Applied Intelligence (2025) 55:324
https://doi.org/10.1007/s10489-024-06087-5

Cross-project defect prediction based on autoencoder with dynamic adversarial adaptation

Wen Zhang[1,2] · Jiangpeng Zhao[1] · Guangjie Qin[3] · Song Wang[4]

Accepted: 19 November 2024 / Published online: 15 January 2025

Abstract
Cross-project defect prediction enables a target software project with limited defect data to build a defect prediction model by leveraging abundant data in the source project. However, existing methods of cross-project defect prediction ignore the relative importance of global and local distributions in learning project-invariant feature spaces. This paper proposes a novel approach for cross-project defect prediction called Adan (autoencoder with dynamic adversarial adaptation) to dynamically adjust a project-invariant feature space by aligning global and local distributions simultaneously with adversarial learning. Specifically, the au-encoder was adopted to produce a latent space used as a project-invariant feature space for source and target artifacts. Global and local discriminators were used to adjust the latent space to ensure that representations of source and target artifacts in the project-invariant feature space have approximate global distribution and local distribution, respectively. The prediction model for the target artifacts was then trained using representations of the source artifacts in the project-invariant feature space. Experiments on four open-source projects with 12 pairs of tasks on cross-project defect prediction demonstrated that the proposed Adan approach outperformed state-of-the-art techniques, with an average improvement of 8.42% in terms of AUC.

Keywords Cross-project software defect prediction · Project-invariant feature space · Global discriminator · Local discriminator · Distribution alignment

1 Introduction

Defect prediction refers to the construction of a predictive model using historical defect data to predict possible defects in software artifacts, which is of great importance in software quality assurance [24, 31]. However, it often suffers from a cold-start problem due to a lack of historical defect data when defect prediction is adopted for a newly developed project or one at an early stage [12, 31]. Thus, cross-project software defect prediction has been proposed to make use of a large number of defects in a source project to train a defect-prediction model for a target project (which has a small number of defects). That is, cross-project defect prediction learns a predictive model from a source project with sufficient historical data on software defects and applies the predictive model to predict the defective artifacts of the target project with insufficient historical defect data [16]. Thus, effective cross-project defect prediction can help software project managers optimize the allocation of limited human resources and financial budgets to software quality assurance [36].

✉ Wen Zhang
zhangwen@bjut.edu.cn

Jiangpeng Zhao
zjp@emails.bjut.edu.cn

Guangjie Qin
qinguangjie@inspur.com

Song Wang
wangsong@yorku.ca

1 College of Economics and Management, Beijing University of Technology, Beijing 100124, People's Republic of China

2 Weiqiao-UCAS Science and Technology Park, Binzhou Institute of Technology, Shandong Province, Binzhou City 256606, People's Republic of China

3 Inspur New Infrastructure Technology, Shandong Province, Jinan 250013, People's Republic of China

4 Department of Electrical Engineering and Computer Science, York University, Toronto, ON M3J 1P3, Canada

Existing studies on cross-project defect prediction can be roughly categorized as sample filtering-based methods and transfer learning-based methods [15]. Sample filtering-based methods sample defects from a source project with a similar data distribution of a target project to augment training data [16]. Transfer learning-based methods attempt to learn a project-invariant feature space (i.e., a common space for the source and target projects to minimize their defect data distribution discrepancy) [12]. The study in this paper follows the stream of transfer learning-based methods (i.e., to learn a project-invariant feature space for both the source and target projects to mitigate the impact of its defect data distribution discrepancy on defect prediction). Then, a defect-prediction model is trained for the target project by transferring knowledge from the source project [41].

Traditional transfer learning-based methods usually make use of matrix decomposition in linear algebra to construct a project-invariant feature space across the source and target projects, such as transfer component analysis (TCA) and canonical correlation analysis (CCA) [4, 27, 28]. However, research gaps remain. First, existing transfer learning-based methods merely focus on aligning either global or local distribution between the source and target projects. Thus, they ignore the relative effect of the global and local distributions in engineering the project-invariant feature space. Second, existing transfer learning-based methods, such as TCA and CCA, usually suffer from the nonlinear distribution discrepancy between the source and the target projects. For instance, TCA makes use of linear mapping with a kernel mixture model to reduce the distribution discrepancy between source-project and target-project instances (Guo et al., 2024; [42]). CCA uses a linear combination of variables with maximal correlation to produce project-invariant features [20], Mehrkanoon et al., 2018).

Motivated by the abovementioned research gap, this paper proposes a novel approach called Adan (autoencoder with dynamic adversarial adaptation) to dynamically adjust the project-invariant feature space by aligning global and local distributions simultaneously with adversarial learning. The contributions of this paper are threefold. First, we propose an autoencoder and global and local discriminators to engineer project-invariant spaces for source and target artifacts. An autoencoder is adopted to produce the latent space as the project-invariant feature space for the source and target artifacts. The global discriminator and the local discriminator are proposed to dynamically adjust the project-invariant feature space to align the global and local distributions between the source and target artifacts in the project-invariant feature space through adversarial learning. Second, we adapted the focal loss to account for the imbalanced defect data in training the defect-prediction model using the source artifacts represented in the project-invariant feature space. We then customized an integrated loss function of the four components involved in the proposed Adan approach to simultaneously update the parameters of the autoencoder, the global discriminator, the local discriminator, and the defect-prediction model. Third, we collected the defect data from four open-source projects (Kafka, Kylin, Ant, and Tomcat) and paired 12 cross-project defect-prediction tasks to investigate the performance of the proposed Adan approach. All the data and source code in the paper can be accessed freely at https://github.com/xywen4bcut/guangjieqing2022.

