Conditional Domain Adversarial Adaptation for Heterogeneous Defect Prediction

LINA GONG,1,2, SHUJUAN JIANG,1 (Member, IEEE), AND LI JIANG,1 (Member, IEEE)
1School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China
2School of Computing, Queen’s University, Kingston, ON K7L 3N6, Canada
Corresponding author: Shujuan Jiang (shjjiang@cumt.edu.cn)
This work was supported by the joint Ph.D. program of “double first rate” construction disciplines of CUMT.

ABSTRACT Heterogeneous defect prediction (HDP) has become a very active research field in software engineering, which predicts the maximum number of bug-suspiciousness modules of a target project by prediction models built on source project with heterogeneous metric set. At present, some researchers have proposed some HDP models with a promising performance. Most of existing HDP models adopted unsupervised transfer learning to map source project and target project into the same feature space, which only considered the metrics space, not the label information from source project and few part of target project. Meanwhile, the predictive ability of these HDP models in effort-aware context have not been compared. Therefore, we set up to investigate the effectiveness of label information on HDP, and to propose a HDP model for improving the predicting performance in classification and effort-aware contexts. In order to use these label information, we propose a novel conditional domain adversarial adaptation (CDAA) approach to tackle heterogeneous problem in SDP, which is motivated by generative adversarial networks (GANs). There are three networks in architecture of our CDAA, including one generator, one discriminator and one classifier. The generator learns how to transfer source instance space to target instance space. The discriminator learns how to identify the fake instances generated by generator. The classifier learns how to correctly classify the label of instances. In our CDAA, the loss function of classifier and discriminator are both back propagate to generator. Then, to ensure a fair comparison between state-of-the art methods and CDAA, we take AUC, MCC and $P_{opt}$ as measures to evaluate 28 open-source projects. Experimental results demonstrate that CDAA method could take advantage of label information to effectively map source project to target project and improve the predictive performance. Also, experimental results demonstrate that our CDAA method is not affected by the number of same metrics between source project and target project.

INDEX TERMS Software defect prediction (SDP), heterogeneous defect prediction (HDP), domain adaptation, generative adversarial networks.

I. INTRODUCTION

Software defect prediction (SDP) aim to detect as many defective modules as possible in a software project by learning models trained on sufficient historical labeled instances [1]–[5], which has caused widespread attention from industrial communities and academic [6], [7]. Prior studies have proved that defect prediction models with sufficient historical labeled instances could generate ideal performance. Unfortunately, it is difficult to obtain sufficient number of labeled training dataset in practice software development, especially for a project in the initial stage [8]–[10]. Cross-project defect prediction (CPDP) is one way to tackle lack of label instances of a project, which builds learning model on labeled instances of other projects [2], [10], [11]. Zimmermann et al. [8] firstly pointed out that due to the difference of data distribution, CPDP was still a tricky question through investigating 622 cross-project combined models. After that, many CPDP approaches have been proposed to solve the difference, such as VCB-SVM [12], TCBoost [13], DTB [14], MNB [15], and HYDRA [9]. However, these methods assumed that train and test projects had the same metric sets. In practice, since projects have different developing environment and developing language, different metric sets would be extracted from these projects (i.e., NASA repository has 37 metrics, while MORPH repository...
only has 20 metrics). Thus, these existing CPDP models are difficult to apply in this scenario, especially for the new open software company.

For this scenario, the biggest problem is the different metric sets between source and target projects, which is called heterogeneous problem. To tackle heterogeneous problem, many heterogeneous defect prediction (HDP) models have been proposed [16], [17], [18]. They mainly employed different technologies to transfer source and target metric spaces into the same feature space. Such as, Jing et al. [17] proposed a transfer canonical correlation analysis (CCA+) approach to handle heterogeneous problem. Nam et al. [18] proposed HDP-KS approach, including both feature selection and feature matching techniques to handle heterogeneous problem. Meanwhile, Li et al. [28] presented a cost-sensitive transfer kernel canonical correlation analysis (CTKCCA) method through introducing cost sensitive and kernel into CCA+ method. However, heterogeneous problem is still a challenge in software defect prediction.

Therefore, in our study, we also tackle the heterogeneous problem in SDP. In order to understand our work easily, we list a description of the specific terms in Table 1.

| Terms       | Description                                      |
|-------------|--------------------------------------------------|
| SDP         | Software defect prediction                       |
| WPD         | Within-project defect prediction                 |
| CPDP        | Cross-project defect prediction                  |
| MPDP        | Mixed-project defect prediction                  |
| HDP         | Heterogeneous defect prediction                  |
| Source data | instances with defect or no-defect label from a source project |
| Target training data | instances with defect or no-defect label from testing project |
| Testing data | Software modules needed to be classified from testing project               |
| CDAA        | conditional domain adversarial adaptation         |

**A. MOTIVATION**

As mentioned above, due to different developed language and environment, different projects may have different metric space (including different number of metrics). Most existing HDP models mostly employ the knowledge of metrics to transfer source and target project into the same feature space.

