is $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$.

Table 5 shows the commonly used q_α values when $\alpha=0.05$ and 0.1 .

If the difference between the average sequence values of the two algorithms exceeds the critical value range CD, the assumption that “the performance of the two algorithms is same” is rejected with the corresponding confidence.

As we can see from Fig. 2, the horizontal axis is the average order, and the vertical axis is each algorithm. Each algorithm uses points to represent the average order, and the horizontal line segment with the point as the center represents the size of the critical value range. If the horizontal lines of the two algorithms overlap, it means that there is no significant difference between the two algorithms. According to Friedman test with the Nemenyi’s post hoc test, we sorted the order value for each method and calculated the average order value.

Figure 2 shows the comparison of MHCPDP against HDP, CCA+, UDP, and WPDP with Friedman test and Nemenyi’s post hoc test on the AUC values and g-measure values. Firstly, when HDP, CCA+, and MHCPDP are compared, in terms of AUC values, HDP has no significant difference compared with MHCPDP, while MHCPDP has a significant difference compared with CCA+, and in terms of g-measure values, MHCPDP, HDP, and CCA+ have no significant differences. Secondly, when WPDP, UDP, and MHCPDP are compared, in terms of AUC values and g-measure values, MHCPDP has no significant difference compared with WPDP, while MHCPDP has a significant difference compared with UDP.

We have known that our method is significantly different from these baseline methods, but how big the difference is depends on the effect size test.

The effect size refers to the difference caused by factors, and it is an index to measure the size of the treatment effect. Unlike the significance test, these indicators are not affected by the sample size. It represents the size of the difference between the overall mean under different treatments and can be compared between different studies. Mean difference, variance analysis explanation ratio, and regression analysis explanation ratio need to be described by effect size. The effect size is not affected by the sample size. When the sample size is significant, it is necessary to report the size of the effect. We

Table 5 The commonly used q_α values when $\alpha=0.05$ and 0.1

α	Number of algorithms K								
	2	3	4	5	6	7	8	9	10
0.05	1.960	2.344	2.569	2.728	2.850	2.949	3.031	3.102	3.164
0.1	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

