PAPER

Unsupervised Deep Domain Adaptation for Heterogeneous Defect Prediction

Lina GONG[†,††], **Shujuan JIANG**[†,†††a)], ***Nonmembers***, **Qiao YU**[††††], ***Member***, ***and*** **Li JIANG**[†,†††], ***Nonmember***

SUMMARY Heterogeneous defect prediction (HDP) is to detect the largest number of defective software modules in one project by using historical data collected from other projects with different metrics. However, these data can not be directly used because of different metrics set among projects. Meanwhile, software data have more non-defective instances than defective instances which may cause a significant bias towards defective instances. To completely solve these two restrictions, we propose unsupervised deep domain adaptation approach to build a HDP model. Specifically, we firstly map the data of source and target projects into a unified metric representation (UMR). Then, we design a simple neural network (SNN) model to deal with the heterogeneous and class-imbalanced problems in software defect prediction (SDP). In particular, our model introduces the Maximum Mean Discrepancy (MMD) as the distance between the source and target data to reduce the distribution mismatch, and use the cross-entropy loss function as the classification loss. Extensive experiments on 18 public projects from four datasets indicate that the proposed approach can build an effective prediction model for heterogeneous defect prediction (HDP) and outperforms the related competing approaches.

key words: *heterogeneous defect prediction, neural networks, maximum mean discrepancy, class-imbalance*

1. Introduction

Software defect prediction (SDP) is a research field that builds effective models for detecting the largest number of defective software modules using sufficient historical data [1], [2], which has attracted a lot of attention from researchers and practitioners. In recent years, most existing prediction models are built from historical data of the same project, which is named as within-project defect prediction (WPDP) [3]–[8]. Usually, WPDP can provide high quality results when there are sufficient within-project data. However, in practice, there are fewer labeled historical instances at the beginning stage of a project, and this hinders the application of WPDP.

To solve this problem, cross-project defect prediction (CPDP) using data from other projects has been proposed [1], [9]. In recent years, many CPDP approaches have been developed including NN-filter [9], VCB-SVM [11] and TCBoost [12]. Existing CPDP approaches demand that data from other projects and testing project should have the same metric sets.

However, there are no sufficient data with the same metrics for a new company in practice. In this scenario, heterogeneous defect prediction (HDP) approaches are proposed [13]–[16]. HDP is to detect the defect-prone modules of a project by using data collected from other companies with heterogeneous metric sets. Since training data have different metric set from testing data, the traditional machine learning methods can not be used directly [14]. Furthermore, the data of software projects are class imbalanced, which hinders the performance of classifiers [11], [12], [36]. Most exsiting HDP models do not consider the class imbalance problem.

In this study, we address problems associated with HDP, and propose an unsupervised deep domain adaptation approach for HDP. In our approach, we design a simple neural network (SNN) model whose loss function simultaneously considering the classification error from labeled training data and the distance error between the training and testing data. In summary, our paper brings the following contributions:

- To deal with the heterogeneous problem for HDP without labeled target training data, a new neural network architecture is designed, which uses an adaptation layer along with a distance measure to automatically learn a representation jointly trained to optimize for the distribution mismatch between training data and testing data.
- To deal with the class imbalance problem, cost-sensitive label is introduced into the cross-entropy loss function that is as the classification error in the new neural network model. The defective instances and non-defective instances have different misclassification costs.
- We explore the Maximum Mean Discrepancy (MMD) and correlation alignment (CORAL) as a regularization embedded in the back-propagation training, and observe that the Maximum Mean Discrepancy (MMD) is better to reduce the distribution mismatch for software data.

We conduct empirical studies on 18 open source projects from four companies including NASA [17], [18], AEEEM [19], SOFTLAB [17] and ReLink [20].

Manuscript received August 17, 2018.
Manuscript revised October 22, 2018.
Manuscript publicized December 5, 2018.

[†]The authors are with School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, China.
[††]The author is with Department of Information Science and Engineering, Zaozhuang University, Zaozhuang, China.
[†††]The authors are with Engineering Research Center of Mine Digitalization of Ministry of Education, Xuzhou, China.
[††††]The author is with School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, China.
a) E-mail: shjjiang@cumt.edu.cn

DOI: 10.1587/transinf.2018EDP7289

Experimental results indicate that our approach can outperform several state-of-the-art WPDP, CPDP and HDP approaches.

The structure of this paper is as follows: Sect. 2 reviews and analyzes the related work of defect prediction model; In Sect. 3, we describe our unsupervised deep domain adaptation approach in detail; the setting of our empirical study and results are presented in Sect. 4; Sect. 5 describes the discussions and the threats to validity of our approach are introduced in Sect. 6; We conclude the paper and provide insights for future work in Sect. 7.

2. Related Work

In this section, we briefly review the CPDP approaches, HDP approaches and class imbalance learning approaches for SDP.

2.1 CPDP Approaches

Cross-project defect prediction (CPDP) is to predict defect-prone modules in a project using historical data from other projects [21]–[27]. Recently researches have developed many CPDP approaches. Some approaches aim to choose the suitable training instances for the testing data. Turhan et al. [9] selected similar instances in different projects as training set, and used K-Nearest Neighbor (KNN) as classifier to train. Seyedrebvar et al. [28] proposed a search-based data selection approach. Firstly, they utilized NN-filter to sample instances from training data as confirmation set. Then they employed genetic algorithm to choose the training set based on which they predicted the testing data.

Some CPDP approaches introduce transfer learning approaches into the CPDP. Pan et al. [29] found that the different distributional characteristics between training dataset and test dataset might result in too low prediction results. To solve this problem, they proposed transfer defect learning which discovered the potential correlation between the training dataset and testing dataset by transfer component analysis (TCA). Nam et al. [10] also used TCA to exploit the common feature space of different projects. They transfered useful information to reduce distribution differences, and selected the optimal normalization strategy for CPDP. Ma et al. [30] proposed the model of Transfer Naive Bayes (TNB) using the weighted training data to construct the classifier, and this model significantly improved the defect prediction performance. Ryu et al. [12] proposed a transfer cost-sensitive boosting approach (TCSBoost) for CPDP with few labeled target data. Yu et al. [31] proposed an effective solution for Cross-company defect prediction. They firstly provided a novel semi-supervised clustering-based data filtering approach to filter out irrelevant cross-company instances. Then, they introduced multi-source TrAdaBoost algorithm into CCDP. However, except TCSBoost existing methods take no account of the class imbalance problem.

In addition, existing CPDP approaches demand that training data should have the same metrics to the testing data. In practice, the number of same metrics between training and testing data may be very small, which would obtain undesirable prediction results based on these approaches.

2.2 HDP Approaches

Recently, some heterogeneous defect prediction (HDP) approaches have been proposed to address the heterogeneous metrics problems. Jing et al. [13] utilized unified metric representation (UMR) and CCA-based transfer learning approach to solve the heterogeneous metrics problems. Experimental results indicated that their approach can make the distribution of testing data similar to that of training data.

Nam et al. [14] firstly applied metric selection to training data to remove redundant and irrelevant metrics. Then, they matched up the training and testing metrics based on metric similarity. Recently, Cheng et al. [16] presented a cost-sensitive correlation transfer support vector machine (CCT-SVM) method to deal with the class imbalance problem for HDP.

2.3 Class-Imbalance Learning Approaches

Software data have more non-defective instances than defective instances, that is class imbalance problem. Some researches have studied this problem. Wang et al. [32] used different approaches to deal with the imbalance datasets, including resampling, thresholding and integration methods. Jing et al. [33] used the sparse representation to eliminate the effects of imbalance datasets. Limsettho et al. [34] proposed CDE-SMOTE approach to solve the class imbalance and distribution mismatch problems. They firstly employed class distribution estimation (CDE) to estimate the class distribution of the target project, then SMOTE was used to modify the class distribution of training data into the reverse of the approximated class distribution of the target project. Liu et al. [35] proposed two level cost sensitive methods to solve the imbalance problem in software defect prediction for the first time. Bennin et al. [36] proposed an efficient synthetic oversampling approach based on the chromosomal theory of inheritance (MAHAKIL) to balance the class distribution. MAHAKIL interpreted two distinct sub-classes as parents and generated a new instance that inherited different traits from each parent. Their experiments indicated that MAHAKIL improved the performance for all the models.