Firstly, when these HDP methods learned the transferable representations between source and testing instances, they didn’t consider the label information of source instances. The label information includes the relation between metrics and defect (or non-defect) label. After transferring source instances into common space, the relation between transferred instance space and original label information may be changed. Thus, the performance of the machine learning classifiers trained on these transferable instances may be hindered.

Secondly, in some case, test projects have a small number of label instances (named training target data), which include the relation between metrics and defect (or non-defect) label. Meanwhile, Turhan et al. [19] pointed out that predicting models trained on datasets including small number of training target data outperformed the predicting models only trained on source datasets. However, the effectiveness of train target data has not been investigated.

Finally, providing a sequential list of test modules by predictive models would be more effective for testers. Most models are measured on the defect-classification models, the effectiveness of models on the Effort-aware models have not been investigated.

Therefore, our study would set out to investigate and employ the label information including source data and target training data to achieve better transferable representations between source and testing instances on the defect-classification and Effort-aware models.

**B. CONTRIBUTION**

Motivated by this, the main contributions of our study could be summarized as follows:

- To take full advantage of the label information from source and target training instances, we introduce GAN into SDP, and propose a conditional domain adversarial adaptation (CDAA) approach. CDAA has three networks, which could learn the label information and generate a classifier when transferring source instance space into target instances space.

- We do numerous experiments on 28 public projects from five different repositories covering NASA [19], [20], AEEEM [7], SOFTLAB [19], ReLink [21] and MORPH [40] to estimate our CDAA method in the defect-classification and effort-aware context. The experimental results demonstrate that CDAA method can improve predicting performance for HDP with the label information.

The rest of our study is organized as follows. We present the related work about SDP learning models in section II; We describe our proposed CDAA method in section III; We describe our experimental setup in section IV; Experimental results and analysis are presented in section V; In section VI, we describe the threats to the validity of our approach; Finally, Section VII gives the conclusions and future directions.

**II. RELATED WORK**

We briefly review the CPDP models using common metric sets, HDP models, and domain adversarial networks in this section.

**A. CPDP USING COMMON METRIC SETS**

In real software development, CPDP learning models become imperative when a project doesn’t have abundant historical labeled instances at the beginning step. Many researchers have focused on designing effective learning models for CPDP [13], [22], [23].

Zunnernabb et al. [24] conducted broad-scale research on the CPDP learning model to draw conclusions about which metrics (domain or process) were more beneficial for cross-project. Through 622 combination experiments, they found...
that testers should first quantify and evaluate the code, process and domain metric before training CPDP learning model.

After that, some researchers proposed to solve different distribution problems by searching for similar instances. Such as, Turhan et al. [19] firstly proposed to select similar instances from other projects as training datasets, and build the learning model with K-nearest Neighbor (KNN). Then, Liu et al. [25] proposed another search-based learning model using multiple projects. They applied genetic algorithm to search, which included baseline, validation and validation-and-Voting classifiers. Canfora et al. [26] also proposed a multi-objective logistic regression learning model based on genetic algorithm. Xia et al. [9] proposed a Hybrid model reconstruction Approach (HYDRA) to build CPDP learning model. They generated a large number of classifiers to predict defective modules by genetic learning and integrated learning phases. Recently, Hosseini et al. [27] proposed a novel search-based method to select instances. Firstly, they applied NN-filter to extract confirmation dataset from train instances, and then used genetic algorithm to select instances as training set. Finally, they applied Naive Bayes as classifier to predict the test project.

Moreover, some researchers have proposed mixed-project defect prediction models to improve the performance by using small amount of label instances from test project. Such as, Turhan et al. [15] firstly investigated the effectiveness of mixed-project, and compared it with NB classifier-based WPDP learning model. Experimental results demonstrated using mixed-project performed as well as WPDP learning model in the early stages of a project. Chen et al. [14] proposed a double transfer boosting (DTB) method that integrated two layers of data transmission. They firstly applied data gravity to reshape the overall distribution of cross-company (CC) instances. Secondly, they eliminated negative effect instances by a NB classifier-based transfer boosting method. Ryu et al. [12] presented a value-cognitive boosting method based on support vector machine (VCB-SVM) to build CPDP learning model. This method simultaneously sampled and modified instances to improve performance. In addition, Ryu et al. [13] also presented a transfer cost-sensitive boosting (TCSBoost) method to build learning model that used a few labeled target data.