The remainder of the paper is organized as follows. Section 2 presents the related work. Section 3 formulates the problem of cross-project defect prediction. Section 4 proposes the Adan approach. Section 5 conducts the experiments for the performance evaluation of the proposed Adan approach. Section 6 concludes this paper and indicates future work.

2 Related work

The related work of this paper includes two aspects of cross-project defect prediction. The first is sample filtering-based methods, and the second is transfer learning-based methods. In the first aspect, existing methods usually make use of a sampling technique to create an approximate sample set for the target project by filtering and weighting samples from the source project. For instance, Peter et al. (2013) proposed a two-stage sample filtering approach based on the k-nearest neighbor algorithm to make full use of samples from both source and target projects. Liu et al. [19] proposed a two-phase model for cross-project defect prediction. The first phase is to use a source-project estimator to automatically choose candidates from two source projects with the highest distribution similarity to a target project, and the second phase is to build two prediction models based on the two selected projects and combine their prediction results to improve prediction performance.

Tong et al. [34] used the maximal information coefficient for feature importance evaluation for feature filtering to improve the training of Naïve Bayes. Chen et al. [7] used min–max normalization and Z-score normalization based on the mean and standard deviation of the source and target projects to generate different distributions of samples in the feature space for source data expansion. Sotto-Mayor and Kalech [29] applied a mini-batch K-means clustering algorithm to reduce the search space of the complete set of instances from the target projects and to select the top k instances with the smallest distance from the target projects in the closest cluster to expand the training instances.

In the second aspect, existing methods make use of feature distribution alignment to minimize the distribution discrepancy of defect data between source and target projects by constructing a shared feature space. Nam et al. [23] proposed improved transfer component analysis (TCA+) to create a shared feature space for aligning the feature distribution discrepancy between source and target projects. Then, logistic regression is trained for cross-project software defect prediction with automatic normalization selection for TCA. Jing et al. [14] maintained that the data distributions of different projects are heterogeneous (i.e., the features, as well as the ranges of source and target projects, are different, and they propose a unified metric representation to alleviate the heterogeneity based on CCA).

Li et al. [15] proposed landmark selection-based kernelized discriminant subspace alignment for cross-project defect prediction to reduce the discrepancy in the data distributions between source and target projects and to characterize complex data structures to increase the probability of linear separability of the data. Jiang et al. [13] used TCA and supervised TCA for cross-project clone consistent-defect prediction,they found that expanding the quantity of data contributed to the enhancement of efficiency in prediction.

Existing studies have made significant achievements in cross-project defect prediction. There remains space for further research, especially on the dynamic alignment of the global and local distributions of source and target projects simultaneously. Existing methods merely focus on aligning either global or local distribution between source and target projects. These methods usually ignore the relative importance of global and local distributions when engineering project-invariant feature spaces. However, existing techniques, such as TCA and CCA, suffer from a nonlinear distribution discrepancy between the source and target projects and have limited capability in handling the complex nonlinear distribution discrepancy between source and target projects (Mehrkanoon et al., 2018; [38, 42]).

3 Problem statement

The goal of cross-project defect prediction is to learn a defect-prediction model using a large number of labeled source artifacts to predict defects for a small number of unlabeled target artifacts [4, 23]. Thus, cross-project software defect prediction is proposed to make use of the defects from various projects with abundant data, to increase the overall sample size of the training data, and to improve the generalization of the learned defect prediction model. That is, cross-project defect prediction learns a predictive model from the source project with sufficient historical data on software defects and applies the predictive model to predict the defective artifacts of the target project with insufficient historical defect data [16]. This paper proposes a solution for cross-project defect prediction by aligning the global and local distributions between the source and target projects dynamically and simultaneously.

Suppose we have a source project as $D^{(S)} = \{(x_i^{(S)}, y_i^{(S)}) | x_i^{(S)} \in X^{(S)}, y_i^{(S)} \in \{0,1\}\}_{i=1}^{N^{(S)}}$ containing $N^{(S)}$ labeled artifacts, where $x_i^{(S)}$ is a numerical vector denoting the i-th artifact in the source project and $y_i^{(S)}$ is a Boolean value denoting its label (as either defective as $y_i^{(S)} = 1$, or nondefective as $y_i^{(S)} = 0$). Meanwhile, we have a target software project as $D^{(E)} = \{(x_j^{(E)}) | x_j^{(E)} \in X^{(E)}\}_{j=1}^{N^{(E)}}$ containing $N^{(E)}$ unlabeled artifacts where $x_j^{(E)}$ is a numeric vector denoting the j-th artifact in the target project. Further, $X^{(S)}$ denotes the original feature space of the source project $D^{(S)}$ and $X^{(E)}$ denotes the original feature space of the target project $D^{(E)}$. The objective of cross-project defect prediction is to train a defect-prediction model G_y on the source project $D^{(S)}$ that can be generalized to predict the label of the target artifact $x_j^{(E)}$ in the target project $D^{(E)}$. For clarity, global distribution and local distribution are defined as follows for a better understanding of cross-project defect prediction, following Pan and Yang [25] and Liu et al. [18].