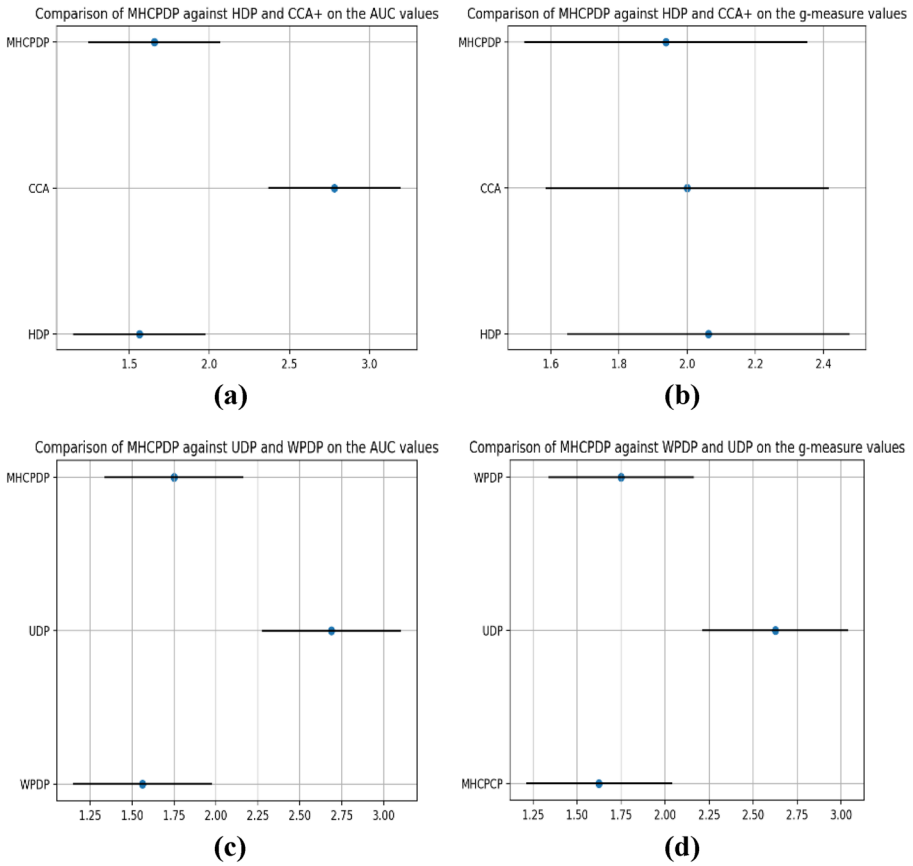


Fig. 2 Comparison of MHCPDP against HDP, CCA+, UDP, and WPDP with Friedman test and Nemenyi's post hoc test on the AUC values and g-measure values

choose Cohen's *d* values. The greater the Cohen's *d* value, the greater the difference. The calculation formula of the Cohen's *d* is as follows:

$$\text{Cohen's } d = \frac{M_1 - M_2}{\sqrt{\frac{SD_1^2 - SD_2^2}{2}}}$$

where M_1 and M_2 are the average values of two sets of comparative experiments. SD_1 and SD_2 are the standard deviations of two sets of comparative experiments. The results are shown in Table 6 below:

Table 6 The values of Cohen's *d*

MHCPD vs. baselines	Cohen's <i>d</i>	
	G-measure	AUC
MHCPDP vs. WPDP	0.060	0.300
MHCPDP vs. UDP	0.460	0.578
MHCPDP vs. HDP	0.217	0.156
MHCPDP vs. CCA+	0.223	0.877

From the above point of view, our method MHCPDP and HDP are both effective methods to conduct defect prediction on heterogeneous datasets. To sum up, our method MHCPDP achieves competitive results compared with CCA+ and HDP on most datasets in terms of the two indicators.

6 Discussion

6.1 How much time does it take for the MHCPDP methods and CCA+ methods?

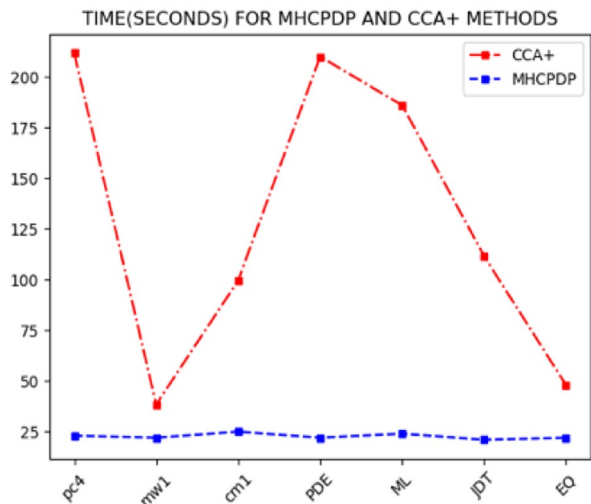
In addition to comparing the effectiveness of our method in the experiment, we also evaluated the efficiency of the method. Because our method and CCA+ method involve multiple projects as source datasets, we specially selected the CCA+ method as the reference object. The time we compare is the time of instance selection in MHCPDP and the time of obtaining the mapped project data in CCA+.

Our method involves the time-consuming extraction of the example selection process of the autoencoder. We have specially experimented with this, and the time that spent the experiment is kept at about lower level. The time spent by CCA+ is to obtain the mapped project data, so we select several items to measure the time spent on CCA+. As shown by Fig. 3, the time spent by CCA+ is about a few minutes. When the source project is a large-scale dataset, the time may be longer, which is very inconvenient for the actual operation, but our method can adapt well to large-scale datasets.