In addition, Seiffert et al. [37] made a detailed study of classification method and sampling method under the imbalance datasets. They believed that the classification results would be greatly improved if the datasets were sampled before trained. Bennin et al. [38] also assessed the impact of resampling approaches. They analyzed six resampling approaches on 40 releases of 20 open source projects, and the experimental results indicated that: (1) there were statistical differences between the prediction results with and without resampling methods, but resampling could not improve the AUC values; (2) RUS and Borderline-SMOTE were proved to be the more stable resampling approaches

among the studied resampling; (3) The performance of resampling approaches depended on the imbalance ratio, that is to say, oversampling approaches should aim at generating relevant and informative instances.

Different from the above class imbalance learning methods, we integrate the cost-sensitive learning technique into unsupervised deep domain adaptation for HDP.

3. Our Approach

In this section, we firstly describe the notation that will be used in this study. Then, we describe the preprocessing software metrics for training and testing data. Lastly, we provide our new neural network approach for HDP.

3.1 Notation

At the beginning stage of a project in a new company, the number of available defect data is not adequate for a prediction model, so the data from other open source projects can be used for this project. This model can be defined as follows:

Supposed that D_s is the labeled instance space from other projects, D_t is the unlabeled instance space from the testing project. D_s contains a data set $X_s = \{\chi_s{}^1, \chi_s{}^2, \cdots, \chi_s{}^N\}$ and a label set $Y_s = \{y_s{}^1, y_s{}^2, \cdots, y_s{}^N\}$, where $\chi_s{}^i$ denotes the i^{th} module in X_s, $y_s{}^i$ is the corresponding label, and N is the number of modules of X_s. D_t contains unlabeled data set $X_t = \{\chi_t{}^1, \chi_t{}^2, \cdots, \chi_t{}^M\}$, where $\chi_t{}^i$ denotes the i^{th} module in X_t and M is the number of modules of X_t. Each instance in X_s is represented as $\chi_s{}^i = [a_s{}^{i1}, a_s{}^{i2}, \ldots, a_s{}^{id_s}]$, where $a_s{}^{ij}$ refer to the value of j metric in the instance $\chi_s{}^i$, and d_s is the number of metrics of χ_s in D_s. Each instance in X_t is represented as $\chi_t{}^i = [a_t{}^{i1}, a_t{}^{i2}, \ldots, a_t{}^{id_t}]$, where $a_t{}^{ij}$ refer to the value of j metric in the instance $\chi_t{}^i$, and d_t is the number of metrics in D_t. Note that the metrics in X_s and X_t are different and $d_s \doteq d_t$.

3.2 Preprocessing Software Metrics

The first layer of the neural network is the metrics. However, training data have different metrics from testing data for HDP, which can not use the neural network directly. So we should firstly preprocess the software metrics.

Jing et al. [13] proposed the unified metric representation (UMR) to solve the different metrics sets, which is used in our paper. Supposed that $A_s = [a_s{}^1, a_s{}^2, \ldots, a_s{}^{d_s}]$ is the metric space of training data X_s. $A_t = [a_t{}^1, a_t{}^2, \ldots, a_t{}^{d_t}]$ is the metric space of testing data X_t. The unified metric representation contains three parts. One part is the same metrics between A_s and A_t, whose number is d_c. Another part is the specific metrics except the same metrics in A_s. The third part is the specific metrics except the same metrics in A_t. The number of the UMR is $d_s + d_t - d_c$. Figure 1 shows the construction of UMR.

The $\chi_s{}^i$ and $\chi_t{}^i$ can be converted as follows:

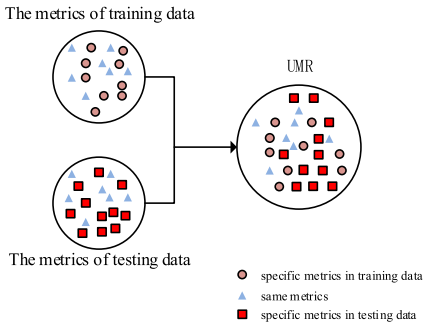

Fig. 1 The construction of UMR.

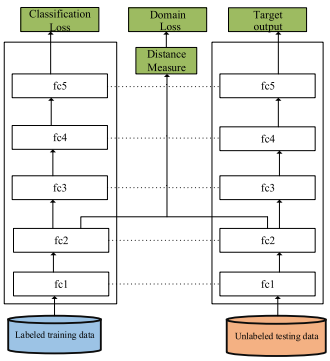

Fig. 2 The architecture of our neural network.

$$\bar{X}_s = \begin{bmatrix} X_s^c \\ X_s^s \\ 0_{(d_t - d_c)\times N} \end{bmatrix} and \quad \bar{X}_t = \begin{bmatrix} X_t^c \\ 0_{(d_s - d_c)\times M} \\ X_t^t \end{bmatrix} \tag{1}$$

3.3 Proposed Method

3.3.1 Our Architecture

Since the training data has different distribution from the testing data, directly training a neural network based on only the training data often leads to overfitting to the training data and causes the bad performance when predicting the testing label. Our motivation is to train a classifier on the training labeled data that can directly apply to the testing unlabeled data by learning a representation that minimizes the distance between training and testing distributions. Our architecture optimizes a simple neural network for both classification loss and domain invariance, which consists of a training and testing NN. The architecture is shown in Fig. 2.

In Fig. 2, the first two hidden layers fc1 and fc2 encourage the representations induced by X_s to be close enough wit the ones induced by X_t. The fc3, fc4 and fc5 are layers to learn a strong classification representation. Note that the fc3 and fc4 layers have a Batch Normalization (BN) between full connection and activation function, which can assure the activation values presented at each distribution interval.

3.3.2 Training Our Neural Network

We introduce a new neural network architecture to learn a

visual representation for both heterogeneous (between training and testing data) and a strong classification. So our approach not only minimize the distance between training and testing data, but also train a strong classifier. To meet the criteria, the loss defined in Eq. (2) should be minimized.

$$\iota = \iota_C(X_s, Y_s) + \lambda \times Distance(Q_s, Q_t) \tag{2}$$

Where $\iota_C(X_s, Y_s)$ denotes the classification loss on the labeled training data X_s, and $Distance(Q_s, Q_t)$ is the distance between the training data and testing data, Q_s and Q_t are the outputs of the second layer of our neural network for training and testing data respectively. The hyperparameter λ is a constant controlling the importance of distance contribution to the loss function.

To calculate classification loss, we use the cross-entropy loss function as $\iota_C(X_s, Y_s)$ that measures the similarity between the real label distribution and the predicted label distribution, as formulated by Eq. (3).

$$C = -\frac{1}{N} \times \sum_{i=1}^{N} \sum_{k=0}^{1} ([y_s^i]_k log[f(\chi_s^i)]_k) \tag{3}$$

Where l is the number of label that is two in our problem (defective and non-defective); k is the classified label (0:non-defective, 1:defective); N is the number of instances; y_s^i is the real label, and $f(\chi_s^i)$ is the predicted label obtained by softmax activation function in our approach.

In practice, a project has more non-defective modules than defective modules which leads the software data class-imbalanced. The classs imbalance results in severe class distribution skews. However, defective modules are more important than non-defective modules. To settle this problem, we introduce an appropriate weight into cross-entropy loss called weighted cross-entropy loss function, which is defined as

$$\iota_C(X_s, Y_s) = -\frac{1}{N} \times \sum_{i=1}^{N} \sum_{k=0}^{1} (weight[k] \times [y_s^i]_k log[f(\chi_s^i)]_k) \tag{4}$$

Where entry $weight[k]$ represents the weight associated with classifying the target instance (with true class k) as wrong label. In our case, more defective modules should be found. Thus, more weight is put on misclassifying defective instances that is $weight[1] > weight[0]$. In our approach, We set $weight[0] = 1$ and $weight[k] = \frac{n_0}{n_1}$, in which n_0 is the number of non-defective instances in D_s, and n_1 is the number of defective instances in D_s.

