Heterogeneous Defect Prediction via Exploiting Correlation Subspace

Ming Cheng[1], Guoqing Wu[1], Min Jiang[2], Hongyan Wan[1] , Guoan You[1] and Mengting Yuan[1]
1. State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, China
{chengming, wgq, why0511, 2014202110064, ymt}@whu.edu.cn
2. Department of Cognitive Science and Technology and Fujian Key Laboratory of Brain-Like Intelligent Systems, Xiamen University, Xiamen 361005, China
minjiang@xmu.edu.cn

Abstract—**Software defect prediction generally builds models from intra-project data. Lack of training data at the early stage of software testing limits the efficiency of prediction in practice. Thereby researchers proposed cross-project defect prediction using the data from other projects. Most previous efforts assumed the cross-project defect data have the same metrics set which means the metrics used and size of metrics set are same in the data of projects. However, in real scenarios, this assumption may not hold. In addition, software defect datasets have the class imbalance problem increasing the difficulty for the learner to predict defects. In this paper, we advance canonical correlation analysis for deriving a joint feature space for associating cross-project data and propose a novel support vector machine algorithm which incorporates the correlation transfer information into classifier design for cross-project prediction. Moreover, we take different misclassification costs into consideration to make the classification inclining to classify a module as a defective one, alleviating the impact of imbalanced data. Experiments on public heterogeneous datasets from different projects show that our method is more effective, compared to state-of-the-art methods.**

Keywords-defect prediction; heterogeneous metrics; class imbalance; canonical correlation analysis; support vector machine

I. INTRODUCTION

Recently, most effective software defect prediction approaches have been proposed and attracted a lot of attention from academic and industrial communities. Most prior studies generally focused on Within-Project Defect Prediction (WPDP), which trained prediction model from historical data to detect the defect proneness of new software modules within the same project [1][2][3][4]. However, researchers often confront the situations that there are not enough historical data available, and they have to resort to the data from other projects to aid the learning of the target projects. Owing to this reason, Cross-Project Defect Prediction (CPDP) is proposed. It is the art of using data of inter-project to predict software defects in the target project with a small amount of local data [5][6].

Existing CPDP approaches are based on the underlying assumption that both source and target project data should exhibit the same data distribution or are drawn from the same feature space (i.e., the same software metrics). When the distribution of the data changes, or when the metrics features for source and target projects are different, one cannot expect the resulting prediction performance to be satisfactory. We consider this scenarios as Heterogeneous Cross-Project Defect Prediction (HCPDP) [7][8].

Mostly, the software defect data sets are imbalanced, which means the number of the defective modules is usually much smaller than that of the defective-free modules [9][10]. The imbalanced nature of data can cause poor prediction performance. That is, the probability of defect prediction can be low while the overall performance is high. Without taking this issue into account, the effectiveness of software defect prediction in many real-world tasks would be greatly reduced.

Recently, some researchers have noticed the importance of these problems in software defect prediction. For example, Nam et al. [7] used the metrics selection and metrics matching to select similar metrics for building a prediction model with heterogeneous metrics set. They discarded dissimilar metrics, which may contain useful information for training. Jing et al. [8] introduced Canonical Correlation Analysis (CCA) into HCPDP, by constructing the common correlation space to associate cross-project data. Then, one can simply project the source and target project data into this space for defect prediction. Like previous CPDP methods, the class imbalance problem of software defect datasets was not taken into account. Ryu et al. [10] designed the Value-Cognitive Boosting with Support Vector Machine algorithm (VCB-SVM) which exploited sampling techniques to solve the class imbalance issue for cross-project environments. Nevertheless, sampling strategy alters the distribution of the original data, where it may discard some potentially useful samples that could be important for prediction process. Therefore, these methods are not good solutions for addressing the class imbalance issue under heterogeneous cross-project environments.

Motivated by these observations, we advance CCA for driving a joint feature space for associating cross-project data, and incorporate the correlation transfer information in the derived subspace for improved prediction performance. During the defect prediction process, the two types of misclassification errors are encountered. Type I misclassifies a defective-free module as defective one (increasing the development cost), while type II misclassifies a defective module as defective-free one (leading to the more risk cost). Hence, we can take different misclassification costs into consideration by incorporating the cost factors into the SVM model. Fig. 1 shows a summary of our method.

In this paper, we propose a novel Cost-sensitive Correlation Transfer Support Vector Machine (CCT-SVM) algorithm to

DOI reference number: 10.18293/SEKE2016-090

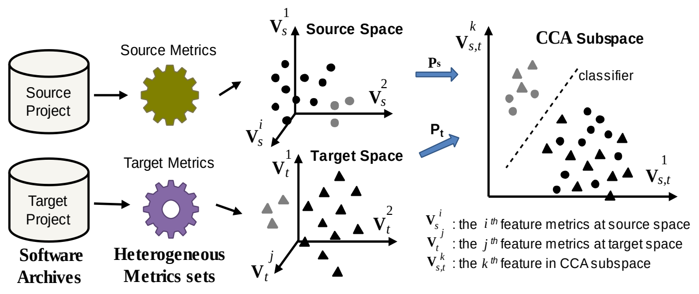

Figure 1. The overview of heterogenous cross-project prediction

deal with the class imbalance problem under HCPDP settings. Our contributions to the current state of research are summarized as follows.