However, existing CPDP learning models assumed that train and test projects had the same metric space. Due to some reasons, the number of common metric sets is very small in practice, which hinders the application of these CPDP learning models.

B. HDP MODELS
In recent years, some researchers have focused on heterogeneous metrics problems and proposed some HDP models. Such as, Jing et al. [17] proposed canonical correlation analysis (CCA) to convert the UMR instances into the same metric space.

After that, Cheng et al. [16] proposed a cost-sensitive correlation transfer support vector machine (CCT-SVM) method to build a HDP learning model. Based on CCT-SVM method, Li et al. [28] presented a novel cost-sensitive transfer kernel canonical correlation analysis (CTKCCA) method to build a HDP learning model, which simultaneously considered linearly inseparable and class imbalance problems.

However, these existing HDP models mainly employed the feature space of source and test project to solve the heterogeneous problems. When transferring the feature space, the label information both from source project and small part of test project are not been considered. Thus, our paper is to leverage the label information to build a conditional domain adversarial adaptation model.

C. DOMAIN ADVERSARIAL NETWORKS
Machine learning techniques need the same distributional characteristics between training and testing dataset. If this condition is not met, the performance of classifiers would be reduced. In order to conquer this gap, transfer learning algorithm transfers feature knowledge from train to test dataset by weakening these assumptions. Deep transfer learning techniques study the transferable knowledge by deep neural networks, which many researchers have paid attention in recent years.

In these deep transfer learning techniques, adversarial-based deep transfer learning is to find transferable representations by introducing adversarial technology of generative adversarial nets (GANs) [29]. Because of its good prediction performance and strong practicality, researchers proposed different adversarial-based deep transfer learning nets. Such as, Chen et al. [30] introduced adversarial technology into transfer learning. Meanwhile, they introduced the domain adaptive regularization term into the loss function. Ganin and Lempitsky [31] expanded some standard layers and one new gradient reversal layer to build adversarial-based feed-forward networks. Tzeng et al. [32] combined GANs loss with discriminator, and proposed a new domain adaptation approach.

Distinguishing with the mentioned adversarial-based deep transfer learning approaches, we build our conditional domain adversarial adaptation to solve the heterogeneous problem in SDP.

III. OUR PROPOSED APPROACH
Firstly, to understand our approach a little bit better, we define the notation used in our work. After that, we present our proposed conditional domain adversarial adaptation (CDAA) approach including the architecture and training process.

A. NOTATION
When a new software company develops a new project, this company may not have enough labeled instances to build WPDP or CPDP learning model. One way is to employ defect
dataset from some open source projects. Different projects have different metric sets called heterogeneous problem. In this scenario, the dataset of new project is target dataset, while the dataset of other project is source dataset.

Supposed that $D_s$ is labeled instance space from source dataset, $D_{st}$ is the target training instance space, and $D_t$ is unlabeled instances space from target data.

$D_s$ contains a data set $X_s = \{x_s^1, x_s^2, \cdots, x_s^N\}$ and a label set $Y_s = \{y_s^1, y_s^2, \cdots, y_s^N\}$, where $x_s^i$ denotes the $i^{th}$ module in $X_s$, $y_s^i$ is the associated label, and $N$ is the number of modules of $X_s$. $D_{st}$ contains a data set $X_{st} = \{x_{st}^1, x_{st}^2, \cdots, x_{st}^M\}$ and a label set $Y_{st} = \{y_{st}^1, y_{st}^2, \cdots, y_{st}^M\}$, where $x_{st}^i$ denotes the $i^{th}$ module in $X_{st}$, $y_{st}^i$ is the associated label, $M$ is the number of labeled modules of target data. $D_t$ contains unlabeled dataset $X_t = \{x_t^1, x_t^2, \cdots, x_t^M\}$, where $x_t^i$ denotes the $i^{th}$ module in $X_t$ and $M$ is the number of modules of $X_t$. Note that the metric set in $X_s$, $X_{st}$ and $X_t$ are different and $d_s \neq d_t$, where $d_s$ is the number of metrics in $X_s$ and $d_t$ is the number of metrics in $X_{st}$ and $X_t$.

**B. CONDITIONAL DOMIAN ADVERSARIAL ADAPTATION**

In software engineering, some researchers have proved that few label instances from target project could ameliorate the predictive performance of CPDP learning models [8], [19]. Motivated by this, we introduce the generative adversarial networks (GNAs) into SDP and build conditional domain adversarial adaptation (CDAA) networks. CDAA can simultaneously learn the edge distribution from source project when transferring the feature space of source project to the feature space of target project.