To calculate $Distance(Q_s, Q_t)$, we consider the Maximum Mean Discrepancy (MMD) [39]. The MMD is defined as followed:

$$MMD(Q_s, Q_t) = (\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1}^{N} k(q_s^i, q_s^j) + \frac{1}{M(M-1)} \sum_{i=1}^{M} \sum_{j=1}^{M} k(q_t^i, q_t^j) - \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} k(q_s^i, q_t^j))^{\frac{1}{2}} \tag{5}$$

Where $k(.,.)$ is a kernel function, such as Gaussian kernel, which can refer to [39] to see the detail explanation.

In addition, to validate the effect of the Maximum MeanDiscrepancy (MMD), we also consider the correlation alignment (CORAL) [40] as the $Distance(Q_s, Q_t)$. According to [40], the CORAL is defined as squared Euclidean distance between second order statistics, and is shown as followed:

$$\iota_{CORAL}(Q_s, Q_t) = \frac{1}{4d^2} \|C_s - C_t\|^2 \tag{6}$$

Where C_s and C_t are the covariance matrices, and are shown as follows:

$$C_s = \frac{1}{N-1}(Q_s^T Q_s - \frac{1}{N}(1^T Q_s)^T (1^T Q_s)) \tag{7}$$

$$C_t = \frac{1}{M-1}(Q_t^T Q_t - \frac{1}{M}(1^T Q_t)^T (1^T Q_t)) \tag{8}$$

The aim of our approach is to minimize the loss outlined in Eq. (2) by using the gradient descent optimization. Only the labeled training data are used to compute the classification loss, while training and testing data are both used to compute the domain confusion loss. Supposed that $U = [u_1, u_2, u_3, u_4, u_5]$ are the parameter matrices containing both weights and biases in each layer. For each epoch, the $\iota_C(X_s, Y_s)$ is minimized by adjusting u_1, u_2, u_3, u_4 and u_5 using the standard mini-batch back-propagation. The $Distance(Q_s, Q_t)$ is minimized by re-adjusting u_1 and u_2 with respect to the MMD gradient. The learning steps is described in Algorithm 1.

Table 1 The back-propagation learning algorithm of our neural network.

Algorithm1 The back-propagation learning algorithm of our neural network

Inputs: labeled instances from other projects D_S;
unlabeled instances from testing project D_T;
The number of epochs T;
The learning rate α;
The hyperparameter λ;

Outputs: The label of the testing instances

1: Initialization: U with small random real values, T, α and λ
2: For $t = 1, 2, \cdots, T$:
3: Update $U = [u_1, u_2, u_3, u_4, u_5]$ using the mini-batched gradient descent $\iota_C(X_s, Y_s)$
4: Update u_1, u_2 by the offline gradient descent as follow:
5: $u_i^{t+1} = u_i^t - \alpha\lambda \frac{\partial Distance(Q_s, Q_s)}{\partial u_i}$
6: Output the label of testing instances

4. Experiments

To evaluate our proposed SNN approach, we conduct large-scale experiments on 18 public projects from four companies including NASA, SOFTLAB, AEEEM and ReLink. The evaluation is centered around the follow three research questions:

- RQ1: Does SNN approach perform better than the

Table 2 The experimental datasets.

Dataset	Project	Number of metrics	Number of total instances	Number of defective instances	% of defective instances
NASA	CM1	37	327	42	12.84
	MW1	37	253	27	10.67
	PC1	37	705	61	8.65
	PC3	37	1077	134	12.44
	PC4	37	1458	178	12.21
AEEEM	EQ	61	324	129	39.81
	JDT	61	997	206	20.66
	LC	61	691	64	9.26
	ML	61	1862	245	13.16
	PDE	61	1497	209	13.96
SOFTLAB	AR1	29	121	9	7.44
	AR3	29	63	8	12.70
	AR4	29	107	20	18.69
	AR5	29	36	8	22.22
	AR6	29	101	15	14.85
ReLink	Apache	26	194	98	50.52
	Safe	26	56	22	39.29
	ZXing	26	399	118	29.57

Table 3 Nember of same metrics between projects of different companies.

number / Company / Company	NASA	SOFTLAB	AEEEM	ReLink
NASA	37	28	1	4
SOFTLAB	28	29	1	4
AEEEM	1	1	61	1
ReLink	4	4	1	26

CPDP and WPDP methods?

- RQ2: Could SNN approach obtain better performance for HDP?
- RQ3: Does the multi-source approaches perform better than the single-source approaches for HDP?

4.1 Datasets

In this study, we employ 18 publicly available and commonly used projects from four different companies including NASA [17], [18], AEEEM [19], SOFTLAB [17] and ReLink [20] as the experiment data. Table 2 lists the brief properties of 18 projects. Table 3 lists the number of same metrics of each pair of companies. Since the data quality affects the performance of classifier, we use the technique for cleaning the data proposed by Shepperd et al. [18] to clean the 18 public projects.

Each dataset in NASA represents a NASA software system or sub-system. We only use five projects including CM1, PC1, PC3, MW1 and PC4. The AEEEM dataset was collected by D'Ambros et al. [19], which consists of 61 metrics including EQ, JDT, LC, ML and PDE projects. The SOFTLAB dataset consists of AR1, AR3, AR4, AR5 and AR6 projects which exist 29 same metrics. ReLink dataset was collected by Wu et al. [20], which consists of Apache, Safe and ZXing projects and 26 metrics.

As can be seen above, NASA [17], [18], AEEEM [19], SOFTLAB [17] and ReLink [20] have different metric space. Thus, we use the UMR (introduced in Sect. 3.2) method to preprocess the software metrics before building our neural network. For example, when CM1 in NASA is as the training data and Apache in ReLink is as the testing data, there are 59 $(37 + 26 - 4)$ metrics used to build the first layer of the neural network.

Table 4 Defect prediction measure.

	Defective	No-defective
Predict as defective	TP	FP
Predict as no-defective	FN	TN

4.2 Evaluation Performance Metrics

In our experiments, we separately employ the prediction performance on the non-defective or defective class and the overall performance. The *Recall* are commonly used to measure the prediction performance on the defective class, the *AUC* [40] and Mattews correlation coefficient (MCC) [13] are utilized to measure the overall performance. These three measures can be calculated by true positive (*TP*), false positive (*FP*), true negative (*TN*) and false negative (*FN*) in Table 4. *TP* is the number of defective instances that are correctly predicted. *FP* is the number of non-defective instances that are predicted as defective. *TN* is the number of non-defective instances that are correctly predicted. *FN* is the number of defective instances that are predicted as non-defective.

Recall is defined as the ratio of the number of defective instances that are correctly predicted to the total number of defective instances, and it is calculated as *TP*/(*TP*+*FN*).

AUC is defined as the area under the receiver operating characteristic curve, whose *x*-axis is *pf* and *y*-axis is *pd*.

MCC encompasses all four components of the confusion matrix, that has been demonstrated to be a reliable measure of prediction performance. It is defined as

$$MCC = \frac{TP \times TN - FP \times FN}{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}. \quad (9)$$

Recall and *AUC* range from 0 to 1, and *MCC* ranges from −1 to 1. Obviously, an good defect prediction model should have high values of *Recall*, *AUC* and *MCC*.

4.3 Experimental Design

4.3.1 Evaluation Settings

We use 18 projects from NASA, AEEEM, SOFTLAB and ReLink as experiment object to perform HDP. In experiment, each project is as the testing data in turn, and other project from other companies is separately as the training data. For example, when Apache in ReLink is as the testing, there are 15 (18 − 3) heterogeneous defect prediction combinations. We mainly focus on the data with heterogeneous metric sets, so we did not conduct defect prediction

Table 5 The standard parameter setting of the SNN.

Parameters	values
Learning rage (α)	0.02
Epoch	100
Dropout fraction	0.5
Lambda (λ)	0.35
Batch size	32

across projects in the same company. In general, we have 240 possible prediction combinations from 18 projects. In order to avoid randomness, we repeat the above experiments 20 times and report the mean results for each testing project.

4.3.2 Parameter Settings

In all our experiments, we set fc1 layer in SNN with $d_s + d_t - d_c$ nodes, fc2, fc3 and fc4 layers with 356 hidden nodes, and fc5 layer with 2 output nodes.