- We advance CCA for deriving a joint feature space to associate cross-project data, so the correlation transfer information can be exploited to improve the prediction performance.
- During the software defect prediction process, we emphasize the risk cost to make the classification inclining to classify a module as a defective one, alleviating the impact of imbalanced data.
- Conduct an extensive and large-scale empirical study to evaluate our method.

II. RELATED WORK

A. Cross-project Defect Prediction

Software defect prediction employs historical data on reported (or repaired) defects to predict the location of previously unknown defects that hide in the code. However, sufficient historical training data from the same project is not always available or is hard to collect in practice. To address this issue, researchers proposed CPDP as an alternative solution in the last few years [11] [12] [13].

Zimmermann et al. [11] evaluated cross-project prediction performance based on large-scale experiments for 12 real-world projects. It indicated that current CPDP models do not perform well in most cases. Similar to Zimmermann' work, He et al. [12] investigated CPDP by focusing on training data selection. They pointed out that the prediction performance was related to the distributional attributes of datasets. Turhan et al. [13] proposed a Nearest-Neighbor filter method (NN-filter) to select similar data from source project. They only used nearest neighbors for each test data to construct training set, which have similar features to local data. Unlike the prior work selecting training data which are similar from the test data, Ma et al. [5] proposed Transfer Naive Bayes (TNB), by using the information of all the proper features in training data. The TNB transferred cross-project data information into the weights of the training data. Based on these weighted data, the prediction model was built. Nam et al. [6] applied Transfer Component Analysis (TCA) to CPDP and proposed TCA+ by selecting a suitable normalization automatically to preprocess data.

However, existing CPDP approaches are based on the assumption that the source and target data have the same software metrics set. When the metrics features for source and target projects are different, these methods will be unavailable. Recently, Nam et al. [7] proposed the Heterogeneous Defect Prediction (HDP) to predict defects across projects with heterogeneous metrics sets. They conducted metrics selection and metrics matching to build a prediction model using 28 projects collected from public defect datasets. Jing et al. [8] introduced CCA into CPDP to make the data distributions of source and target projects similar. They extended CCA to CCA+, by unified metric representation technique to preprocess data, so that the correlation between the projected data in CCA space is maximized.

Previous work proposed various defect prediction models under heterogeneous cross-project settings, but the class imbalance issue of defect datasets was not taken into account. Hall et al. [14] pointed out that data imbalance with regard to specific classification approaches may produce poor performance. Ignoring this issue, a learner that minimizes the prediction error would often produce a useless predictive model that predicts all the modules as defect-free.

B. Class Imbalance Learning

Class imbalance learning refers to learning from data that exhibit significant imbalance among classes. The challenge of class imbalance is that relatively underrepresented class cannot draw equal attention to the learning algorithm compared with the majority class, which often leads poor prediction performance [15]. To achieve better sensitivity to the minority class, the class imbalance issue should be explicitly tackled.

Wang et al. [15] explored the impact of class imbalance issue and provided guidance and valuable information for designing good predictor for software defects. Zheng et al. [16] employed three cost-sensitive boosting neural network algorithms for software defect prediction and found that threshold-moving algorithm was the best. Grbac et al. [17] studied the performance of machine learning techniques with different level of imbalance for software defect data. The feature selection and data sampling were exploit together. This method addressed class imbalance by modifying the training data. Ren et al. [18] proposed kernel based prediction method to address the class imbalance. The NASA and SOFTLAB datasets were used for experiments. Sun et al. [19] presented a coding based ensemble learning method, which converted class imbalance data into balanced multiclass data with specific coding scheme to avoid the class imbalance problem. Ryu et al. [10] first investigated whether the class imbalance learning can improve the prediction performance under CPDP settings. And they designed the value-cognitive boosting with support vector machine algorithm dealing with the class imbalance issue for cross-project environments. Experimental results showed that the class imbalance learning can be beneficial for CPDP.

III. HETEROGENEOUS SOFTWARE DEFECT PREDICTION

In this section, we present our method for HCPDP, which includes two parts: unified metrics representation and the CCT-SVM model.

A. Unified Metric Representation for Heterogeneous Data

To effectively utilize the heterogeneous metrics features from cross-project data, Jing et al. [8] proposed a Unified Metric Representation (UMR) to make the heterogeneous data can be compared. Based on the UMR, the standard CCA was exploited to find a common space for data from source and target project such that the correlation between the projected data in that space was maximized.