1) THE ARCHITECTURE OF CDAA

In order to use label information in learning feature space transformation, and to get a classifier at the same time, our CDAA networks include three networks: generator (G), discriminator (D) and classifier (C). Generator is to learn a transferable representation from source instances to target instances, which can also be called feature extractor. Discriminator learns how to distinguish the source from the target project. Classifier learns how to minimize the misclassifying loss of source instances $X_s$ and target training instances $X_{st}$. Finally, the source instance space is mapped to the test instance space, in which the label information are included through back-propagation of classifier. The architecture is shown in Fig. 1.

In our CDAA networks, generator, discriminator and classifier are all three layers, which are input layer, hidden layer, and output layer. The number of neurons in first layer of generator is $d_s$, which is the number of metrics in source project. The number of neurons in first layer of discriminator and classifier are both $d_t$, which is the number of metrics in target project. Meanwhile, we add a Batch Normalization (BN) between full connection and activation function in each layer of these three networks, which could keep the input of each layer of networks in the same distribution during training.

**2) TRAINING OUR NETWORKS**

The aim of our CDAA is to obtain a strong classifier when studying a feature representation from source project to target project. Thus, during training our CDAA networks, we seek the parameters of generator to maximize the misclassifying generated instances loss of discriminator and to minimize the misclassifying label loss of classifier, the parameters of discriminator to minimize the misclassifying loss for generated and test instances, and the parameters of classifier to minimize the misclassifying label loss of generated and target training instances. To meet these criteria, the loss function of discriminator is defined as:

$$
Loss_D = E(-\log D(x_s + x_t)) + E(-\log (1 - D(G(x_s))))
$$

(1)

where, $G(x_s)$ is generated by generator. The discriminator is to minimize the loss of misclassified generated and target instances.

Meanwhile, in order to consider the label information of source project, the loss of classifier on source instances is added in generator. The loss function of generator is defined as:

$$
Loss_G = E(-\log D(G(x_s))) + E(-y_s \times \log(C(G(x_s)))) + E(-(1 - y_s) \times \log(1 - C(G(x_s))))
$$

(2)

Finally, the classifier is to minimize the loss of label information including both generated instances and target training instances. Thus the loss function of classifier is defined as:

$$
Loss_C = E(-y_s \times \log(C(G(x_s)))) + E(-(1 - y_s) \times \log(1 - C(G(x_s)))) + E(-y_{st} \times \log(C(x_{st}))) + E(-(1 - y_{st}) \times \log(1 - C(x_{st})))
$$

(3)

In order to understand our training process, the pseudo code for training CDAA is listed in algorithm 1.

**IV. EXPERIMENTAL SETUP**

**A. RESEARCH QUESTION**

For effectively evaluating our proposed CDAA networks, we conduct experiments with the following three questions:
Algorithm 1 Pseudocode for CDAA Approach

**Inputs:** Source labeled data from other project: \([X_s, Y_s]\)
unlabeled instances from target project \(X_t\);
target training data from target project \([X_{st}, Y_{st}]\);
The number of epochs: \(T\);
The number of epochs of discriminator: \(K\);

**Outputs:** The label of instances in \(y_t\)

1. Initialization: \(T, m, K\)
2. For \(t = 1, 2, \ldots, T\):
   3. Randomly sample \(m_t\) instances from \([X_s, Y_s]\)
   4. Randomly sample \(m_t\) instances from \([X_{st}, Y_{st}]\)
   5. For \(k = 1, 2, \ldots, K\):
      6. Feed discriminator with \(m_t\) and \(m_t\)
      7. Calculate loss function of discriminator
      8. Update the parameters of discriminator by Adam
      9. End For
   10. Feed generator with \(m_t\) instances and \(m_t\) instances
   11. Feed classifier with \(m_t\) instances and \([X_{st}, Y_{st}]\)
   12. Calculate loss function of classifier
   13. Update the parameters of classifier by Adam
   14. Calculate loss function of generator
   15. Update the parameters of generator by Adam
   16. End For
   17. Input \(x_t\) into classifier, and get the label of \(y_t\)

**RQ1:** Are label information effective for heterogeneous problem in SDP?

**RQ2:** Does the number of common metrics between source and target project affect the performance of CDAA method?

**RQ3:** Could our proposed CDAA method perform better than state-of-art SDP methods?

**B. DATASETS**

In our work, we apply 28 projects from five different companies covering NASA, AEEEM, SOFTLAB, ReLink and MORPH to conduct experiments. Table 2 lists the details about these projects. The first to six columns list the repository’s name, test project, number of metrics, number of instances, number of defective instances, and the percentage of defective instances, respectively.