In the hyperparameter adjustment, since the learning rate determines whether the neural network can converge to the global minimum, the regularization terms in loss function is not taken into account firstly. We firstly adjust the learning rate to get a more appropriate threshold value, and take half of the threshold value as the initial value in the process of adjusting learning rate. After that, the size of mini-batch size is determined through experiments. Afterward, the learning rate is carefully adjusted and using the verified learning rate to choose the Lambda (λ). Lastly, the learning rate can be re-optimized by going back. The epoch can be determined by the overall observation of the above experiments.

In this process, the batch size was chosen to 8, 16, and 32 (the power of two); the range of learning rate was chosen to between 0.01 and 0.8; the dropout was set 0.5, which has been proven to produce better performance in [47]; the MMD regularization parameter (λ) is chosen to between 0.2 and 0.5, and this made the objective be primarily weighted towards classification error, and also with enough regularization to avoid overfitting [48]. And based on the hyperparameter adjustment, We set the parameters for the back-propagation learning specified in Table 5.

4.3.3 Performance Comparison

To statistically investigate the detailed prediction results, we conduct a non-parametric *Wilcoxon signed-rank test* [41] at a confidence level of 95% on multiple models over 18 benchmark projects. If $P - Value < 0.005$, it shows that there is a significant difference between the two methods.

Furthermore, To measure the degree of differences in *Recall*, *AUC* and *MCC* results between our method and the compared methods, we apply *Cohen's d* to measure the effect size, and this is calculated as followed [42]:

$$Cohen's\ d = \frac{M_1 - M_2}{\sqrt{(\sigma_1^2 + \sigma_2^2)/2}}. \quad (10)$$

Where M_1 and M_2 are the means, and σ_1, σ_2 are the standard deviations of experimental and control groups.

There are three levels of effect size in Cohen as $0 \leqslant d < 0.2$ (Negligible, N), $0.2 \leqslant d < 0.5$ (Small, S), $0.5 \leqslant d < 0.8$ (Medium, M) and $d \geqslant 0.8$ (Large, L).

4.4 Experimental Results and Analysis

In this section, we compare with DNN (DNN utilizes cross-entropy loss as the classification error and has the same network architecture with SNN), CNN (CNN applies the approach proposed by [34] to balance data before training neural network), some HDP methods including CCA+ [13] and CTKCA [43], some CPDP methods including NN-Filter [9], VCB-SVM [11] and TCBoost [12], and WPDP approach to validate the performance of our approach for HDP. The results of *Recall*, *AUC* and *MCC* are listed in Tables 6, 7 and 8. The best value of each testing project is in bold font. We also analyzed the performance results based on evaluating the variability of compared approaches across multiple runs by using violin figure in Figs. 3, 4 and 5. Furthermore, we examine the statistical differences and effect size between our approach and the compared approaches, and the results are listed in Table 9.

4.4.1 RQ1: Does SNN Approach Perform Better than the CPDP and WPDP Methods?

In order to address RQ1, we compare our SNN approach with WPDP approaches and typical CPDP approaches including NN-filter [9], VCB-SVM [11] and TCBoost [12]. We make comparion with 10% of training data for WPDP. In CPDP methods, we select one project as testing data in turn, and separately use other projects in the same company as source data.

In terms of the overall results in 18 testing projects, SNN approach improves the mean *Recall* by 29%, 13.1% and 11.5%, the mean *AUC* by 11%, 4.5% and 4.1% and the mean *MMC* by 15.2%, 4.5% and 4% over NN-filter, VCB-SVM and TCBoost, respectively. Compared with WPDP method, SNN approach improves the mean *Recall* by 6.7%, however, the *AUC* and *MCC* are lower than WPDP.

We further statistically evaluate the effect of our method using *Wilcoxon signed-rank test* and *Cohen's d* explained in Sect. 4.3.3, and conclude the following conclusions:

- Compared with NN-Filter, VCB-SVM and TCBoost approaches, most $P - Value < 0.005$, we can conclude that SNN can statistically improve the performance in *Recall*, *AUC* and *MCC* in most projects, and achieve large performance improvement compared to NN-filter method, and medium performance improvement compared to VCB-SVM and TCBoost methods.
- Compared with WPDP approach, all $P - Value > 0.005$. That is to say our approach is in the same level with WPDP approach, that indicate our SNN approach can solve the HDP problem.

Table 6 Results in *Recall* for each project.

Dataset	project	NN-Filter	VCB-SVM	TCBoost	WPDP	CCA+	CTKCCA	DNN	CNN	SNN	
										Single	Multi
NASA	CM1	0.165	0.4	0.458	0.36	0.09	0.308	0.445	0.45	**0.52**	0.46
	MW1	0.243	0.278	0.4	0.3	0.183	0.348	0.58	**0.584**	0.527	0.433
	PC1	0.175	0.428	0.403	0.27	0.126	0.262	0.487	0.487	**0.533**	0.543
	PC3	0.14	0.263	0.265	0.36	0.132	0.375	0.551	0.557	**0.629**	0.473
	PC4	0.213	0.238	0.233	0.37	0.165	0.388	0.482	0.443	**0.594**	0.367
SOFTLAB	AR1	0.22	0.388	0.333	0.5	0.178	0.494	0477	0.367	**0.632**	0.557
	AR3	0.498	0.813	**0.875**	0.67	0.178	0.475	0.64	0.691	0.741	0.5
	AR4	0.288	0.45	0.613	**0.75**	0.112	0.475	0.672	0.658	0.708	0.367
	AR5	0.468	0.845	0.783	0.8	0.199	0.462	0.858	0.742	**0.915**	0.543
	AR6	0.183	0.253	0.25	0.6	0.144	0.295	0.472	0.483	**0.488**	0.31
AEEEM	EQ	0.223	0.308	0.313	0.73	0.149	0.255	0.619	0.599	0.682	**0.797**
	JDT	0.3	0.555	0.6	0.62	0.164	0.329	0.567	0.551	0.618	**0.743**
	LC	0.403	0.453	0.433	**0.65**	0.138	0.337	0.43	0.498	0.569	0.6
	ML	0.285	**0.445**	0.4	0.38	0.108	0.278	0.328	0.336	0.348	0.393
	PDE	0.283	0.433	0.385	0.37	0.148	0.284	0.358	0.481	0.515	**0.52**
ReLink	Apache	0.495	**0.785**	0.755	0.72	0.063	0.349	0.574	0.579	0.617	0.483
	Safe	0.5	0.565	0.635	**0.75**	0.06	0.301	0.631	0.682	0.658	0.62
	ZXing	0.405	0.455	**0.505**	0.3	0.0487	0.347	0.367	0.419	0.407	0.283
	median	0.305	0.464	0.48	0.528	0.133	0.353	0.529	0.533	**0.595**	0.499

Table 7 Results in *AUC* for each project.

Dataset	project	NN-Filter	VCB-SVM	TCBoost	WPDP	CCA+	CTKCCA	DNN	CNN	SNN	
										Single	Multi
NASA	CM1	0.543	0.614	0.621	0.608	0.48	0.496	0.612	0.618	**0.641**	0.624
	MW1	0.571	0.579	0.614	0.635	0.532	0.529	0.677	0.682	**0.692**	0.654
	PC1	0.539	0.648	0.633	0.629	0.499	0.45	0.646	0.616	**0.668**	0.658
	PC3	0.534	0.587	0.585	0.637	0.502	0.527	0.655	0.657	**0.678**	0.643
	PC4	0.547	0.583	0.582	0.676	0.511	0.529	0.639	0.62	**0.677**	0.594
SOFTLAB	AR1	0.58	0.66	0.634	**0.721**	0.518	0.543	0.587	0.583	0.686	0.654
	AR3	0.682	0.66	0.634	0.721	0.518	0.543	0.74	0.766	**0.805**	0.71
	AR4	0.607	0.667	0.715	**0.789**	0.514	0.53	0.723	0.698	0.756	0.649
	AR5	0.685	0.846	0.864	**0.9**	0.565	0.614	0.839	0.847	0.843	0.741
	AR6	0.573	0.598	0.609	**0.762**	0.503	0.462	0.624	0.663	0.659	0.594
AEEEM	EQ	0.578	0.609	0.61	0.662	0.536	0.497	0.707	0.671	0.729	**0.755**
	JDT	0.576	0.665	0.68	0.626	0.54	0.531	0.695	0.696	**0.726**	0.723
	LC	0.593	0.639	0.63	**0.762**	0.512	0.498	0.629	0.647	0.692	0.72
	ML	0.56	0.62	0.609	**0.875**	0.494	0.504	0.591	0.596	0.62	0.61
	PDE	0.559	0.602	0.601	0.582	0.528	0.493	0.59	0.629	**0.659**	0.658
ReLink	Apache	0.565	0.698	0.693	0.667	0.639	**0.753**	0.486	0.459	0.695	0.644
	Safe	0.669	0.704	0.701	**0.751**	0.496	0.476	0.704	0.693	0.722	0.649
	ZXing	0.611	0.619	0.616	**0.771**	0.463	0.509	0.578	0.587	0.591	0.57
	median	0.587	0.652	0.656	**0.719**	0.512	0.509	0.661	0.662	0.697	0.658

Table 8 Results in *MCC* for each project.