Similar to [8], we also exploit the UMR technique to make heterogeneous data to be compared. Suppose that $\mathbf{X}_s = \{\mathbf{x}_s^1, \mathbf{x}_s^2, \cdots, \mathbf{x}_s^N\} \in \mathbb{R}^{d_s \times N}$ and $\mathbf{X}_t = \{\mathbf{x}_t^1, \mathbf{x}_t^2, \cdots, \mathbf{x}_t^M\} \in \mathbb{R}^{d_t \times M}$ separately denote the source and target project data, where $\mathbf{x}_s^i$ indicates the i^{th} model in $\mathbf{X}_s$, N and M denote the numbers of modules in $\mathbf{X}_s$ and $\mathbf{X}_t$, respectively. $\mathbf{x}_s^i = [a_s^{i1}, a_s^{i2}, \cdots, a_s^{id_s}]$ and $\mathbf{x}_t^i = [a_t^{i1}, a_t^{i2}, \cdots, a_t^{id_t}]$ represent a module in the source and target project, where a_s^{ij} indicates the j^{th} metrics feature of the i^{th} model in source projcet, d_s and d_t are the numbers of metrics in $\mathbf{X}_s$ and $\mathbf{X}_t$, $d_s \neq d_t$. Here, we exploit UMR to restructure data as follows:

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

where the $\mathbf{X}_s^c$ and $\mathbf{X}_t^c$ are the same common metrics, $\mathbf{X}_s^s$ and $\mathbf{X}_t^t$ are specific metrics in source and target project, respectively. Note that if there exist no common metrics, then $\mathbf{X}_s^c = \mathbf{X}_t^c = 0$.

B. Learning Correlation Subspace via CCA

Based on the obtained UMR for heterogeneous data, we employ CCA technique to determine a common representation (*e.g.* a joint subspace) for features extracted from source and target projects, so that the model trained in the source project can be applied to detect the test modules in the target project. CCA learns two projection vectors $\mathbf{p}_s \in \mathbb{R}^{d_s}$ and $\mathbf{p}_t \in \mathbb{R}^{d_t}$, which maximize the following linear correlation coefficient ρ:

$$\max_{\mathbf{p}_s, \mathbf{p}_t} \rho = \frac{\mathbf{p}_s^T \Sigma_{st} \mathbf{p}_t}{\sqrt{\mathbf{p}_s^T \Sigma_{ss} \mathbf{p}_s}\sqrt{\mathbf{p}_t^T \Sigma_{tt} \mathbf{p}_t}} \tag{2}$$

where Σ_{ss} and Σ_{tt} represent the within-project covariance matrices of $\bar{\mathbf{X}}_s$ and $\bar{\mathbf{X}}_t$ respectively, while $\Sigma_{st} = \Sigma_{ts}$ represents the cross-project covariance matrix of $\bar{\mathbf{X}}_s$ and $\bar{\mathbf{X}}_t$. Σ_{ss}, Σ_{tt} and Σ_{st} are separately defined as:

$$\Sigma_{ss} = \frac{1}{N}\sum_{i=1}^{N}(\bar{\mathbf{x}}_s^i - \mathbf{m}_s)(\bar{\mathbf{x}}_s^i - \mathbf{m}_s)^T \tag{3}$$

$$\Sigma_{tt} = \frac{1}{M}\sum_{i=1}^{M}(\bar{\mathbf{x}}_t^i - \mathbf{m}_t)(\bar{\mathbf{x}}_t^i - \mathbf{m}_t)^T \tag{4}$$

$$\Sigma_{st} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(\bar{\mathbf{x}}_s^i - \mathbf{m}_s)(\bar{\mathbf{x}}_t^j - \mathbf{m}_t)^T \tag{5}$$

where $\bar{\mathbf{x}}_s^i$ represents the i^{th} software module in $\bar{\mathbf{X}}_s$, $\mathbf{m}_s$ and $\mathbf{m}_t$ are the mean modules of $\bar{\mathbf{X}}_s$ and $\bar{\mathbf{X}}_t$. As proved in [8], the optimization of (2) can be solved as a generalized eigenvalue decomposition problem:

$$\begin{pmatrix} 0 & \Sigma_{st} \\ \Sigma_{st} & 0 \end{pmatrix}\begin{pmatrix} \mathbf{p}_s \\ \mathbf{p}_t \end{pmatrix} = \lambda \begin{pmatrix} \Sigma_{ss} & 0 \\ 0 & \Sigma_{tt} \end{pmatrix}\begin{pmatrix} \mathbf{p}_s \\ \mathbf{p}_t \end{pmatrix} \tag{6}$$