Global distribution $P(X)$ The global distribution $P(X)$ refers to the marginal distribution of the feature values X, without taking into account the class labels Y [18, 25]. It characterizes the global statistics of features in the dataset while ignoring class labels. The alignment of global distribution aims to learn a feature mapping G_f between the artifact $x_i^{(S)}$ in the source feature space $X^{(S)}$ and the target artifact $x_j^{(E)}$ in the target feature space $X^{(E)}$, so that $P(G_f(x_i^{(S)}))_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)}))_{j=1}^{N^{(E)}}$.

Local distribution $P(X|Y)$ The local distribution refers to the class-specific distribution of feature values X conditioned on the class labels Y [18, 25]. It characterizes the local statistics of features inside each class. The alignment of local distribution aims to learn a feature mapping G_f between the artifact $x_i^{(S)}$ in the source feature space $X^{(S)}$ and the target artifact $x_j^{(E)}$ in the target feature space $X^{(E)}$, so that $P(G_f(x_i^{(S)})|y = 0)_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)})|y = 0)_{j=1}^{N^{(E)}}$ and $P(G_f(x_i^{(S)})|y = 1)_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)})|y = 1)_{j=1}^{N^{(E)}}$.

With the above definition, the problem of cross-project defect prediction can be formulated as follows. For a given source project $D^{(S)}$ containing $N^{(S)}$ labeled artifacts $x_i^{(S)}$ and a target project $D^{(E)}$ containing $N^{(E)}$ unlabeled artifacts $x_j^{(E)}$, the problem is to train a defect prediction model G_y using the labeled source artifacts $x_i^{(S)} (1 \leq i \leq N^{(S)})$ to predict the labels (i.e., defective, or nondefective) of the target artifacts $x_j^{(E)} (1 \leq j \leq N^{(E)})$ by learning the feature mapping G_f to project both the source artifacts target artifacts into a common space with the alignment of both global and local distributions.

4 The proposed approach: Adan

4.1 Overall architecture

The basic idea of the proposed Adan approach is to construct a project-invariant feature space for both source and target artifacts and then to use representations of the source artifacts in the project-invariant feature space to predict defects for the target artifacts. Figure 1 illustrates the overall architecture of the proposed Adan approach, which consists of four components: feature autoencoder, global discriminator, local discriminator, and defect predictor.

The feature autoencoder comprises the feature encoder G_f and the feature decoder G_r where G_f is used to encode the source or target artifact $x_k^{(\cdot)}$ in its own space (i.e., the source space or the target space), into the latent space, i.e., the project-invariant feature space $f \in \mathbb{R}^{D_f}$, to derive the representation $G_f(x_k^{(\cdot)})$ of $x_k^{(\cdot)}$ in the project-invariant feature space. The decoder G_r is used to decode the representation $G_f(x_k^{(\cdot)})$ in the project-invariant feature space to approximate the original artifact $x_k^{(\cdot)}$ as $\hat{x}_k^{(\cdot)}$. The denotation $(\cdot)$ means that $x_k^{(\cdot)}$ can be from either the source project S or the target project E.

The global discriminator G_d is adopted to determine whether the input artifact $x_k^{(\cdot)}$ comes the source project or the target project, without considering if the input artifact $x_k^{(\cdot)}$ is defective or nondefective [9]. Local discriminators G_d^0 and G_d^1 are used to determine whether the input artifact $x_k^{(\cdot)}$ comes from the source project or the target project while considering whether the input artifact $x_k^{(\cdot)}$ is defective or nondefective [26]. The global and local discriminators are referred to as domain discriminators, which are proposed to collaboratively align the global and local distributions between the source and target projects in the project-invariant feature space $f \in \mathbb{R}^{D_f}$ by adversarial learning [39].

Finally, the defect predictor G_y is trained by learning the predictive relationship between the representation $G_f(x_i^{(S)})$ of the source artifact $x_i^{(S)}$ in the project-invariant feature space $f \in \mathbb{R}^{D_f}$ and its label (i.e., defective or nondefective). The parameters of the four components in the proposed Adan approach were learned simultaneously using backpropagation with an integrated loss function.

4.2 Feature autoencoder

In deep learning, an autoencoder is usually used to generate a new feature space by utilizing the nonlinear transformations of the input features so that the new feature space can more effectively capture the underlying patterns present in the data [2]. By mapping the data into the new feature space, nonlinear methods can find patterns that are not apparent in the original feature space, even in an unusual case where the number of features significantly exceeds the number of samples. A feature autoencoder can also capture complex nonlinear relationships between input features; it does not need domain knowledge or prior assumptions about data and often leads to better predictive performance [37].

As shown in Fig. 2, the feature autoencoder comprises feature encoder G_f and feature decoder G_r with a symmetric structure. The feature extractor G_f consists of two dense