Dataset	project	NN-Filter	VCB-SVM	TCBoost	WPDP	CCA+	CTKCCA	DNN	CNN	SNN	
										Single	Multi
NASA	CM1	0.097	0.201	0.196	0.18	-0.037	-0.01	0.191	0.196	**0.217**	0.204
	MW1	0.154	0.159	0.195	**0.368**	0.087	0.054	0.241	0.248	0.312	0.263
	PC1	0.0708	0.224	0.212	**0.347**	-0.002	-0.083	0.203	0.206	0.231	0.209
	PC3	0.09	0.186	0.186	**0.258**	0.013	0.041	0.233	0.236	0.257	0.228
	PC4	0.104	0.192	0.193	**0.487**	0.021	0.046	0.233	0.236	0.26	0.203
SOFTLAB	AR1	0.217	0.31	0.31	**0.367**	0.034	0.044	0.17	0.175	0.248	0.251
	AR3	0.288	0.45	0.47	**0.604**	0.084	0.025	0.417	0.445	0.526	0.528
	AR4	0.325	0.367	0.422	0.439	0.052	0.055	0.378	0.381	0.447	**0.452**
	AR5	0.351	0.629	**0.742**	0.828	0.24	0.28	0.609	0.566	0.614	0.696
	AR6	0.254	0.259	0.32	**0.523**	0.01	-0.059	0.224	0.209	0.317	0.238
AEEEM	EQ	0.229	0.285	0.292	0.355	0.132	0.008	0.431	0.465	0.46	**0.504**
	JDT	0.159	0.313	0.334	0.235	0.132	0.072	0.347	0.331	**0.426**	0.376
	LC	0.136	0.24	0.205	**0.526**	0.056	0.002	0.216	0.216	0.323	0.322
	ML	0.118	0.307	0.185	**0.783**	-0.013	0.006	0.179	0.14	0.239	0.186
	PDE	0.115	0.19	0.199	0.197	0.092	-0.006	0.263	0.17	0.209	**0.275**
ReLink	EQ	0.134	0.403	0.39	**0.505**	0.002	-0.008	0.347	0.356	0.401	0.298
	Safe	0.377	0.422	0.397	**0.494**	-0.026	-0.061	0.417	0.406	0.46	0.317
	ZXing	0.237	0.241	0.225	**0.417**	-0.013	0.027	0.164	0.176	0.187	0.217
	median	0.192	0.299	0.304	**0.439**	0.048	0.02	0.286	0.288	0.344	0.32

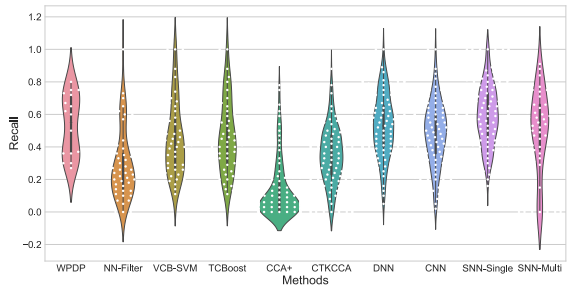

Fig. 3 The violin of *Recall* values of compared methods.

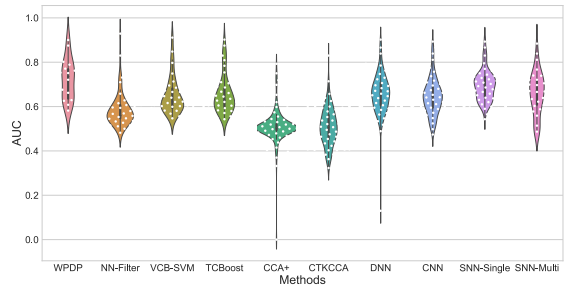

Fig. 4 The violin of *AUC* values of compared methods.

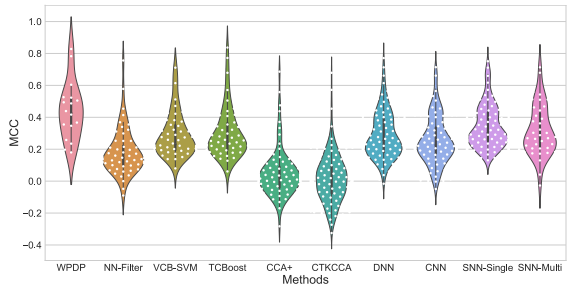

Fig. 5 The violin of *MCC* values of compared methods.

Possible reasons are that these compared CPDP approaches only use the same metrics in training and testing projects that limits the performance.

4.4.2 RQ2: Could SNN Approach Obtain Better Performance for HDP?

In order to address RQ2, we compare our SNN with DNN (DNN utilizes cross-entropy loss as the classification error and has the same network architecture with SNN), CNN (CNN applies the approach proposed by [34] to balance data before training neural network) and two exiting HDP approaches including CCA+ [13], and CTKCCA [43]. The settings of these approaches are the same as our approach.

From these Tables 6, 7 and 8, we observe that our approach achieve the highest *Recall*, *AUC* and *MCC* in most projects compared with DNN, CNN, CCA+ and CTKCCA. In terms of the overall results in 18 testing projects, SNN approach improves the mean *Recall* by 6.6%, 6.2%, 46.2% and 24.2%, the mean *AUC* by 3.6%, 3.5%, 18.5% and 18.8%, and the mean *MCC* by 5.8%, 5.6%, 29.6% and 32.4% over DNN, CNN, CCA+ and CTKCCA, respectively.

Table 9 The test results of compared approaches.

Methods	Measure	Wilcoxon singed-rank test	*Cohen's d*
SNN-multi	**Recall**	0.053	0.423(S)
	AUC	0.278	0.4468(S)
	MCC	0.381	0.213(S)
WPDP	**Recall**	0.197	0.356(S)
	AUC	0.407	-0.282
	MCC	0.031	-0.604
NN-Filter	**Recall**	**0**	1.536(L)
	AUC	**0**	1.537(L)
	MCC	0.381	0.213(S)
VCB-SVM	**Recall**	**0**	0.692(M)
	AUC	**0.001**	0.603(M)
	MCC	**0.089**	0.364(S)
TCBoost	**Recall**	**0**	0.631(M)
	AUC	**0.002**	0.53(M)
	MCC	0.01	0.296(S)
CCA+	**Recall**	**0**	2.615(L)
	AUC	**0**	2.765(L)
	MCC	**0**	2.141(L)
CTKCCA	**Recall**	**0**	1.358(L)
	AUC	**0**	2.476(L)
	MCC	**0**	2.321(L)
DNN	**Recall**	**0.002**	0.341(S)
	AUC	**0.001**	0.452(S)
	MCC	**0.001**	0.414(S)
CNN	**Recall**	**0.001**	0.453(S)
	AUC	**0.001**	0.423(S)
	MCC	**0.001**	0.398(S)

We also further statistically evaluate the effect of our method, and conclude the following conclusions:

- Compared with DNN and CNN approaches, SNN achieves the highest *Recall*, *AUC* and *MCC* in most projects and $P - Value < 0.005$. That is to say, the weighted cross-entropy loss in SNN could well solve the class imbalance problem in HDP.
- Compared with CCA+ and CTKCCA approaches, SNN achieves the highest *Recall*, *AUC* and *MCC* in most projects and $P - Value < 0.005$, we can conclude that SNN can statistically improve the performance in *Recall*, *AUC* and *MCC* in most projects. At the same time, CTKCCA considered the cost-sensitive to solve the class imbalance problem and the kernel canonical correlation analysis (TCCA) to solve the distribution differences. That is to say, the deep neural network that has the complex nonlinear mapping capabilities is helpful for HDP.
- For *Cohen's d*, SNN achieves a small performance improvement compared to DNN and CNN, and large improvement compared to CTKCCA. That is to say, the contribution of deep domain neural network would be higher than weighted cross-entropy loss.