The λ is the generalized eigenvalue corresponding to the generalized eigenvector $\begin{pmatrix} \mathbf{p}_s \\ \mathbf{p}_t \end{pmatrix}$. Generally, we can derive more than one pair of canonical components $\{\mathbf{p}_s^k\}_{k=1}^{d_v}$ and $\{\mathbf{p}_t^k\}_{k=1}^{d_v}$ with corresponding ρ_i in a descending order ($\rho_i > \rho_{i+1}$). Note that d_v is the dimension number of the correlation subspace of CCA. We can construct the projective transformation matrices $\mathbf{P}_s = [\mathbf{p}_s^i, \cdots, \mathbf{p}_s^{d_v}] \in \mathbb{R}^{d_s \times d_v}$ and $\mathbf{P}_t = [\mathbf{p}_t^i, \cdots, \mathbf{p}_t^{d_v}] \in \mathbb{R}^{d_t \times d_v}$. Once the correlation subspace is derived, the test modules at target project can be directly detected by the model learned from the source project data projected onto the subspace.

C. Cost-sensitive Correlation-transfer SVM

In derived CCA subspace, each dimension $\mathbf{v}_{s,t}^i$ is associated with a different correlation coefficient ρ_i. A higher ρ_i denotes a better correlation, which results in a better transfer ability for the associated dimension $\mathbf{v}_{s,t}^i$. On the other hand, poorer transfer ability will increase classification error, even the classifier is trained using the projected source project data. Obviously, higher correlation coefficient can obtain more class discriminant information which is more useful for constructing classifiers [20].

In SVM, if the i^{th} feature attribute has the better discrimination ability, the classical SVM could produce a larger magnitude for the corresponding model (e.g., a larger $|w_i|$). Here, we introduce a correlation regularizer and propose a linear SVM model which integrates the cross-project transfer ability and class discrimination in a unified formulation. Moreover, we employ a specifical misclassification costs to minimize a classification-oriented loss. We set two misclassification cost values C^+ and C^-. C^+ is the misclassification cost for the defective modules, while C^- is the misclassification cost for the defective-free modules. By assigning a higher misclassification cost for the minority defective modules than the majority defective-free modules (i.e., $\mathrm{C}^+ > \mathrm{C}^-$), the effect of class imbalance could be reduced. Then, the modified SVM decision function can be represented as follows:

$$\min_{\mathbf{w}} \left(\frac{1}{2}\|\mathbf{w}\|_2^2 + \mathrm{C}^+ \sum_{[i|y_i=+1]}^{\mathrm{N}} \varepsilon_i + \mathrm{C}^- \sum_{[i|y_i=-1]}^{\mathrm{N}} \varepsilon_i - \Phi(\rho_i) \right)$$

$$s.t.\ \ y_i\left(\langle \mathbf{w}, \mathbf{P}_s^T \bar{\mathbf{x}}_s^i \rangle + b\right) \geq 1 - \varepsilon_i,\ \varepsilon_i \geq 0,\ \forall (\bar{\mathbf{x}}_s^i, y_i) \in \mathcal{D}_l^s \tag{7}$$

where $\Phi(\rho_i) = \frac{1}{2} Abs(\mathbf{w})\boldsymbol{r}^T$, $Abs(\mathbf{w}) = [|w_1|, |w_2|, \cdots, |w_{d_v}|]$ and $\boldsymbol{r} = [\rho_1, \cdots, \rho_{d_v}]$ is the correlation vector in which each element denotes the correlation coefficient of CCA subspace for each pair of projection dimension. Parameter ε_i is slack variable as in standard SVM. It should be noted that only labeled source data $\bar{\mathbf{x}}_s^i \in \mathcal{D}_l^s$ is available for training, and y_i is the associated class label. We put $\mathbf{P}_s^T \bar{\mathbf{x}}_s^i$ as the projection of source project data $\bar{\mathbf{x}}_s^i$ onto the correlation subspace.

In (7), the term $\Phi(\rho_i)$ is introduced for model adaptation based on CCA. By doing so, a smaller correlation coefficient ρ_i is obtained for the i^{th} dimension of CCA subspace, then the above equation would enforce the reduction of the corresponding $|w_i|$ and restrict the trained SVM model along that dimension. On the other side, a larger ρ_i favors the contribution of the associated $|w_i|$ when minimizing (7).

Since it is not forthright to solve the minimization problem in (7), we seek the approximated solution by modifying the correlation regularizer term $\Phi(\rho_i)$ into the following form:

$$\Phi(\rho_i) = \frac{1}{2}\, Abs(\mathbf{w} \odot \mathbf{w})(\boldsymbol{r} \odot \boldsymbol{r})^T \tag{8}$$

where $\odot$ denotes the element-wise multiplication. By incorporating (8) into (7), the objective function can be rewritten into a unified form:

$$\min_{\mathbf{w}} \left(\frac{1}{2} \sum_{i=1}^{d_c} (1-\rho_i^2) w_i^2 + \mathrm{C}^+ \sum_{[i|y_i=+1]}^{\mathrm{N}} \varepsilon_i + \mathrm{C}^- \sum_{[i|y_i=-1]}^{\mathrm{N}} \varepsilon_i \right)$$

$$s.t. \quad y_i \left(\langle \mathbf{w}, \mathbf{P}_s^T \bar{\mathbf{x}}_s^i \rangle + b \right) \geq 1 - \varepsilon_i,\ \varepsilon_i \geq 0,\ \forall (\bar{\mathbf{x}}_s^i, y_i) \in \mathcal{D}_l^s \qquad (9)$$