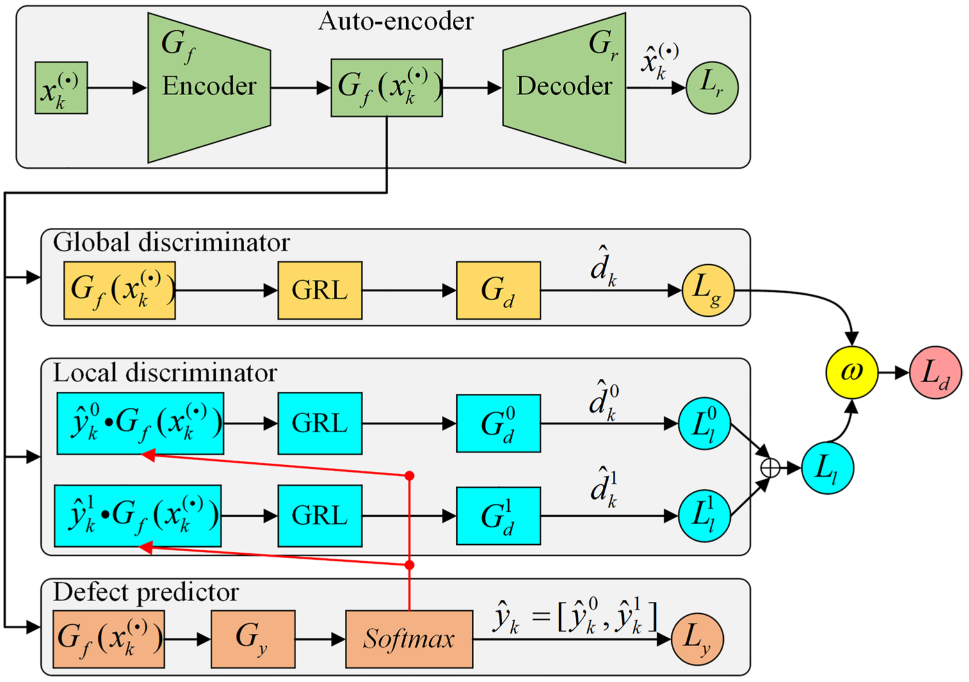

Fig. 1 Overall architecture of the proposed Adan approach

layers as H_f^1 and H_f^2, and the feature decoder consists of two dense layers as H_r^1 and H_r^2. The numbers of neurons in H_f^1 and H_r^2 are set as the same as $2 \cdot |x_k^{(\cdot)}|$ and the number of neurons in H_f^2 and H_r^1 are set as the same as $|x_k^{(\cdot)}|$ by trial and error where $|x_k^{(\cdot)}|$ is the length of the vector of the input sample $x_k^{(\cdot)}$. The input artifact $x_k^{(\cdot)} \in \mathbb{R}^{|x_k^{(\cdot)}|}$ is processed by the layers in the feature encoder whose output is the representation of $x_k^{(\cdot)}$ as $G_f(x_k^{(\cdot)}) \in \mathbb{R}^{D^f}$ (i.e., $|D^f| = |G_f(x_k^{(\cdot)})|$) in the project-invariant feature space. In the feature decoder, the extracted representation $G_f(x_k^{(\cdot)})$ is fed into the feature decoder G_r to reconstruct the input artifact $x_k^{(\cdot)}$ as $\hat{x}_k^{(\cdot)}$. Thus, we have the following Eqs. (1) and (2), where W_f^1 (W_r^1) and W_f^2 (W_r^2) are the weight matrices and b_f^1 (b_r^1) and b_f^2 (b_r^2) are the bias vectors of the two dense layers H_f^1 (H_r^1) and H_f^2 (H_r^2) in the feature encoder (decoder):

$$G_f(x_k^{(\cdot)}) = W_f^2 \cdot (W_f^1 \cdot x_k^{(\cdot)} + b_f^1) + b_f^1 \tag{1}$$

$$\hat{x}_k^{(\cdot)} = W_r^2 \cdot (W_r^1 \cdot G_f(x_k^{(\cdot)}) + b_r^1) + b_r^2 \tag{2}$$

Here, the reconstruction loss function G_r is shown in Eq. (3). The sum $(n_S + n_E)$ is the number of input artifacts in a batch where n_S is the number of artifacts drawn from the source project and n_E is the number of artifacts drawn from the target project. The feature autoencoder minimizes the L2 distance between the input artifact $x_k^{(\cdot)}$ and its reconstructed representation $\hat{x}_k^{(\cdot)}$ in the training process.

$$L_r = \frac{1}{n_S + n_E} \sum_{k=1}^{n_S+n_E} \| x_k^{(\cdot)} - \hat{x}_k^{(\cdot)} \|_2 \tag{3}$$

4.3 Global discriminator

The global discriminator G_d is used to align the global distributions between source and target software projects [9]. In training the proposed Adan approach, a batch of $(n_S + n_E)$ artifacts is composed of n_S artifacts drawn from the source project D^S and n_E artifacts drawn from the target project D^E. It can be seen from Fig. 1 that the input of the global discriminator G_d is the representation $G_f(x_k^{(\cdot)})$ of the input artifact $x_k^{(\cdot)}$ in the project-invariant feature space $f \in \mathbb{R}^{D_f}$. Note that the artifact $x_k^{(\cdot)}$ can be drawn from either the source project (i.e.,$x_k^{(\cdot)} = x_i^{(S)}$) or the target project (i.e., $x_k^{(\cdot)} = x_i^{(E)}$).

The global discriminator is devised as a binary neural network classifier used to discriminate whether the artifact $x_k^{(\cdot)}$ comes from source project D^S or target project D^E. The output $\hat{d}_k \in [0,1]$ of the global discriminator G_d is the predicted probability that the input artifact $x_k^{(\cdot)}$ is drawn from the target project (i.e.$G_d(G_f(x_k^{(\cdot)})) = \hat{d}_k$). For simplicity, the neural networks of the global discriminator G_d, the local discriminators G_d^0 and G_d^1, and the defect predictor G_y are made to share the same structure, as shown in Fig. 3.