Possible reasons are that: (1) CCA approach focused on learning linear correlation or similarity between the training and testing data, and did not consider the class imbalance problem. (2) Though CTCCA considered the class imbalance problem, they used kernel canonical correlation analysis to map the source and testing data into the same

metric space, whose learning ability of nonlinear mapping is not better to deep neural network.

4.4.3 RQ3: Does the Multi-Source Approaches Perform Better than the Single-Source Approaches for HDP?

In order to address RQ3, we compare with the multi-source projects approach. Each project was used as the testing data in turn, the other projects in other company were separately used as the training data. For example, when Apache in ReLink is as the testing data, all AEEEM projects (or all SOFTLAB projects or all ReLink projects) are as the multi-source training data. In this case, we have 54 possible prediction combinations from 18 projects.

From Tables 6, 7 and 8, SNN approach with single-source improves the mean *Recall* by 9.6%, the mean *AUC* by 3.9%, and the mean *MMC* by 2.4% over multi-source approach. We observe that the multi-source SNN perform inferior to the single-source SNN approach in *Recall*, *AUC* and *MCC* for most projects.

We also further statistically evaluate the effect of our method, and conclude the following conclusions:

- Compared with multi-source approach, all $P-Value >$ 0.005. That is to say our approach with single-source is in the same level with multi-source approach.
- SNN can achieve small performance improvement compared to SNN-Multi.

Possible reason may be that there are different distributional character among different project data, the mixed project data as the training data may be more complex to learn the model.

To sum up, we can conclude that our SNN with single-source can perform better than typical CPDP approaches including NN-filter [9], VCB-SVM [11] and TCBoost [12], and HDP methods including CCA+ and CTKCCA. Our SNN is also comparable to WPDP method. The single-source SNN is better than multi-source for most projects.

5. Discussion

5.1 Effect of the MMD Distance

In this experiment, we investigate the effect of the Maximum Mean Discrepancy (MMD) and correlation alignment (CORAL) as the $Distance(Q_s, Q_t)$ with no $Distance(Q_s, Q_t)$. The results of *Recall*, *AUC* and *MCC* are listed in Table 10. To graphically visualize the prediction results across multiple runs, we use the violin Figs. 6, 7 and 8 to analyze the performance results.

From Table 10 and Figs. 6, 7 and 8, we can observe that the $Distance(Q_s, Q_t)$ being introduced into the loss function are useful for the heterogeneous defect prediction, and the Maximum Mean Discrepancy (MMD) is better used to reduce the distribution mismatch for software data.

Table 10 The results of different distance.

Projects	Measure	MMD	CORAL	No-Distance
CM1	**Recall**	**0.52**	0.228	0.234
	AUC	**0.641**	0.566	0.567
	MCC	**0.217**	0.175	0.181
MW1	**Recall**	**0.527**	0.306	0.168
	AUC	**0.692**	0.615	0.552
	MCC	**0.312**	0.309	0.161
PC1	**Recall**	**0.533**	0.258	0.183
	AUC	**0.668**	0.582	0.564
	MCC	**0.231**	0.21	0.178
PC3	**Recall**	**0.629**	0.33	0.198
	AUC	**0.678**	0.594	0.552
	MCC	**0.257**	0.206	0.133
PC4	**Recall**	**0.594**	0.26	0.235
	AUC	**0.677**	0.574	0.567
	MCC	**0.26**	0.177	0.149
AR1	**Recall**	**0.632**	0.205	0.11
	AUC	**0.686**	0.549	0.527
	MCC	**0.248**	0.112	0.111
AR3	**Recall**	**0.741**	0.403	0.318
	AUC	**0.805**	0.677	0.631
	MCC	0.526	**0.58**	0.47
AR4	**Recall**	**0.708**	0.381	0.262
	AUC	**0.756**	0.647	0.594
	MCC	0.447	**0.46**	0.341
AR5	**Recall**	**0.915**	0.425	0.345
	AUC	**0.843**	0.687	0.634
	MCC	0.614	**0.615**	0.462
AR6	**Recall**	**0.488**	0.162	0.168
	AUC	**0.65**9	0.567	0.573
	MCC	0.317	0.323	**0.339**
EQ	**Recall**	**0.682**	0.071	0.126
	AUC	**0.729**	0.53	0.54
	MCC	**0.46**	0.184	0.218
JDT	**Recall**	**0.618**	0.115	0.046
	AUC	**0.726**	0.546	0.536
	MCC	**0.426**	0.222	0.239
LC	**Recall**	**0.569**	0.136	0.086
	AUC	**0.692**	0.552	0.537
	MCC	**0.323**	0.195	0.228
ML	**Recall**	**0.348**	0.133	0.077
	AUC	**0.62**	0.547	0.524
	MCC	**0.239**	0.173	0.108
PDE	**Recall**	**0.515**	0.081	0.121
	AUC	**0.659**	0.532	0.536
	MCC	**0.263**	0.149	0.131
Apache	**Recall**	**0.401**	0.296	0.178
	AUC	**0.695**	0.588	0.543
	MCC	**0.401**	0.296	0.178
Safe	**Recall**	**0.658**	0.213	0.123
	AUC	**0.722**	0.594	0.549
	MCC	**0.46**	0.377	0.326
ZXing	**Recall**	**0.407**	0.164	0.079
	AUC	**0.591**	0.537	0.517
	MCC	**0.187**	0.141	0.088

5.2 Effect of the Number of Same Metrics between Training and Testing Data

In this experiment, we investigate the influence of the number of same metrics between training and testing data. All experiments are conducted in single source scenario. Specifically, the used number of same metrics ranges in 28, 4 and 1. The results of *Recall*, *AUC* and *MCC* are listed in

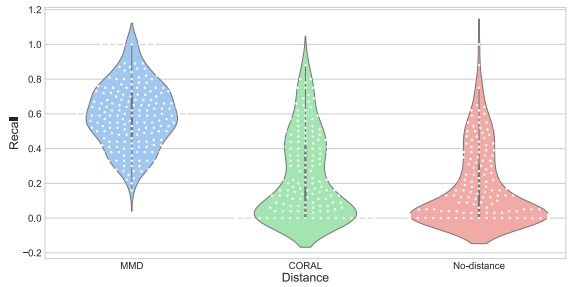

Fig. 6 The violin of *Recall* values of different distance.

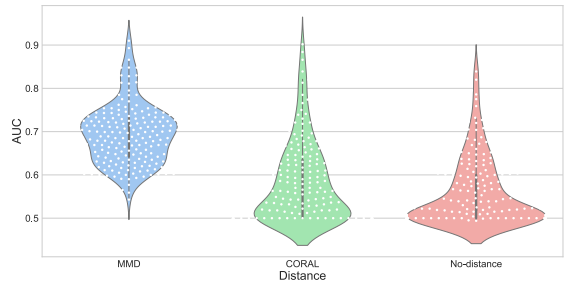

Fig. 7 The violin of *AUC* values of different distance.

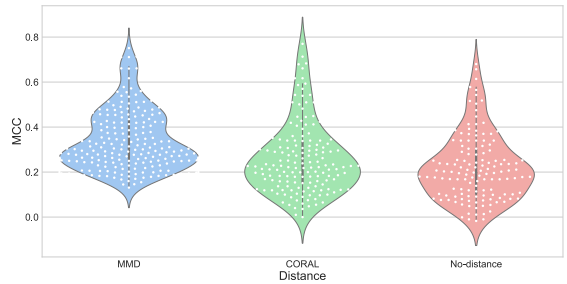

Fig. 8 The violin of *MCC* values of different distance.