We refer to (9) as our cost-sensitive correlation transfer SVM. Since the correlation coefficient ρ_i ranges from 0 to 1, the above object function is a convex optimization problem. We apply the Newton-Armijo algorithm for solving SVM optimization problems. As a result, our modified SVM could adapt the derived classification model $\mathbf{w}$ relied on the cross-project transfer ability of CCA. Moreover, we assign two misclassification costs to alleviate the effect of class imbalance. Thus, it can present the better classification performance in correlation subspace. The decision function for classifying the test modules in target project is shown as follows:

$$f(\mathbf{x}) = sgn(\langle \mathbf{w}, \mathbf{P}_t^T \bar{\mathbf{x}}_t \rangle + b) \qquad (10)$$

where $\mathbf{P}_t^T$ projects the target data $\bar{\mathbf{x}}_t$ from the target space onto the correlation subspace. Finally, $sgn(z)$ returns 1 if and only if $z > 0$, and -1 otherwise.

IV. Experiments

A. Experimental Datasets

We collect publicly available defect datasets from prior researches, including NASA, SOFTLAB, AEEEM and ReLink [7][8]. Among these datasets, the percentage of defective components ranges from 8.65% to 50.52%. It is obvious that most datasets are imbalanced. Table I shows detailed project information in our experiments.

B. Experimental Design

To validate the effectiveness of the proposed approach for HCPDP, we compare our approach with several representative methods including TNB [5], TCA+ [6], CCA+ [8] and NN-filter [13]. We design the three experiments to evaluate our approach: (1) HCPDP with partially different metrics. We build model using common metrics between source and target datasets as in previous studies [8]. (2) HCPDP with entire different metrics. In this part, we present CCA+ and the within-project defect prediction results as references. (3) The impact of different class-imbalance rates on HCPDP, exploring whether or not our proposed method can effectively deal with class-imbalance problem in HCPDP.

For WPDP, we employ the standard SVM as their base classifier. We use the 50:50 random splits to obtain training and test sets. Thus, we repeat this process 30 times to get the average prediction results. In our approach, in order to emphasize the risk cost, the parameters C^+ and C^- are set as $\mathrm{C}^+:\mathrm{C}^-$=5:1. For different projects, user can select different ratios [21]. The parameter ε is determined by searching a wide range and choosing the one which produces the best F-measure value. Although we have verified that these choices of parameters work well in our experiments, we recognize that a finer tuning of them may further improve the performance.

To evaluate the performance of our method, we use two widely-used evaluation measures, F-measure and Area Under the Receiver Operating Characteristic (ROC) Curve (AUC). F-measure is the harmonic mean of *precision* and *recall*, falling in the range [0, 1], as (11). The *recall* is defined as the ratio of the number of modules correctly classified as defect to the number of defective modules. The *precision* is the ratio of the number of modules correctly classified as defect to the number of modules classified as defect.

TABLE I. Heterogeneous datasets from different projects

Group	Dataset	Instances	Buggy (%)	Metrics
NASA	CM1	327	42(12.84%)	37
	MW1	253	27(10.67%)	
	PC1	705	61(8.65%)	
AEEEM	EQ	325	129(39.69%)	61
	JDT	997	206(20.66%)	
	LC	399	64(9.26%)	
	ML	1862	245(13.16%)	
	PDE	1492	209(14.01%)	
ReLink	Apache	194	98(50.52%)	26
	Safe	56	22(39.29%)	
	ZXing	399	118(29.57%)	
SOFTLAB	AR3	63	8(12.72%)	29
	AR4	107	20(18.69%)	
	AR5	36	8(22.22%)	

$$F-measure = \frac{2 \times recall \times precision}{recall + precision} \qquad (11)$$

For more comprehensive evaluation of predictors in the imbalanced context, the AUC is exploited to evaluate the prediction performance. AUC estimates the area under the ROC curve, which illustrates the trade-off between detection and false alarm rates, varying in [0, 1]. A better classifier should produce a higher F-measure and AUC.

C. Experimental results

Table II and Table III show that the F-measure and AUC values of heterogeneous defect prediction, where 28 common metrics exist in source and target data. In each table, the best performance are presented with boldface in all experimental datasets. The last rows of tables show the average performances on all the experimental datasets.