For simplicity, the hidden layers in the structure are dense layers H^1 and H^2 where the number of neurons of H^1 is set as $2 \cdot |D_f|$ and the number of neurons of H^2 is set equal to $|D_f|$ by trial and error. Batch normalization and a dropout strategy are adopted in layer H^1 to accelerate model convergence

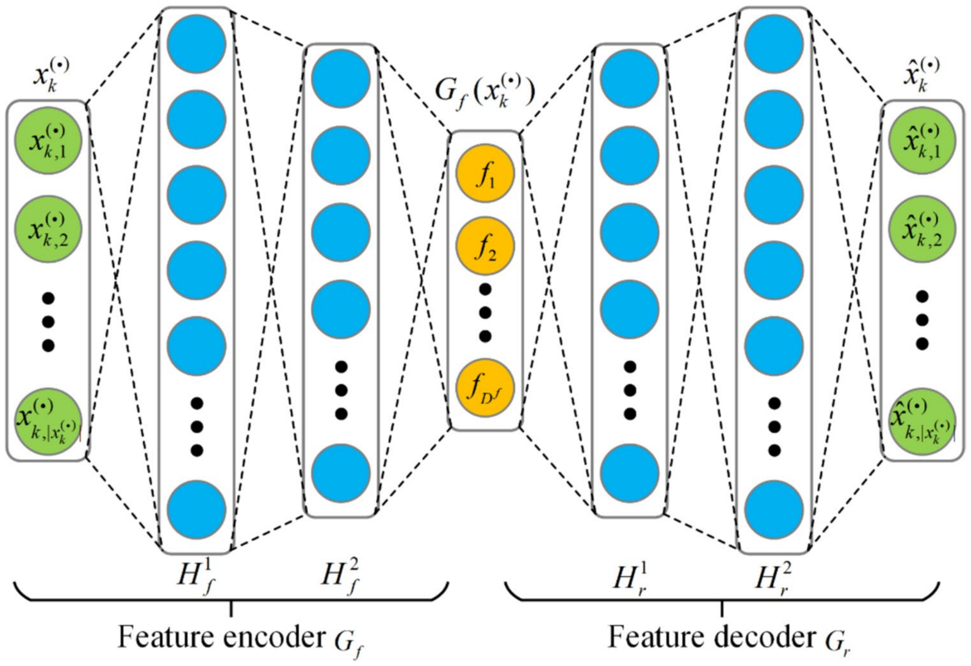

Fig. 2 The feature autoencoder in the proposed Adan approach

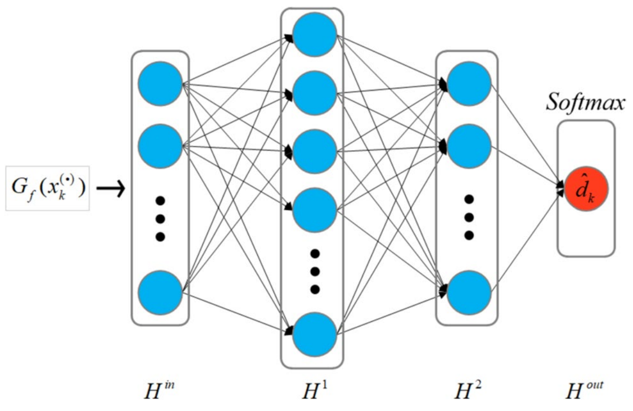

Fig. 3 The shared model structure of the global discriminator G_d, the local discriminators G_d^0 and G_d^1, and the defect predictor G_y

and avoid overfitting [10, 30]. The computation process of the output $\hat{d}_k$ can be described in Eq. (4) where W^{in},W^1,W^2 and b^{in},b^1,b^2 are the weight matrices of the layers H^{in},H^1 and H^2 correspondingly:

$$\hat{d}_k = W^2 \cdot (W^1 \cdot (W^{in} \cdot G_f(x_k^{(\cdot)}) + b^{in}) + b^1) + b^2 \tag{4}$$

The objective of the global discriminator is to minimize the cross-entropy loss, as shown in Eq. (5), where $d_k \in \{0,1\}$ denotes the project label $x_k^{(\cdot)}$,$d_k = 0$ denotes that the input artifact $x_k^{(\cdot)}$ is drawn from the source project (i.e., $x_k^{(\cdot)} \in D^S$) and $d_k = 1$ denotes that the input sample $x_k^{(\cdot)}$ is drawn from the target project (i.e., $x_k^{(\cdot)} \in D^E$) as shown in Eq. (6):

$$L_g = \frac{1}{n_S + n_E} \sum_{k=1}^{n_S+n_E} -d_k \log\hat{d}_k - (1 - d_k)\log(1 - \hat{d}_k) \tag{5}$$

$$d_k = \begin{cases} 0, \ if \ x_k^{(\cdot)} \in D^S \\ 1, \ if \ x_k^{(\cdot)} \in D^E \end{cases} \tag{6}$$

4.4 Local discriminator

Considering that the distributions of defective and nondefective data might be inherently heterogeneous, the local discriminators (i.e.,G_d^0 and G_d^1) are developed to align the local distributions between the source and target software projects. This is different from the global discriminator G_d without considering that the input sample $x_k^{(\cdot)}$ is defective or nondefective. It can be seen from Fig. 1 that the input artifact of the two local discriminators is the probability-weighted representations $\hat{y}_k^0 G_f(x_k^{(\cdot)})$ and $\hat{y}_k^1 G_f(x_k^{(\cdot)})$ where $\hat{y}_k^0$ is the predicted probability that the input artifact $x_k^{(\cdot)}$ is nondefective, $\hat{y}_k^1$ is the predicted probability that the input artifact $x_k^{(\cdot)}$ is defective, i.e., $\hat{y}_k^0$+$\hat{y}_k^1$=1.