Table 11. We also apply the *Cohen's d* to measure the effect. We count the number of projects in each effectiveness level to give a clearer comparison of the prediction and report the test results in Table 12. From Table 12, We observe that the number of same metrics has little effect on SNN approach. That is to say that our approach can use data from other companies with little same metrics.

5.3 Why SNN Performs Best

The discovery of this study provide substantial empirical support demonstrating that our SNN approach can solve the heterogeneous defect prediction problem. For most projects, SNN can achieve better *Recall*, *AUC* and *MCC*. The performance of SNN can be attributed to considering the distribution differences (MMD) and classification error (weighted cross-entropy loss) simultaneously.

HDP is to detect the largest number of defective modules in one project by using other projects with different metrics. Though UMR is used to map the different metrics into a unified metric representation, they have different distribution differences. As we know, deep neural network has a strong learning capacity, so we utilize the MMD as the distance measure regularization embedded in the supervised back-propagation training. Where, MMD is used to determine whether the distribution between source and testing data is the same. To demonstrate whether MMD regularization can indeed improve the performance of neural networks, we train the neural network with the MMD and

Table 11 The results of different number of same metrics

Projects	Measure	1 same metric	4 same metric	28 same metric
CM1	**Recall**	0.492	**0.653**	0.468
	AUC	0.631	**0.669**	0.635
	MCC	0.199	**0.238**	0.223
MW1	**Recall**	0.538	**0.557**	0.498
	AUC	0.694	**0.707**	0.681
	MCC	0.309	**0.329**	0.306
PC1	**Recall**	0.466	**0.617**	0.55
	AUC	0.6448	**0.695**	0.675
	MCC	0.211	**0.252**	0.239
PC3	**Recall**	0.634	**0.7**	0.582
	AUC	0.667	**0.71**	0.671
	MCC	0.236	**0.299**	0.253
PC4	**Recall**	0.616	**0.687**	0.516
	AUC	0.676	**0.71**	0.658
	MCC	0.253	**0.294**	0.247
AR1	**Recall**	0.576	**0.89**	0.532
	AUC	0.679	**0.726**	0.669
	MCC	**0.261**	0.238	0.242
AR3	**Recall**	0.7	0.75	**0.776**
	AUC	0.79	0.778	**0.837**
	MCC	0.503	0.451	**0.595**
AR4	**Recall**	0.67	**0.75**	0.72
	AUC	0.728	0.758	**0.783**
	MCC	0.399	0.434	**0.503**
AR5	**Recall**	0.926	**0.96**	0.876
	AUC	0.816	0.848	**0.866**
	MCC	0.561	0.592	**0.681**
AR6	**Recall**	0.494	**0.603**	0.412
	AUC	**0.696**	0.695	0.65
	MCC	0.399	**0.402**	0.329
Apache	**Recall**	0.612	**0.62**	-
	AUC	**0.696**	0.695	-
	MCC	0.4	**0.402**	-
Safe	**Recall**	0.636	**0.669**	-
	AUC	0.721	**0.722**	-
	MCC	**0.46**	0.459	-
ZXing	**Recall**	0.406	**0.407**	-
	AUC	0.59	**0.592**	-
	MCC	0.185	**0.188**	-
EQ	**Recall**	0.682	-	-
	AUC	0.729	-	-
	MCC	0.46	-	-
JDT	**Recall**	0.618	-	-
	AUC	0.726	-	-
	MCC	0.426	-	-
LC	**Recall**	**0.569**	-	-
	AUC	**0.692**	-	-
	MCC	**0.323**	-	-
ML	**Recall**	**0.348**	-	-
	AUC	**0.62**	-	-
	MCC	**0.239**	-	-
PDE	**Recall**	**0.515**	-	-
	AUC	**0.659**	-	-
	MCC	**0.263**	-	-

Table 12 The effectiveness levels of SNN with different number of same metrics with the *Cohen's d* effect test.

Measure	Level	Against	
		1 same metric	4 same metric
Recall	**N**	0	1
	S	2	5
	M	3	2
	L	0	1
AUC	**N**	0	1
	S	2	1
	M	1	1
	L	2	4
MCC	**N**	1	0
	S	0	2
	M	0	2
	L	3	3

without MMD, respectively (introduced in Sect. 5.1). Figures 6, 7 and 8 indicate that the utilization of MMD might gain more adaptation ability than without MMD.

As the same time, we consider the class imbalance problem which hinders the classification performance. Firstly, we use the cross-entropy loss as the classfication error, which can measure the similarity between the real label distribution and the predicted label distribution. Also, the defective modules misclassified are more important in software engineering, we introduce an appropriate weight into cross-entropy loss function. To demonstrate the weighted cross-entropy loss function, we train the neural network with cross-entropy loss (DNN), weighted cross-entropy loss (SNN) and preprocessing SMOTE approach (CNN), Figs. 3, 4 and 5 indicate that the utilization of weighted cross-entropy loss might gain more adaptation ability. Especially for NASA datasets, we achieve the highest *Recall*, *AUC* and *MCC* for all projects. The proportion of defective data are all below 15%, this also indicates the weighted cross-entropy loss can solve the class-imbalance problem better.

6. Threats to Validity

In this section, we describe the potential threats to validity of our study.

6.1 Threats to Construct Validity

Our proposed method utilizes the weighted cross-entropy loss function as the classification error. Using other loss function might obtain different results. However, the cross-entropy loss function is widely used in deep learning field [44], [45].

With regard to the weight, we adaptively allocate different misclassification costs according to the number of non-defective instances and defective instances of source data. However, how to set the weight is a problem that is not yet effectively solved [46].

6.2 Threats to Internal Validity

We use *Recall*, *AUC* and *MCC* to report the prediction performance. Other comprehensive measures, such as *F-measure*, *G-means* are not reported.

In addition, we carefully implement the compared approaches, as reported by their authors. However, our experiments may not be totally the same as the original papers, which leads to some possible biases in comparison between our method and compared methods.

6.3 Threats to External Validity

We choose projects from four public available companies including NASA, AEEEM, SOFTLAB and ReLink that have been used in many researches. Our chosen projects have the diversity in size and ratio of defective instances, and this helps draw the generalization of our findings. Also, we consider a larger number of datasets, and this further validates our studies. However, further investigations on other business datasets are needed.

6.4 Threats to Statistical Conclusion Validity

In our study, we use the Wilcoxon sign rank test for the statistical analysis. To find the practical effects of the results, *Cohen's d* is used to measure the effect size.

7. Conclusions

In this paper, we present unsupervised deep domain adaptation approach for heterogeneous defect prediction. To address the heterogeneous problem, we utilize the Maximum Mean Discrepancy (MMD) to calculate the distance between the source and target data. To address the class imbalance problem, we use the weighted cross-entropy loss function as the classification loss.

We perform the non-parametric *Wilcoxon signed-rank test* and *Cohen's d* effect size test for the evaluation. Extensive experiments on 18 public projects from four companies indicate that the proposed approach can build an effective prediction model for heterogeneous defect prediction (HDP).

For the future work, we would like to employ more data from commercial projects to evaluate the effectiveness of our approach.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China (No. 61673384 and No. 61502497), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No.18KJB520016), and Research Support Program for Doctorate Teachers of Jiangsu Normal University (No. 17XLR001).

References

[1] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," IEEE Computer Society, pp.45–54, 2013.