6.2 How different classifiers affect the performance of MHCPDP?

In MHCPDP, we choose the decision tree (DT) classifier as the basic prediction model. In order to explore the impact of different classifiers on the performance of MHCPDP, we choose three classifiers that are Naive Bayes (NB), logistic regression (LR), and random

Fig. 3 The time (s) for MHCPDP and CCA+



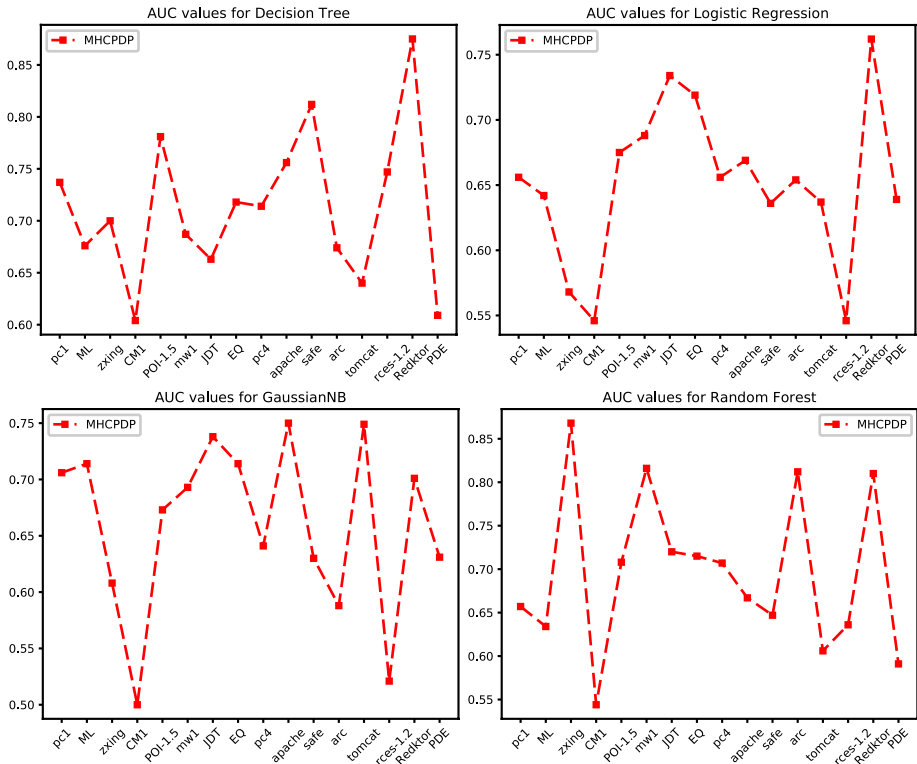


Fig. 4 The impact of different classifiers on AUC values

forest (RF) and compare the AUC value and g-measure value of these classifiers. Figures 4 and 5 depict the results of our experiments.

We can see that several classifiers have achieved good results in terms of AUC value, but the overall performance of decision tree is better. On the g-measure value, random forest and decision tree perform are better, and the performance of Naive Bayes and logistic regression are relatively poor.

Figure 6 shows the performance comparison of DT against RF, LR, and NB with Friedman test and Nemenyi’s post hoc test on the AUC values and g-measure values. By comparing g-measure and AUC values of DT, RF, LR, and NB in Fig. 6, DT has less overlap with the other three classifiers, and the average order value of DT is less than other three classifiers, it indicates that DT has a significant difference, and DT is better than the other three classifiers.