The objective of a local discriminator is the same as the global discriminator in discriminating whether the input artifact $x_k^{(\cdot)}$ is drawn from the source or target project. The output of G_d^0 is the predicted probability $\hat{d}_k^0$ denoting that the input artifact $x_k^{(\cdot)}$ is drawn from the source project (i.e.,$\hat{d}_k^0 = G_d^0(\hat{y}_k^0 G_f(x_k^{(\cdot)}))$) and, the output of G_d^1 is the predicted probability $\hat{d}_k^1$ denoting that the input artifact $x_k^{(\cdot)}$ is drawn from the target project (i.e.$\hat{d}_k^1$=$G_d^1(\hat{y}_k^1 G_f(x_k^{(\cdot)}))$). The training losses of G_d^0 and G_d^1 are computed using Eqs. (7) and (8) respectively, where $d_k \in \{0,1\}$ is the project label of the input sample $x_k^{(\cdot)}$. The total loss of G_d^0 and G_d^1 is the sum of L_l^0 and L_l^1 in Eq. (9):

$$L_l^0 = \frac{1}{n_S + n_E} \sum_{k=1}^{n_S+n_E} -d_k \log\hat{d}_k^0 - (1 - d_k)\log(1 - \hat{d}_k^0) \tag{7}$$

$$L_l^1 = \frac{1}{n_S + n_E} \sum_{i=1}^{n_S+n_E} -d_k \log\hat{d}_k^1 - (1 - d_k)\log(1 - \hat{d}_k^1) \tag{8}$$

$$L_l = L_l^0 + L_l^1 \tag{9}$$

Thus, it can be seen that the input artifacts are mapped into a project-invariant feature space f by the feature encoder G_f with two objectives. The first objective is to use G_f to capture the underlying patterns present in both the source and target artifacts by utilizing nonlinear transformations of the input features. To improve the quality of the representations $G_f(x_i^{(S)})$ and $G_f(x_j^{(E)})$ in the project-invariant feature space D_f, the feature decoder G_r is used to reconstruct the input artifacts $x_i^{(S)}$ and $x_j^{(E)}$ from the representations to ensure that the representations $G_f(x_i^{(S)})$ and $G_f(x_j^{(E)})$ can be reconstructed as the original artifacts $x_i^{(S)}$ and $x_j^{(E)}$ as approximate as possible. The second objective is to use G_f to confuse the global discriminator G_d and the two local discriminators (i.e.,G_d^0 and G_d^1) to prevent them from discriminating that the input artifact $x_k^{(\cdot)}$ comes from the source or target project. That is, to ensure that the global alignment between the source and target projects as $P(G_f(x_i^{(S)}))_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)}))_{j=1}^{N^{(E)}}$ and the local alignment between the source and target projects as $P(G_f(x_i^{(S)})|y = 0)_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)})|y = 0)_{j=1}^{N^{(E)}}$ and $P(G_f(x_i^{(S)})|y = 1)_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)})|y = 1)_{j=1}^{N^{(E)}}$, as described in Sect. 3.

Given a pair of input artifacts $x_i^{(S)}$ and $x_j^{(E)}$, the feature encoder G_f maps them into the project-invariant feature space D_f as the representations $G_f(x_i^{(S)})$ and $G_f(x_j^{(E)})$, respectively. If the global discriminator and local discriminator cannot discriminate between which one of the source projects, or the target project $G_f(x_i^{(S)})$ and $G_f(x_j^{(E)})$ come from, we can regard that the extracted feature space D_f is project-invariant.

4.5 Defect predictor

The defect predictor G_y is a neural network binary classifier to predict the label of the input artifact $x_k^{(\cdot)}$ as defective or nondefective. The defect predictor G_y is trained using the labeled artifacts in the source project D^S to learn the predictive relationship between the input artifact $x_i^{(S)}$ and its corresponding label $y_i^{(S)}$. Further, the trained defect predictor G_y is used predict the label of the unlabeled artifact $x_j^{(E)}$ in target project D^E. In the training phase, the input of the defect predictor G_y is the representation $G_f(x_i^{(S)})$ of the source artifact $x_i^{(S)}$ in the project-invariant space f. The output of the defect predictor G_y is a vector $\widehat{y}_i = [\widehat{y}_i^0, \widehat{y}_i^1]$, where $\widehat{y}_i^0$ denotes the predicted probability of the source artifact $x_i^{(S)}$ is nondefective and $\widehat{y}_i^1$ denotes that the predicted probability of the source artifact $x_i^{(S)}$ is defective (i.e., $\widehat{y}_i^0 + \widehat{y}_i^1 = 1$ and $0 \le y_i^0, y_i^1 \le 1$).