Each project in NASA contains various static code metrics and corresponding label. In our study, we use CM1, MW1, PC1, PC3 and PC4 projects as experimental objects, which are cleaned by [20].

Each project of AEEEM includes 61 metrics consisting of 17 source code metrics, 17 entropy-of-code metrics, 5 entropy-of-change metrics, 17 churn-of source code metrics, and 5 previous-defect metrics. Marco et al. [7] gathered these datasets.

SOFTLAB repository includes AR1, AR3, AR4, AR5 and AR6 projects. Each project includes 29 static code metrics. Wu et al. [21] collected ReLink repository, which includes Apache, Safe and ZXing projects with 26 code metrics.

Jureczko and Madeyski [22] collected MORPH repository from the online PROMISE data repository. Each project in MORPH has 20 metrics including McCabe’s cyclomatic metrics, object-oriented metrics.

In order to indicate the number of common metrics among different repositories, Table 3 lists the number of common metrics. As shown above, NASA, AEEEM, SOFTLAB, ReLink and MORPH have few common metrics, which can be used for HDP.

**C. EVALUATION MEASURES**

For fairly evaluating the performance of learning models, we separately apply AUC, Matthews correlation coefficient (MCC) and \(P_{opt}\) as the performance measures. AUC and Matthews correlation coefficient (MCC) are used for evaluating the predicting performance in classification context, while \(P_{opt}\) is used for evaluating the performance in effort-aware context.

AUC is the threshold-independent measure, which is the area under the receiver operating characteristic curve (ROC). MCC is a measure considering all four confusion matrix categories. Also Chicco and Jurman [41] showed that MCC could provide a more informative and truthful score than F1-score and accuracy. MCC can be calculated by the formula:

\[
MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{4}
\]

\(P_{opt}\) is an effort-aware indicator, which is based on the area under the cost-effectiveness (CE) curve. A CE curve of a model is list in Figure 2, whose x-axis is the percentage of LOC reviewed and y-axis is the percentage of defects discovered. Meanwhile, OPT shows the optimal scenario which review the code based on the descending defect density. Defect density refers to the radio of number of defects to lines of code in a module. Worst shows the worst scenario which review the code based on the ascending defect density, and prediction model shows a actual model \(m\). So the \(P_{opt}\) of a

![FIGURE 2. Cost-Effectiveness curve.](image-url)
model \( m \) can be calculated by the following formula [42]:

\[
P_{opt} = \frac{Area_{\sigma}(\text{optimal}) - Area_{\sigma}(\text{worst})}{Area_{\sigma}(m) - Area_{\sigma}(\text{worst})} \tag{5}
\]

For these three measures, the bigger the measure is, the better the learning model is.

In addition, we compute the Wilcoxon-signed rank test [43], [44] and the Cohen’s d effect size test [27], [45] between CDAA method and other compared methods to statistically compare the experimental results.

The Wilcoxon-signed rank test is a paired difference test to test whether the results of two methods are statistically different. In our experiment, we set the confidence level of Wilcoxon-signed rank test is 99%. Cohen’s d is a effect size measure to quantify the difference between our method and other compared method, and it is defined as:

\[
Cohen’s d = \frac{M_1 - M_2}{\sqrt{\delta_1^2 + \delta_2^2}} \tag{6}
\]

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                            | 50.32                   |
|         | Sale    | 26               | 56                        | 22                            | 39.29                   |
|         | Zxing   | 26               | 399                       | 118                           | 29.57                   |
| MORPH   | ani1.3  | 20               | 125                       | 20                            | 16                      |
|         | camel1.0| 20               | 339                       | 13                            | 3.83                    |
|         | poi1.5  | 20               | 237                       | 141                           | 59.49                   |
|         | tomcat  | 20               | 858                       | 77                            | 8.97                    |
|         | velocity1.4 | 20       | 196                      | 147                          | 75                      |
|         | Xalan2.4| 20               | 723                       | 110                           | 15.21                   |
|         | xerces1.2| 20             | 440                       | 71                            | 16.14                   |
|         | arc     | 20               | 234                       | 27                            | 11.54                   |
|         | redtor  | 20               | 176                       | 27                            | 15.34                   |
|         | skarbonka| 20            | 45                        | 9                             | 20                      |

TABLE 3. Number of same metrics between projects of different companies.