[2] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An Investigation On the Feasibility of Cross-project Defect prediction," Automated Software Engineering, vol.19, no.2, pp.167–199, 2012.
[3] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with Noise in Defect Prediction," 2011 International Conference on Software Engineering (ICSE), pp.481–490, 2011.
[4] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A Large-scale Empirical Study of Just-in-time Quality Assurance," IEEE Trans. Softw. Eng., vol.39, no.6, pp.757–773, 2013.
[5] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," IEEE International Conference on Automated Software Engineering, pp.279–289, 2014.
[6] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary Learning based Software Defect Prediction," Proc. 36th International Conference on Software Engineering, pp.414–423, ACM, 2014.
[7] S. Wang, T. Liu, and L. Tan, "Automatically Learning Semantic Features for Defect Prediction," IEEE International Conference on Software Engineering, pp.297–308, 2017.
[8] T. Lee, J. Nam, D. Han, S. Kim, and H.P. In, "Developer Micro Interaction Metrics for Software Defect Prediction," IEEE Trans. Softw. Eng., vol.42, no.11, pp.1015–1035, 2016.
[9] B. Turhan, T. Menzies, A.B. Bener, and J.D. Stefano, "On the Relative Value of Cross-company and Within-company Data For Defect Prediction," Empirical Software Engineering, vol.14, no.5, pp.540–578, 2009.
[10] J. Nam, S.J. Pan, and S. Kim, "Transfer Defect Learning," International Conference on Software Engineering, pp.382–391, IEEE, 2013.
[11] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," Empir. Softw. Eng., vol.21, no.1, pp.43–71, 2016.
[12] D. Ryu, J.-I. Jang, and J. Baik, "A Transfer Cost-sensitive Boosting Approach for Cross-project defect prediction," Software Quality Journal, vol.25, no.1, pp.235–272, 2017.
[13] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," Joint Meeting, pp.496–507, 2015.
[14] J. Nam, F. Wei, S. Kim, T. Menzies, and T. Lin, "Heterogeneous defect prediction," Proc. 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'2015), pp.508–519, 2015.
[15] P. He, B. Li, and Y. Ma, "Towards cross-project defect prediction with imbalanced feature sets," Computer Science, 2014.
[16] M. Cheng, G. Wu, M. Jiang, H. Wan, G. You, and M. Yuan, "Heterogeneous Defect Prediction via Exploiting Correlation Subspace," The International Conference on Software Engineering and Knowledge Engineering, pp.171–176, 2016.
[17] G. Blanchard and R. Loubère, "High Order Accurate Conservative Remapping Scheme on Polygonal Meshes Using a Posteriori MOOD Limiting," Computers and Fluids, vol.136, pp.83–103, 2016.
[18] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," IEEE Trans. Softw. Eng., vol.39, no.9, pp.1208–1215, 2013.
[19] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating Defect Prediction Approaches: a Benchmark and an Extensive Comparison," Empirical Software Engineering, vol.17, no.4-5, pp.531–577, 2012.
[20] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: Recovering Links Between Bugs and Changes," ACM Sigsoft Symposium and the European Conference on Foundations of Software Engineering, pp.15–25, 2011.
[21] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a Fault Prediction Model to Allow Inter Languagereuse," International Workshop on Predictor MODELS in Software Engineering, pp.19–24, ACM, 2008.
[22] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of Cross-project Defect Prediction," the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp.1–11, 2012.
[23] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing Privacy and Utility in Cross-Company Defect Prediction," IEEE Trans. Softw. Eng., vol.39, no.8, pp.1054–1068, 2013.
[24] F. Peters, T. Menzies, and L. Layman, "LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction," IEEE International Conference on Software Engineering, pp.801–811, 2015.
[25] Y. Kamei, T. Fukushima, S. Mcintosh, K. Yamashita, N. Ubayashi, and A.E. Hassan, "Studying Just-in-time Defect Prediction Using Cross-project Models," Empirical Software Engineering, vol.21, no.5, pp.2072–2106, 2016.
[26] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards Building a Universal Defect Prediction Model," Proc. 11th Working Conference on Mining Software Repositories (MSR 2014), pp.182–191, 2014.
[27] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," Empirical Software Engineering, vol.22, no.4, pp.1–37, 2016.
[28] G.I. Taylor and A.E. Green, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," Information and Software Technology, vol.95, no.2, pp.1–17, 2017.
[29] J.P. Sinno, W.T. Ivor, T.K. James, and Q. Yang, "Domain adaptation via transfer component analysis," IEEE Trans. Neural Netw., vol.22, no.2, pp.199–210, 2013.
[30] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer Learning for Cross-company Software Defect Prediction," Information and Software Technology, vol.54, no.3, pp.248–256, 2012.
[31] X. Yu, M. Wu, Y. Jian, K.E. Bennin, M. Fu, and C.X. Ma, "Cross-company Defect Prediction via Semi-supervised Clustering-based Data Filtering and MSTrA-based Transfer Learning," Software Computing, vol.22, no.10, pp.3461–3472, 2018.
[32] S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," IEEE Trans. Rel., vol.62, no.2, pp.434–443, 2013.
[33] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary Learning based Software Defect Prediction," 2014 Proc. 36th International Conference on Software Engineering (ICSE), pp.414–423, 2014.
[34] N. Limsettho, K.E. Bennin, J.W. Keung, H. Hata, and K. Matsumoto, "Cross Project Defect Prediction Using Class Distribution Estimation and Oversampling," Information and Software Technology, vol.100, pp.87–102, 2018.
[35] M. Liu, L. Miao, and D. Zhang, "Two-Stage Cost-Sensitive Learning for Software Defect Prediction," IEEE Trans. Rel., vol.63, no.2, pp.676–686, 2014.
[36] K.E. Bennin, K. Jacky, P. Phannachitta, A. Monden, and S. Mensah, "MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," IEEE Trans. Softw. Eng., vol.44, no.6, pp.534–550, 2018.
[37] C. Seiffert, T.M. Khoshgoftaar, J.V. Hulse, and A. Folleco, "An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy software quality data," Information Sciences and International Journal, vol.259, pp.571–595, 2014.
[38] K.E. Bennin, J.W. Keung, and A. Monden, "On the Relative Value of data Resampling Approaches for Software Defect Prediction," Empirical Software Engineering, no.1, pp.1–35, 2018.
[39] A.J. Smola, A. Gretton, and K.M. Borgwardt, "Maximum mean discrepancy," Technical report, NICTA-SML-06-001, National ICT Australia, 2006.
[40] B. Sun and K. Saenko, "Deep CORAL: Correlation alignment for deep domain adaptation," European Conference on Computer Vision, pp.443–450, 2016.
[41] G. Shieh, S.-L. Jan, and R.H. Randles, "Power and sample size Determinations for the Wilcoxon signed-rank test," Journal of Statis-

tical Computation and Simulation, vol.77, no.8, pp.717–724, 2007.
[42] K. Muller, "Statistical power analysis for the behavioral sciences," Technometrics, vol.31, no.4, pp.499–500, 1988.
[43] Z. Li, X.Y. Jing, F. Wu, X. Zhu, and B. Xu, "Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction," Automated Software Engineering, no.1, pp.1–45, 2017.
[44] G.S. Kamaledin, "Competitive cross-entropy loss: A study on training single-layer neural networks for solving nonlinearly separable classification problems," Neural Processing Letters, pp.1–8, 2018.
[45] K. Hu, Z. Zhang, X. Niu, C. Cao, F. Xiao, and X.P. Gao, "Retinal vessel segmentation of color fundus images using multiscale convolutional neural network with an improved cross-entropy loss function," Neurocomputing, vol.309, 2018.
[46] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," Expert Systems with Application, vol.37, no.6, pp.4537–4543, 2010.
[47] P. Baldi and P. Sadowski, "The Dropout Learning Algorithm," Artificial intelligence, vol.210, pp.78–122, 2014.
[48] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," Computer Science, 2014.

Lina Gong is a PhD student at School of Computer Science and Technology, China University of Mining and Technology. Her research interests include software analysis and testing, machine learning.

Shujuan Jiang received the Ph.D. degree from Soutest University in 2006. She was a visiting scholar at Georgia Institute of Technology from September 2008 to April 2009. She is a professor and Ph.D. supervisor at School of Computer Science and Technology, China University of Mining and Technology. Her research interests include compilation techniques and software engineering, etc.

Qiao Yu received the Ph.D. degree from China University of Mining and Technology in 2017. She is a lecturer at School of Computer Science and Technology, Jiangsu Normal University. Her research interests include software analysis and testing, machine learning. She is a member of IEICE.

Li Jiang is a MS student at School of Computer Science and Technology, China University of Mining and Technology. Her research interests include software analysis and testing, machine learning.