We observed that less than 12% of artifacts in the investigated software projects were defective in the collected data for experiments (see Sect. 5.1), and this is consistent with existing studies reporting that the defects are extremely imbalanced distributed in a software project [32]. Traditional binary classification loss functions, such as binary cross-entropy, often prioritize the class of majority and neglect the class of minority, leading to suboptimal performance for software defect prediction. For this reason, we adopt the focal loss [17] to tackle the class imbalance problem in training the defect predictor G_y as shown in Eqs. (10), (11), and (12). Here, $\widehat{y}_i^1$ denotes the defective probability of $x_i^{(S)}$ and $\widehat{y}_i^0$ denotes the nondefective probability of $x_i^{(S)}$. The parameter $\alpha \in (0,1)$ is a balance factor to trade off the nondefective and defective classes with a larger weight for the defective class than that for the nondefective class. The parameter $\gamma \in [0,5]$ is a focusing factor to reshape the loss function to downweight the easy samples ($\widehat{p}_i > 0.5$) to make the defect predictor G_y focus more on hard samples ($\widehat{p}_i < 0.5$) [17]. The focusing parameter γ reduces the loss contribution from easy samples where the defect predictor G_y is already quite confident in predicting its labels. Meanwhile, it increases the loss contribution of hard examples that are often misclassified by the defect predictor G_y:

$$L_y = \frac{1}{n_S} \sum_{i=1}^{n_S} -\alpha_i (1 - \widehat{p}_i)^{\gamma} \log(\widehat{p}_i) \tag{10}$$

$$\widehat{p}_i = \begin{cases} \widehat{y}_i^0, & if \ x_i^{(S)} \ is\, non-defective \\ \widehat{y}_i^1, & if \ x_i^{(S)} \ is\, defective \end{cases} \tag{11}$$

$$\alpha_i = \begin{cases} 1-\alpha, & if \ x_i^{(S)} \ is\, non-defective \\ \alpha, & if \ x_i^{(S)} \ is\, defective \end{cases} \tag{12}$$

In model implementation, the global discriminator G_d, the local discriminators G_d^0 and G_d^1, and the defect predictor G_y share the same model structure as binary classification neural networks with different parameters and objectives. The input of the proposed Adan approach is the representation $G_f(x_k^{(\cdot)})$ in the project-invariant feature space produced by the feature encoder G_f as shown in Fig. 2. The output of the proposed Adan approach is a binary vector as $[\widehat{y}_k^0, \widehat{y}_k^1]$ produced by a softmax function, where $\widehat{y}_k^0$ and $\widehat{y}_k^1$ are the predicted probabilities of $x_k^{(\cdot)}$ belonging to the nondefective class and defective class, respectively. In the training process, if the source artifact $x_i^{(S)}$ is nondefective, we set $\widehat{p}_i$ as $\widehat{y}_k^0$. Otherwise, if the source artifact $x_i^{(S)}$ is defective, we set $\widehat{p}_i$ as $\widehat{y}_k^1$.

For the global discriminator G_d, we extend its output $\widehat{d}_k$ as $[\widehat{d}_k^0, \widehat{d}_k^1]$ by setting $\widehat{d}_k^0$=1-$\widehat{d}_k^1$ and $\widehat{d}_k^1$=$\widehat{d}_k$ where $\widehat{d}_k^1$ indicates the probability that $x_k^{(\cdot)}$ comes from the source project and $\widehat{d}_k^0$ indicates the probability that $x_k^{(\cdot)}$ comes from the target project. For the local discriminators as G_d^0 and G_d^1, its output $[\widehat{d}_k^0, \widehat{d}_k^1]$ has the same meaning as that of the global discriminator. For the defect predictor G_y, its output $[\widehat{y}_k^0, \widehat{y}_k^1]$ indicates the probability that the input artifact $x_k^{(\cdot)}$ is nondefective or defective.

4.6 Loss function

The proposed Adan approach consists of four components: an autoencoder, a global discriminator, a local discriminator, and a defect predictor. Thus, the overall loss function is the weighted summarization of the four losses, as shown in Eqs. (13) and (14). The parameters β and λ are parameters used to trade off the prediction loss L_y, feature reconstruction loss L_r and domain discriminator loss L_d. Note that the domain discriminator loss L_d is a weighted sum of the global discriminator loss L_g and the local discriminator loss L_l. By trial and error, we set β=0.5 and λ=0.5 for the proposed Adan approach in the experiments in Sect. 6:

$$L = L_y + \beta L_r - \lambda L_d \tag{13}$$

$$L_d = (1-\omega)L_g + \omega L_l \tag{14}$$

The relative importance of the global and local distributions is estimated by the dynamic adversarial factor $\omega \in [0,1]$. We follow Yu et al.'s research (2019) to estimate the global distribution distance between source and target projects using Eq. (15) and estimate the local distribution distance between them using Eq. (16):

$$dis_g(D^{(S)}, D^{(E)}) = 2(1 - 2L_g) \tag{15}$$

$$dis_l(D^S, D^E) = [dis_l^0(D^{(S)}, D^{(E)}) + dis_l^1(D^{(S)}, D^{(E)})]/2 \\ = [2(1 - 2L_l^0) + 2(1 - 2L_l^1)]/2 = 2(1 - 2L_l) \quad (16)$$

In Eq. (17), the value of ω is dynamically adjusted during the training process to tradeoff the relative importance between global and local distributions. The basic idea of the dynamic loss function in Eq. (13) is quite straightforward. That is, when the global discriminator loss L_g is small, which means that the representations of source artifacts and target artifacts cannot confuse the global discriminator adequately, $dis_g(D^S, D^E)$ would be large, resulting in a large value for ω. Then, the weight 1-ω of L_g in the domain discriminator loss would be small, making it relatively less important than the local discriminator L_l since the representations of source and target artifacts can be well discriminated by the global discriminator. In this way, the global discriminator loss L_g would become large to align the global distribution between the source and target projects, that is, to ensure $P(G_f(x_i^{(S)}))_{i=1}^{N^{(S)}} \approx P(G_f(x_j^{(E)}))_{j=1}^{N^{(E)}}$:

$$\omega := \frac{dis_g(D^S, D^E)}{dis_g(D^S, D^E) + dis_l(D^S, D^E)} \quad (17)$$