| Number of same metrics between projects | NASA | SOFTLAB | AEEEM | ReLink | MORPH |
|---------------------------------------|------|---------|-------|--------|-------|
| Company                               |      |         |       |        |       |
| NASA                                  | 37   | 28      | 1     | 4      | 1     |
| SOFTLAB                               | 28   | 29      | 1     | 4      | 1     |
| AEEEM                                 | 1    | 1       | 61    | 1      | 8     |
| ReLink                                | 4    | 4       | 1     | 26     | 1     |
| MORPH                                 | 1    | 1       | 8     | 1      | 20    |

TABLE 4. Effectiveness Levels Based on Cohen’s d.

| Cohen’s d | Level       |
|-----------|-------------|
| 0 ≤ |d| ≤ 0.2 | Negligible (N) |
| 0.2 ≤ |d| < 0.5 | Small (S) |
| 0.5 ≤ |d| < 0.8 | Medium (M) |
| |d| ≥ 0.8 | Large (L) |

Depended on the values of Cohen’s d, it can be divided into four levels as shown in Table 4.

D. PARAMETER SETTINGS

In our CDAA networks, the number of neurons of hidden layer in generator, discriminator and classifier are all 256. The number of neurons of output layer in generator is the number of metrics in target project, while the number of neurons of output layer in discriminator and classifier are one. In addition, based on method suggested by reference [49], we tune the parameter in the CDAA networks. Table 5 displays the hyper-parameter values set in our CDAA networks.
TABLE 5. The hyper-parameter setting of the CDAA networks.

| Hyper-parameters       | Values |
|------------------------|--------|
| Learning rate ($\alpha$) | 0.002  |
| Epoch                  | 100    |
| K (iterations of discriminator) | 2      |
| Batch Size             | 32     |

V. EXPERIMENTAL RESULTS
A. RQ1: ARE LABEL INFORMATION EFFECTIVE FOR HETEROGENEOUS PROBLEM IN SDP?

Approach: To investigate the effect of target label information, we compare the different proportion of target training data which ranges from {0%, 10%, 15%, 20%}. Meanwhile, to investigate the effect of label information from source project, we compare with generative adversarial networks (GANs) method that doesn’t consider the label information of source data. GANs transfer the source instances into target instance space, and then use the transferred source instances to train RF classifier [46], [47]. We use RF classifier because several studies show that Random Forest learners often produce the best performing models across all of the studied SDP scenarios [36], [48].

Results: Figure 3, 4 and 5 respectively shows the AUC, MCC and Popt values with different proportion of target training data and label information from source instances across all 28 projects from five repositories. From these figures, we observe that CDAA method with some target training data (for instance, {10%, 15%, 20%}) outperform CDAA method without target training instances and GANs method. We also observe that CDAA method with 10%, 15%, and 20% of target training instances has similar performance.

Table 6 lists the statistical results and effect size between CDAA method with 10% target training data and CDAA methods with other label information. From this table, we observe that CDAA method with 10% target training data has statistically significant difference and large effect size with CDAA method without target training data and GAN method across all performance measures. We also observe that CDAA method with 10% target training data has no statistically significant difference and small effect size with CDAA method with 15% and 20% target training data.

TABLE 6. The statistical analysis results of different labeled information.

| Measure | 10% Vs. |
|---------|---------|
|         | 0% | GAN | 15% | 20% |
| AUC     | P-value | 0  | 0   | 0.06 | 0.03 |
|         | Cohen’s d | 1.357(L) | 1.001(L) | -0.279(S) | -0.371(S) |
| MCC     | P-value | 0  | 0   | 0.158 | 0.034 |
|         | Cohen’s d | 1.653(L) | 0.859(L) | -0.151(N) | -0.238(S) |
| Popt    | P-value | 0  | 0   | 0.043 | 0.029 |
|         | Cohen’s d | 0.914(L) | 0.847(L) | -0.242(N) | -0.342(S) |

Analysis: Compare with CDAA method without target training data, CDAA method with 10% target training data improve large performance. That is to say, target label information is useful to improve the performance of HDP. Compared with GANs method, CDAA method with 10% target training data improve large performance. That is to say, label information from source project is useful to transfer source to target, which improves the performance of HDP. Compare with CDAA method with 15% and 20% target training data, CDAA method with 10% target training data could have similar performance. That is to say, increasing the number of target training instances has little impact on performance of CDAA. CDAA can effectively capture the label information of target data from target training data.
B. RQ2: DOES THE NUMBER OF COMMON METRICS BETWEEN SOURCE AND TARGET PROJECT AFFECT THE PERFORMANCE OF CDAA METHOD?