From Table II, we see that CCT-SVM can obtain better F-measure values in most prediction scenes, compared with other methods, and the average F-measure of CCT-SVM is the highest. This indicates that after carefully taking the class imbalance nature of defect data into consideration, CCT-SVM is able to improve defect performance dramatically. The trends in Table III are similar to that shown in Table II. The reason is that our method uses all metrics rather than only common metrics, and these metrics usually contain some useful discriminant information. We exploit the cross-project transfer ability in derived subspace when designing the associated SVM classifier. And we emphasize the risk cost to make the classification inclining to classify a module as a defective one, alleviating the impact of imbalanced data. Wilcoxon's rank sum test at a 0.05 significance level indicates that performance improvement on each pair dataset is statistical significance. This fact suggests that addressing the class imbalance problem is beneficial to construct better predictive model in software defect prediction.

In practical scenario, we often confront the situations that there are no common metrics between source and target project data. Hence we conduct experiment to investigate the performance of CCT-SVM with entire different metrics. In this section, we conduct within-project prediction results of a target project as baseline. Specifically for each dataset, we randomly

TABLE II. Median F-measures with 28 common metricss

Source⇒Target	**NN**	**TNB**	**TCA+**	**CCA +**	**CCT–SVM**
CM1 ⇒ AR3	0.403	0.271	0.333	0.582	**0.612**
CM1 ⇒ AR4	0.632	0.337	0.416	**0.772**	0.756
CM1 ⇒ AR5	0.293	0.325	0.376	0.686	**0.711**
PC1 ⇒ AR3	0.596	0.467	0.323	0.791	**0.802**
PC1 ⇒ AR4	0.551	0.331	0.373	0.716	**0.733**
PC1 ⇒ AR5	0.512	0.379	0.516	**0.723**	0.719
MW1 ⇒ AR4	0.591	0.326	0.396	0.689	**0.707**
AR4 ⇒ CM1	0.256	0.296	0.279	0.781	**0.786**
AR4 ⇒ PC1	0.281	0.337	0.219	0.712	**0.751**
AR4 ⇒ MW1	0.523	0.386	0.433	0.768	**0.781**
AVG	0.463	0.346	0.366	0.722	**0.736**

TABLE III. Median AUCs with 28 common metricss

Source⇒Target	**NN**	**TNB**	**TCA+**	**CCA +**	**CCT–SVM**
CM1 ⇒ AR3	0.583	0.556	0.543	0.692	**0.703**
CM1 ⇒ AR4	0.550	0.509	0.539	**0.709**	0.705
CM1 ⇒ AR5	0.581	0.627	0.619	0.744	**0.771**
PC1 ⇒ AR3	0.601	0.593	0.639	0.751	**0.763**
PC1 ⇒ AR4	0.621	0.608	0.613	0.753	**0.759**
PC1 ⇒ AR5	0.653	0.686	0.673	0.839	**0.856**
MW1 ⇒ AR4	0.507	0.567	0.556	0.690	**0.705**
AR4 ⇒ CM1	0.501	0.530	0.522	**0.694**	0.691
AR4 ⇒ PC1	0.431	0.489	0.453	0.571	**0.584**
AR4 ⇒ MW1	0.473	0.516	0.513	0.627	**0.631**
AVG	0.550	0.570	0.567	0.707	**0.721**

choose the 50% samples as the training data and the other 50% are testing data. We repeat this process 30 times and report the average prediction results.

Table IV and V show the F-measure and AUC of different compared methods, where no common metrics exist in the source and target data. From Table IV and Table V, we can see that CCT-SVM can obtain better results in contrast with the CCA+ and within-project prediction in most cases. The results suggest that our method takes the misclassification costs into consideration, which makes the prediction tending to classify the defective-free modules as the defective ones in order to get higher prediction performance. Table V tabulates the AUC values. The trends of AUC values in Table V are similar to that of F-measure shown in Table IV. Therefore, CCT-SVM can be used to address HCPDP effectively.

CCT-SVM can effectively address heterogeneous defect prediction problem even if the class distribution is imbalanced. In order to study the influence of the different class-imbalance rates on CCT-SVM under heterogeneous cross-project setting, we conduct additional experiments, where we alter the different classes distribution of the source data which is customized so that the number of the defective samples over the number of the defective-free samples is roughly δ, $1/\delta \in \{1,2,\cdots 10\}$. If the original proportion is larger than δ, we randomly abandon some defective samples; otherwise, we randomly abandon some defective-free samples. Here, we build a prediction model by using the customized source project data and then apply the model to the target project data.

We repeat the experiment in each customized dataset for 30 times. We plot the average F-measures and AUCs versus the inverse of the minority-majority ratio ($1/\delta$) on the experimental datasets. We only report the experimental results on the three pair representative datasets: MW1⇒ AR4 (28 common metrics), ZXing⇒AR4 (3 common metrics), JDT⇒ ZXing (no common metrics), as shown in Fig. 2-4.