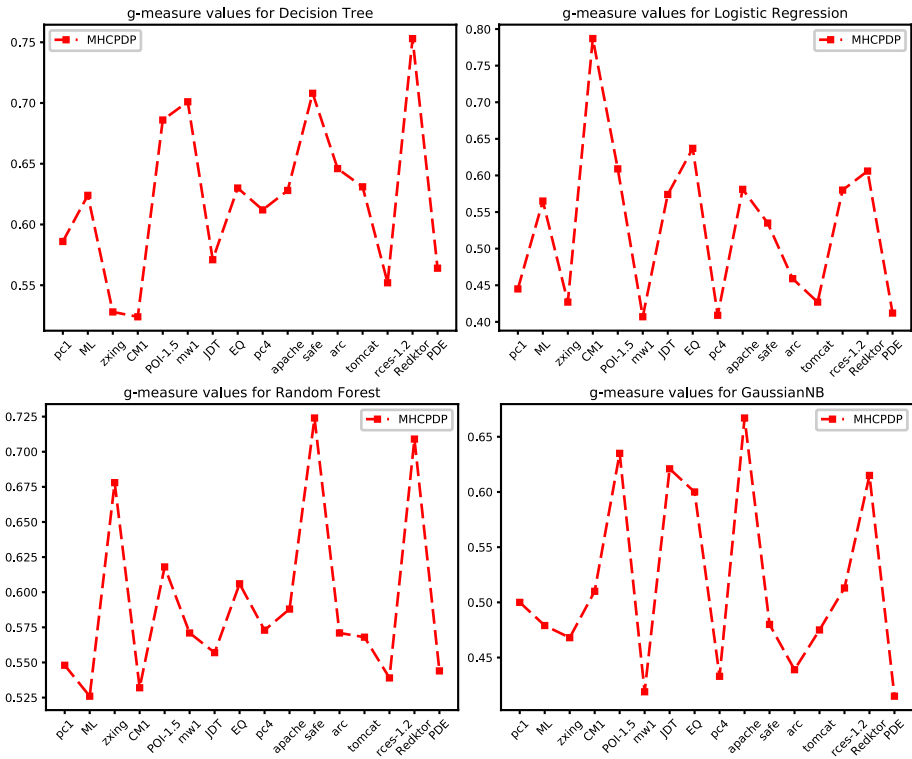


Fig. 5 The impact of different classifiers on g-measure values

6.3 How does the parameter affect the performance of MHCDDP?

From Figs. 7 and 8, we can find that the parameters have a certain influence on our experiment. How to adjust the parameters is that our method shows better performance is our future research topic.

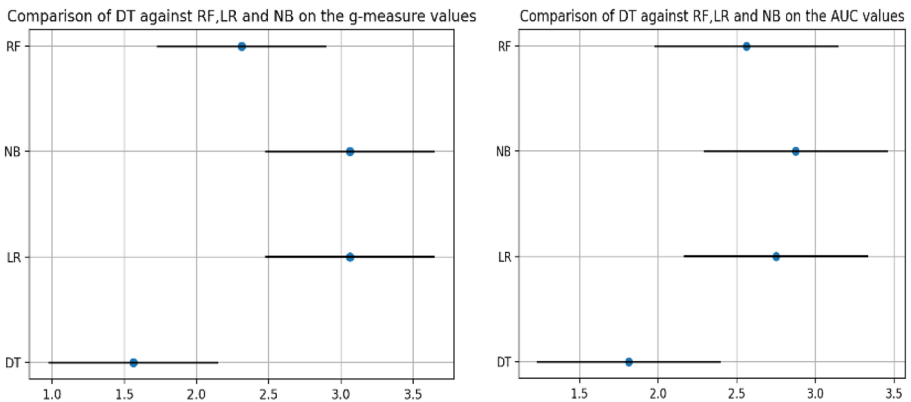


Fig. 6 Comparison of DT against RF, LR, and NB with Friedman test and Nemenyi’s post hoc test on the AUC values and g-measure values