Approach: To investigate whether the number of common metrics affect the performance of our CDAA method, experiments are conducted in different source dataset which has different common metrics with target dataset. As Table 3 shown, we can divide these common metrics into three situations: one, four/eight, and twenty-eight. One represents that there are few common metrics. Four/eight represents that there are small number of common metrics. Twenty-eight represents that there are lots of common metrics.

Results: Figure 6, 7 and 8 respectively shows the AUC, MCC and Popt values with different number of common metrics across all 28 projects from five repositories. Noted that for AEEEM, MORPH and ReLink, there are only one and four of common metrics. From these figures, we observe that except the Popt value of ReLink dataset, CDAA methods with these three types of common metrics have similar predicting performance across the five repositories.

![Figure 6. The AUC performance of CDAA method with different number of common metrics.](image)

![Figure 7. The MCC performance of CDAA method with different number of common metrics.](image)

Table 7 lists the statistical results and effect size between CDAA method with one common metric and CDAA method with other different number of common metrics across all the measures. From this table, we observe that CDAA method with one common metric has no statistically significant difference and small (or negligible) effect size with CDAA method with four/eight and twenty-eight common metrics.

![Figure 8. The Popt performance of CDAA method with different number of common metrics.](image)

C. RQ3: COULD OUR PROPOSED CDAA METHOD PERFORM BETTER THAN STATE-OF-ART SDP METHODS?

Approach: To investigate the predicting performance of our CDAA method, we compare our CDAA with the conventional within-project defect prediction methods including Random forests (RF) [34], ManualDown (MD) [35] and spectral clustering (SC) [37], the typical CPDP methods including VCB-SVM [12], TCBoost [13], DTB [14] and MNB [15], and HDP methods including CCA+ [17] and CTKCCA [28]. Noted that we just use our CDAA method with 10% target training instance. Our experimental environment is Python3.6 and Scikit-learn (0.19.2). Meanwhile, we run 20 times for each target project to avoid randomness.

Results: Figure 9, 10 and 11 respectively shows the AUC, MCC and Popt values of compared methods across all 28 projects from five repositories. We observe that (i) The median values of AUC, MCC and Popt measures obtained by CDAA method outperform compared CPDP methods of...
In addition, we compute the Wilcoxon-signed rank test [43], [44] and the cohen’s d effect size test [27], [45] between our CDAA method and the other studied methods. Meanwhile, we statistically quantify the number of datasets on compared methods with different statistical effect size. Table 8 lists the results of Wilcoxon-signed rank test. Figure 12, 13 and 14 respectively shows the number of datasets with different effect size between CDAA and other studied methods across studied measures. From these figures, we observe that except WPDP method with RF classifier, CDAA method has achieved significant improvements in most projects compared to studied methods. Taking the comparison results of CDAA and MNB as an example, 24 of 28 (13 or 13 of 28) projects are shown large or median positive effect size in terms of the AUC (MCC or Popt). Compared with WPDP method with RF classifier, though RF outperforms our CDAA method, there are not statistical difference in terms of AUC and MCC.

VCB-SVM [12], TCBoost [13], DTB [14] and MNB [15] across all five repositories. (ii) The median values of AUC, MCC and Popt measures obtained by our CDAA method outperform unsupervised learning methods of spectral clustering (SC) across all five repositories, and outperforms Manual-Down (MD) [35] method across 3/5 repositories. (iii) Except SOFTLAB repository, the median values of AUC, MCC and Popt measures obtained by our CDAA method outperform compared methods. (IV) The median values of AUC, MCC and Popt measures obtained by our CDAA method outperform compared HDP methods of CCA+ [17] and CT-KCCA [28].

Analysis: Based on above results, we can conclude CDAA method could get better performance than compared CPDP and HDP methods. Compared with VCB-SVM, DTB, TCBoost, and MNB methods that are CPDP learning models using mixed-project: they only employed the same metrics included in source and target projects to train the prediction models. However, the other metrics would include more
TABLE 8. The statistical analysis results between CDAA and other compared methods.

| Measure | DTB | MNB | TBoost | VCB-SVM | CDAA Vs. |
|---------|-----|-----|--------|---------|----------|
| AUC     | 0.00086 | 0.00146 | 0.00051 | 0.0012 | MD 0.0051 | SC 0.00434 | CCA+ 0.000126 | CKCCA 0.00106 | RF 0.11 |
| MCC     | 0.00099 | 0.00312 | 0.00041 | 0.000447 | 0.257 | 0.00122 | 0.00051 | 0.000626 | 0.021 |
| Popt    | 0.000285 | 0.000337 | 0.00028 | 0.00041 | 0.000251 | 0.000274 | 0 | 0 | 0.0093 |