TABLE IV. Median F-measures with no common metrics

Source⇒Target	**CCA +**	**CCT–SVM**	**Within Target⇒Target**
CM1 ⇒ EQ	0.581	**0.612**	0.576
EQ ⇒ CM1	0.238	0.276	**0.336**
LC ⇒ Apache	0.266	0.307	**0.645**
Apache ⇒ LC	0.288	0.291	**0.373**
ML ⇒ PC1	0.541	**0.567**	0.369
JDT ⇒ PC1	0.501	**0.523**	
PDE ⇒ PC1	0.431	**0.455**	
ML ⇒ AR4	**0.573**	0.559	0.392
JDT ⇒ AR4	0.493	**0.517**	
PDE ⇒ AR4	**0.540**	0.536	
PC1 ⇒ ML	**0.336**	0.333	0.273
AR4 ⇒ ML	0.353	**0.336**	
ZXing ⇒ ML	0.405	**0.446**	
PC1 ⇒ JDT	0.521	0.533	**0.563**
AR4 ⇒ JDT	0.592	**0.631**	
ZXing ⇒ JDT	0.647	**0.675**	
PC1 ⇒ PDE	0.376	**0.401**	0.312
AR4 ⇒ PDE	0.383	**0.413**	
ZXing ⇒ PDE	0.421	**0.473**	
ML ⇒ ZXing	0.486	**0.501**	0.336
JDT ⇒ ZXing	0.466	**0.508**	
PDE ⇒ ZXing	0.474	**0.497**	
AVG	0.451	**0.475**	0.394

TABLE V. Median AUCs with no common metrics

Source⇒Target	**CCA +**	**CCT–SVM**	**Within Target⇒Target**
CM1 ⇒ EQ	0.711	**0.752**	0.651
EQ ⇒ CM1	0.798	**0.816**	0.728
LC ⇒ Apache	**0.806**	0.797	0.769
Apache ⇒ LC	0.718	**0.757**	0.608
ML ⇒ PC1	**0.861**	0.827	0.796
JDT ⇒ PC1	0.759	**0.823**	
PDE ⇒ PC1	0.791	**0.815**	
ML ⇒ AR4	0.765	**0.809**	0.654
JDT ⇒ AR4	0.676	**0.717**	
PDE ⇒ AR4	0.730	**0.766**	
PC1 ⇒ ML	0.673	0.726	**0.754**
AR4 ⇒ ML	0.653	0.696	
ZXing ⇒ ML	0.725	**0.746**	
PC1 ⇒ JDT	0.720	0.751	**0.809**
AR4 ⇒ JDT	0.612	0.679	
ZXing ⇒ JDT	0.751	**0.883**	
PC1 ⇒ PDE	0.702	**0.727**	0.711
AR4 ⇒ PDE	0.681	**0.719**	
ZXing ⇒ PDE	**0.731**	0.701	
ML ⇒ ZXing	**0.684**	0.652	0.609
JDT ⇒ ZXing	0.667	**0.723**	
PDE ⇒ ZXing	**0.721**	0.707	
AVG	0.724	**0.751**	0.715

As expected, F-measure and AUC values of all the compared methods decrease as the dataset becomes more imbalanced, but the influence of the increase of class imbalance on CCT-SVM is the smallest. Fig. 2-4 show that CCT-SVM almost always performs better than the other methods, and when the class distribution is more imbalanced the superiority is more preponderant. This fact suggests that the degree of imbalance has great influence on HCPDP, if it does not address the class imbalance problem explicitly. Therefore, it can be concluded that explicitly tackling the class-imbalance problem is helpful to HCPDP.

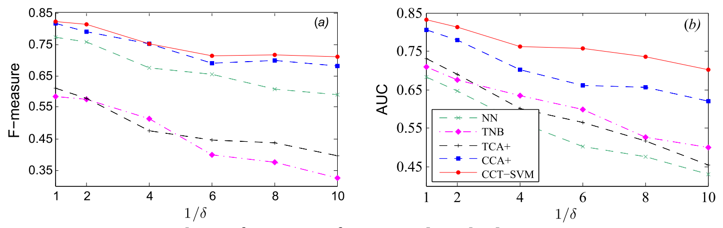

Figure 2. The performance of compared methods on MW1⇒AR4 (28 common metrics) at different minority-majority.