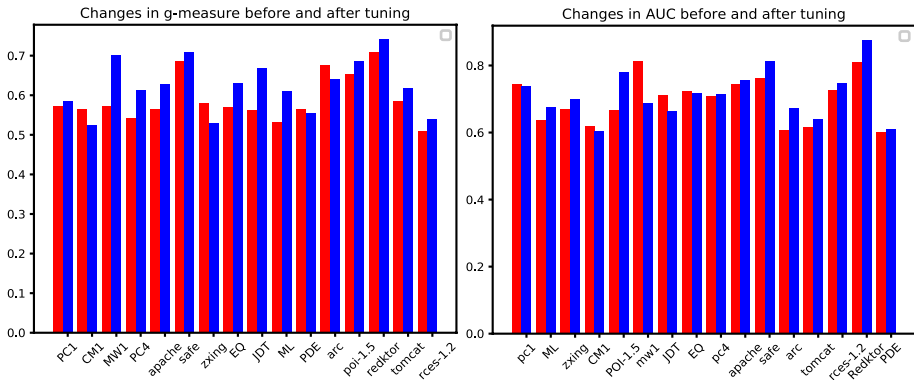


Fig. 7 The effect of x on g-measure values and AUC values

6.3.1 How does the parameter x affect the performance of MHCPDP?

The parameter x is defined as a single training iteration of all batches in the forward and backward propagation. We select $x=40$ in our experiment. Here, we evaluate the influence of different x values on the experimental results by selecting $x=50$ for comparison. The blue in the figure represents the x value selected by our experiment.

6.3.2 How does the parameter y affect the performance of MHCPDP?

The parameter y is the number of samples selected for one training. We select $y=128$ in our experiment. Here, we evaluate the influence of different y values on the experimental results by selecting $y=150$ for comparison. The blue in the figure represents the y value selected by our experiment.

Overall, the adjustment of the parameters has a certain impact on our method. We decided to choose the optimal parameters as our work focus to continue research.

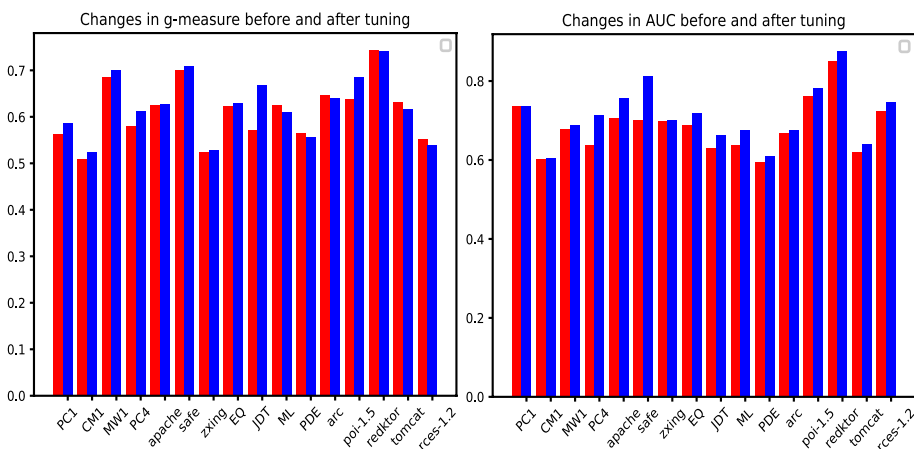


Fig. 8 The effect of y on g-measure values and AUC values

7 Threats to validity

In this section, we discuss some of the threats to the validity of our work.

7.1 Threats to construct validity

The construction validity is based on the selection of performance indicators. In this paper, we selected two commonly used indicators, g-measure and AUC. These performance indicators make our method very good with existing excellent. The comparison of the methods is very helpful for us to evaluate the performance of our method.

7.2 Threats to internal validity

The internal threat is mainly related to the implementation of the method and the parameter setting. The baseline method we selected may have some differences due to the unknown author's parameter settings and some specific information.

However, we reduce this threat by adjusting the parameter settings to choose the optimal combination of parameters.

7.3 Threats to external validity

External threats are primarily the application of our approach to real-world applications. We reduce this threat by selecting five publicly available datasets.

8 Conclusion

This is a challenge for heterogeneous cross-project software defect prediction due to the low correlation between features selected between different datasets. In the face of this challenge, we have proposed our own method MHCPDP. First, we analyzed many previous studies and found that many methods simply delete redundant features and do not mind the deeper meaning behind the data to filter the features. We choose to use the autoencoder to mind the data deeper.