FIGURE 14. The number of datasets with different statistical effect size between CDAA and studied methods in terms of Popt.

information for building models. That is to say, they can only use the data in the same metric space, our CDAA method can use data from different metric space. Compared with MD method which is suggested to compare by Zhou et al. [35], MD considers a larger module as more defect-proneness. Although MD can achieve better values of Recall, it also misclassifies more non-defective instances. Compared with SC: this method is an unsupervised clustering classifier, and its predictive power usually under-perform that of supervised classifiers. Compared with the WPDP with RF classifier: RF can obtain good prediction performance with sufficient historical data, which has the same distribution. However, our CDAA taking advantages of label information and self-learning ability of GANs can effectively transfer source instance space into target instance space.

VI. THREATS TO VALIDITY

A. THREATS TO CONSTRUCT VALIDITY
We constructs CDAA with a target training data by randomly sampling 10% of instances from target project. The experimental results are likely to be associated with the selected target training instances. Thus, in our experiment we run 20 times for each experimental object to mitigate this potential randomness.

In addition, the 28 projects studied in our experiments are gathered from open-source datasets by SZZ algorithm [39] to discover the defective modules. SZZ algorithm may loss some defective modules, which is a potential threat to the validity of construct. In future, the threat would be eliminated by employing completed defect data.

B. THREATS TO INTERNAL VALIDITY
Our study employs three comprehensive measures AUC, MCC and Popt to report the prediction results, and this would be a potential threat to the internal validity. In the future, other measures such as G-means, balance and ER will be measured.

In addition, for the compared approaches, we implement them carefully according to the published papers. Thus, experimental results may have some bias from the original approaches.

C. THREATS TO EXTERNAL VALIDITY
Experimental objects are from five public available repositories, which are used in many studies in software engineering. However, they would not be generalized to other private projects. Furthermore, conducting experiments on private projects will reduce this threat.

VII. CONCLUSION
Heterogeneous defect prediction (HDP) is an important research topic, since it is applicable to construct the prediction model on new projects that don’t have the same metric set with source project. To improve the performance of HDP, motivated by GANs which has strong learning ability to different domain, we propose conditional domain adversarial adaptation (CDAA) approach to tackle heterogeneous problem in SDP. CDAA has three networks of generator, discriminator and classifier, which could take full advantage of the information included in source project and small label instances in target project.

We conduct abundant experiments on 28 projects from five repositories to investigate the performance in both classification and effort-aware contexts. Further, the non-parametric Wilcoxon-signed rank test [43], [44] and the cohen’s d test are used. Experimental results indicate that CDAA could improve the performance of HDP.

In the future, we will apply more software projects including commercial projects to evaluate our method. In addition, there is class imbalance problem that influences the prediction results of learning models, we would like to address the class-imbalance for HDP.

ACKNOWLEDGMENT
This paper would not have been possible without the generous support by joint Ph.D. program of double first rate construction disciplines of CUMT.
[45] S. Hosseini, B. Turhan, and M. Mäntylä, “Search based training data selection for cross project defect prediction,” in Proc. The 12th Int. Conf. Predictive Models Data Analytics Softw. Eng. (PROMISE), 2016, pp. 1–10.

[46] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” IEEE Trans. Softw. Eng., vol. 43, no. 1, pp. 1–18, Jan. 2017.

[47] B. Ghotra, S. McIntosh, and A. E. Hassan, “Revisiting the impact of classification techniques on the performance of defect prediction models,” in Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng., Florence, Italy, May 2015, pp. 1–12.

[48] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The impact of using regression models to build defect classifiers,” in Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR), May 2017, pp. 135–145.

[49] L. Gong, S. Jiang, Q. Yu, and L. Jiang, “Unsupervised deep domain adaptation for heterogeneous defect prediction,” IEICE Trans. Inf. Syst., vol. 102, no. 3, pp. 537–549, Mar. 2019.

LINA GONG is currently a Ph.D. Student with the School of Computer Science and Technology, China University of Mining and Technology, China, and a Visitor with the School of Computing, Queen’s University, Canada. Her research interests include machine learning, software analysis, software testing, and mining software repositories.

SHUJUAN JIANG (Member, IEEE) received the Ph.D. degree from Southeast University, in 2006. From September 2008 to April 2009, she was a Visiting Scholar with the Georgia Institute of Technology. She is currently a Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, China University of Mining and Technology. Her research interests include compilation techniques, software engineering, and so on.

LI JIANG (Member, IEEE) is currently pursuing the M.S. degree with the School of Computer Science and Technology, China University of Mining and Technology. Her research interests include software analysis and testing and machine learning.

***