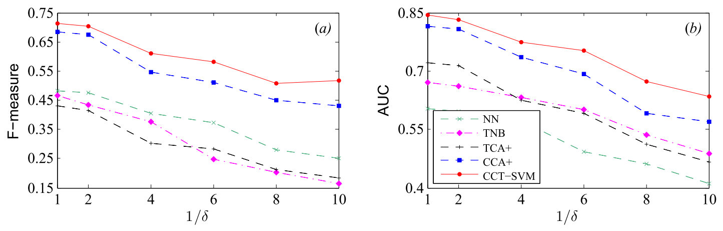

Figure 3. The performance of compared methods on ZXing⇒AR4 (3 common metrics) at different minority-majority

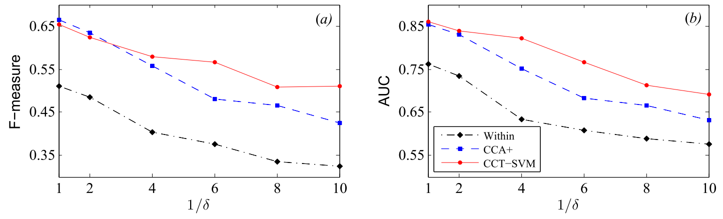

Figure 4. The performance of compared methods on JDT⇒ZXing (no common metrics) at different minority-majority

V. CONCLUSION AND FUTURE WORK

Cross-project software defect prediction plays an important role in improving the quality of a software product in case of projects without sufficient historical data. However, it is difficult to conduct with heterogeneous metrics set. In addition, software defect datasets have the class-imbalance characteristic. Without taking this issue into account, the effectiveness of software defect prediction would be greatly reducing. In this paper, we addressed these two important issues simultaneously and proposed a novel cost-sensitive correlation transfer support vector machine method for heterogeneous defect prediction. Experimental results on the open source projects from different groups showed that our method is feasible and yields promising results.

For the future work, we will introduce other sophisticated class imbalance learning techniques in the heterogeneous cross-project defect prediction, and we will evaluate our approach in more heterogeneous defect datasets.

ACKNOWLEDGMENT

The research in this paper was partially supported by the National Natural Science Foundation of China under Projects No. 61373039, No. 61170022, No. 61003071 and No. 91118003).

REFERENCES

[1] G. Czibula, Z. Marian, and I. G. Czibula, “Software defect prediction using relational association rule mining,” Information Sciences, vol. 264, pp. 260–278, 2014.

[2] X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, “Dictionary learning based software defect prediction,” In Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 414–423.

[3] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” Software Engineering, IEEE Transactions on, vol. 33, no.1, pp. 2–13, 2007.

[4] I. H. Laradji, M. Alshayeb, and L. Ghouti. Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58:388–402, 2015.

[5] Y. Ma, G. C. Luo, X. Zeng and A. Chen, “Transfer learning for crosscompany software defect prediction,” Information and Software Technology, vol. 54, no. 3, pp. 248-256, 2012.

[6] J. Nam, S. J. Pan and S. Kim, “Transfer defect learning,” Proceedings of the 35th International Conference on Software Engineering . San Francisco, 2013, pp. 382-391.

[7] J. Nam and S. Kim, “Heterogeneous defect prediction,” In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 508–519.

[8] X. Y. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, “Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning,” In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 496–507.

[9] Y. Jiang, M. Li, and Z.-H. Zhou, “Software defect detection with rocus,” Journal of Computer Science and Technology, vol. 26, no. 2, pp. 328–342, 2011.

[10] D. Ryu, O. Choi, and J. Baik, “Value-cognitive boosting with a support vector machine for cross-project defect prediction,” Empirical Software Engineering, vol. 21, no. 1, pp. 43-71, 2016.

[11] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: a large scale experiment on data vs. domain vs. process,” In Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 91–100.

[12] 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.

[13] B. Turhan, T. Menzies, A. B. Bener, and J. Di 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.

[14] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic literature review on fault prediction performance in software engineering,” Software Engineering, IEEE Transactions on, vol. 38, no. 6, pp. 1276–1304, 2012.

[15] S. Wang and X. Yao, “Using class imbalance learning for software defect prediction,” Reliability, IEEE Transactions on, vol. 62, no. 2, pp. 434–443, 2013.

[16] J. Zheng, “Cost-sensitive boosting neural networks for software defect prediction,” Expert Systems with Applications, vol. 37, no. 6, pp. 4537–4543, 2010.

[17] T. G. Grbac, G. Mausa, and B. D. Basic, “Stability of software defect prediction in relation to levels of data imbalance,” In SQAMIA, 2013, pp. 1–10.

[18] J. Ren, K. Qin, Y. Ma, and G. Luo, “On software defect prediction using machine learning,” Journal of Applied Mathematics, 2014.

[19] Z. Sun, Q. Song, and X. Zhu, “Using coding-based ensemble learning to improve software defect prediction,” Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol.42, no. 6, pp. 1806–1817, 2012.

[20] Y. R. Yeh, C. H. Huang, and Y. C. F. Wang, “Heterogeneous domain adaptation and classification by exploiting the correlation subspace,” Image Processing, IEEE Transactions on, vol. 23, no. 5, pp. 2009–2018, 2014.

[21] Y. Jiang, B. Cukic, and T. Menzies, “Cost curve evaluation of fault prediction models,” In Software Reliability Engineering, ISSRE, 19th International Symposium on, pp. 197–206, 2008.