Secondly, we modified the autoencoder algorithm for instance selection, which can narrow the gap between the source project and the target projects. Thirdly, we introduce multi-source transfer learning algorithm into heterogeneous software defect prediction, which not only solves the impact of negative transfer on transfer learning but also can carry out multi-source learning. Multiple source project datasets can solve the problem of insufficient training datasets. This idea also comes from transfer learning. In the future, we will continue to explore more ideas in transfer learning and apply them to software defect prediction in order to develop more and better methods. To confirm the effectiveness of our approach, we conducted a number of experiments and compared them with some of the top methods in the UDP, WPDP, and HDP directions. The experimental results show that our method has achieved better results.

Although our approach has achieved good results, there are still some unresolved issues. For example, class imbalance is also a very important topic in software defect prediction; we simply adopted the smote method. In addition to the class imbalance problem, we have not considered the noise problem in the dataset. We will continue to study these experimental results in the future.

Appendix

We provide the datasets and the source code of the proposed approach that are used to conduct this study at <https://github.com/SE-CQU/sdp>.

Funding This work was supported in part by National Key Research and Development Project under grant 2019YFB1706101 and in part by the Science-Technology Foundation of Chongqing, China, under grant cstc2019jscx-mbdx0083.

References

- Briand, L. C., Melo, W. L., & Wust, J. (2002). Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28(7), 706–720.
- Chen, H., Jing, X. Y., Li, Z., Wu, D., Peng, Y., & Huang, Z. (2020). An empirical study on heterogeneous defect prediction approaches. *IEEE Transactions on Software Engineering*, pp. 1–1. <https://doi.org/10.1109/TSE.2020.2968520>
- D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *Proceedings of the Working Conferences on Mining Software Repositories*, pp. 31–41.
- D'Ambros, M., Lanza, M., & Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4), 531–577.
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 1–30.
- Du, X., Zhou, Z., Yin, B., & Xiao, G. (2020). Cross-project bug type prediction based on transfer learning. *Software Quality Journal*, 39–57.
- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27, 861874.
- Ghotra, B., McIntosh, S., & Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performances of defect prediction model. In *Proceedings of the International Conference on Software Engineering*, pp. 789–800.
- He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19(2), 167–199.
- Herzig, K., Just, S., Rau, A., & Zeller, A. (2013a). Predicting defects using change genealogies. *Proceedings In 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)* pp.370–381.
- Herzig, K., Just, S., Rau, A., & Zeller, A. (2013b). Classifying code changes and predicting defects using change genealogie.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural network. *Science*, 313, 504–506.
- Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression*, 398.
- Hosseini, Seyedrebar, Turhan, et al. (2019). A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, 45, pp. 111–147.
- Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5), 561–595.
- Jing, X., Wu, F., Dong, X., Qi, F., & Xu, B. (2015). Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp.496–507.

- Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the International Conference on Predictive Models in Software Engineering*, pp. 1–10.
- Knapp, T. R. (1978). Canonical correlation analysis: A general parametric significance-testing system. *Psychological Bulletin*, 85, 410.
- Lessmann, S., Baseens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34, pp. 485–496.
- Lee, T., Nam, J., Han, D., Kim, S., & In, H. P. (2011). Micro interaction metrics for defect prediction. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 311–321).
- Li, Z., Jing, X. Y., Wu, F., et al. (2017). Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction [J]. *Automated Software Engineering*.
- Li, Z., Jing, X. Y., Zhu, X., et al. (2017). On the multiple sources and privacy preservation issues for heterogeneous defect prediction [J]. *IEEE Transactions on Software Engineering*, 1–1.
- Li, Z., Jing, X. Y., Zhu, X., et al. (2019). Heterogeneous defect prediction with two-stage ensemble learning [J]. *Automated Software Engineering*, 26(3), 599–651.
- Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54(3), 248–256.
- Menzies, T., Greenwald, J., & Frank, A. (2006). Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1), 2–13.
- Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 33, pp 2–13.
- Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2015). Heterogeneous defect prediction. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* 508–519: ACM.
- Nam, J., & Kim, S. (2015). CLAML: Defect prediction on unlabeled datasets. In *Proceedings of the International Conference on Automated Software Engineering*, pp. 452–463.
- Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. *Proceedings International Conference on Software Engineering Transfer defect learning. Proceedings International Conference on Software Engineering*, pp. 382–391.
- Pan, S. J., Tsang, I. W., Kwork, J. T., & Yang, Q. (2011). Domain adaption via transfer component analysis. *IEEE Transactions on Neural Networks*, 22, 199–210.
- Pan., S. J., & Yang., Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22, pp. 1345–1359.
- Peters, F., & Menzies, T. (2012). Privacy and utility for defect prediction: Experiments with morph. In *2012 34th International Conference on Software Engineering (ICSE)* pp. 189–199. IEEE.
- Pingclasai, N., Hata, H., & Matsumoto, K. I. (2013). Classifying code changes and predicting defects using change genealogies. *Proceedings 20th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 13–18.
- Rahman, F., Posnett, D., & Devanbu, P. (2012). Recalling the " imprecision" of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (pp. 1–11).
- Tan, B. Y. et al. (2017) Distant domain transfer learning, In *2013 35th International Conference on Software Engineering (ICSE)* pp. 382–391: IEEE.
- Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., & Matsumoto, K. (2015). The impact of mislabelling on the performance and interpretation of defect prediction models. *Proc IEEE International Conference on Software Engineering (ICSE)*, pp. 812–823.
- Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2008). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5), 540–578.
- Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5), 540–578.
- Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011). Relink: Recovering links between bugs and changes. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering Szeged*, pp. 2–13.
- Xu, Z., Xuan, J., Liu, J., & Cui, X. (2016). MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. *Proceedings IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*.
- Xu, Z., Yuan, P., & Zhang, T. (2018). HDA: Cross-project defect prediction via heterogeneous domain adaptation with dictionary learning. *IEEE Access*, 6, 57597–65761.

- Yao, Y., & Doretto, G. (2010). Boosting for transfer learning with multiple sources. *IEEE conference on computer vision and pattern recognition*, pp. 1855–1862.
- Zhang, F. Z., Luo P., He Q., & Shi Z. Z. (2015). Survey on transfer, pp.26–39.
- Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *Proceeding of the 7th Joint Meeting European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 91–100.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Jie Wu received the bachelor's degree in computer science and technology from Tianjin University of Technology, in 2018. She is currently a graduate student in Chongqing University. Her current research interests include software defect prediction, machine learning, and data mining.



Yingbo Wu received the Ph.D. degree from the College of Mechanical Engineering, Chongqing University, Chongqing, China, in 2012. From Jun. 2015 to Oct. 2015, he was a visiting scholar with the School of Engineering and Applied Science, University of Cincinnati, USA. Since 2003, he has been with the School of Software Engineering, Chongqing University, where he is currently a full professor and the vice director of the Department of Software Engineering. His research interests include software defect prediction and service computing.



Nan Niu (M'08–SM'13) received the B. Eng. degree from the Beijing Institute of Technology in 1999, the M.Sc. degree from the University of Alberta in 2004, and the Ph.D. degree from the University of Toronto in 2009, all in computer science. He is currently an associate-professor with the Department of Electrical Engineering and Computer Science, University of Cincinnati, USA. His research interests include software requirements engineering, information seeking in software engineering, and human-centered computing. He received the U.S. National Science Foundation Faculty Early Career Development Award, the IEEE International Requirements Engineering Conference Best Research Paper Award in 2016, and the Most Influential Paper Award in 2018.



Min Zhou received the bachelor's degree in software engineering from Nanchang University, in 2018. She is currently a graduate student in Chongqing University. Her current research interests include service composition and resource optimization.