4.7 Training with adversarial learning

Adversarial learning is adopted to train the proposed Adan approach, and the two players in the game are the feature encoder (G_f) and the domain discriminators (G_d,G_d^0, and G_d^1). On the one hand, the feature encoder G_f aims to extract project-invariant feature space f from the source and target artifacts while aligning the data distributions of the source and target artifacts to prevent the discriminators from identifying the input artifact $x_k^{(\cdot)}$ coming from the source project D^S or the target project D^E. On the other hand, the global and local discriminators' objectives are to identify the input artifact $x_k^{(\cdot)}$ coming from the source project or the target project. For this reason, in adversarial learning, as shown in Fig. 2, the parameters of the feature encoder (G_f) as θ_f= {W_f^1,W_f^2,b_f^1 ,b_f^2} are learned by maximizing the domain discrimination loss (L_d).

In Fig. 2, the parameters of the feature decoder (G_r) as θ_r= {W_r^1,W_r^2,b_r^1,b_r^2} in Fig. 2 are learned by minimizing the feature reconstruction loss (L_r). Meanwhile, the parameters (θ_d, θ_d^0, θ_d^1) of the domain discriminators (G_d,G_d^0, and G_d^1) are learned by minimizing the local discrimination loss (L_d). The parameters of the defect predictor θ_y are to minimize the defect prediction loss (L_y). Note that the parameters of the domain discriminators (G_d,G_d^0, and G_d^1) and the defect predictor (G_y) are shown in Fig. 3, sharing the same shapes in their weight matrices and bias vectors.

The training process of the proposed Adan approach includes forward propagation of input artifacts for loss computation and backpropagation of the training loss for parameter updating. One batch of training data consists of both the labeled artifacts drawn from both the source project and the unlabeled artifacts drawn from the target project, i.e., $\{x_1^{(S)}, ..., x_{n_S}^{(S)}, x_1^{(E)}, ..., x_{n_E}^{(E)}\}$. Here, n_S and n_E denote the number of artifacts drawn from the source project and the target project, respectively. The stochastic gradient descent and early stopping strategy were adopted to optimize the training process of the proposed Adan approach [10].

In the forward propagation, for the feature encoder G_f and the feature decoder G_r, the feature reconstruction loss L_r is computed using Eq. (3). For the defect predictor G_y, the prediction loss L_y on labeled data $x_i^{(S)}$ is computed using Eq. (10). For the global discriminator G_d, the global discrimination loss L_g is computed according to Eq. (5). For the local discriminators G_d^0 and G_d^1, the local discrimination loss L_l is computed according to Eq. (9). The relative importance of global distribution and local distributions is aligned by dynamical adjustment in model training according to Eq. (17). Therefore, the total loss of discriminators L_d is the dynamically weighted summarization of L_g and L_l. The overall loss L is the weighted summarization of L_r, L_y and L_d computed by Eq. (13).

In the backpropagation of training loss, the parameters that need to be updated include θ_f of feature encoder G_f, θ_r of feature decoder G_r, θ_y of defect predictor G_y, θ_d of global discriminator G_d, (θ_d^0, θ_d^1) of local discriminators (G_d^0, G_d^1). The overall loss of the proposed Adan approach consists of the reconstruction loss L_r, the defect-prediction loss L_y and the discrimination loss L_d. We use $\Theta = \{\theta_f, \theta_r, \theta_y, \theta_d, \theta_d^0, \theta_d^1\}$ to denote the set of parameters needed to be learned in training the proposed Adan approach. The gradients of the parameters in Θ concerning the overall loss L can be mathematically derived in Eq. (18):

$$\nabla L_\theta = \frac{\partial L}{\partial \theta} = \frac{\partial L_y}{\partial \theta} + \beta \frac{\partial L_r}{\partial \theta} - \lambda \frac{\partial L_d}{\partial \theta} \quad (18)$$

The objective of the feature extractor G_f consists of three objectives: to improve the defect-prediction performance (minimize L_y), improve the representativeness of the extracted feature space f(minimize L_r), and to confuse the discriminators G_d,G_d^0, and G_d^1 (maximize L_d). Thus, the parameter θ_f is updated by gradient descent of L_y and L_r, and gradient ascent of L_d. In the proposed Adan approach, the gradient ascent of L_d is achieved by the utilization of a gradient reversal layer (GRL) (Ganin et al. 2015) positioned between the feature encoder G_f and discriminators (G_d,G_d^0,G_d^1). In backpropagation, the GRL negates the gradients, thus encouraging the feature extractor G_f to

Table 1 Statistics of the collected data

Project	Release	# of methods	# of Defective Methods	Defective Proportion
Kafka	2.0.0	10,504	1231	11.72%
Kylin	2.0.0	7077	778	10.99%
Ant	1.7.0	10,868	305	2.81%
Tomcat	8.5.0	19,487	557	2.86%

learn project-invariant features. In forward propagation, the GRL behaves as an identity function, transmitting the input sample $x_k^{(\cdot)}$ unchanged to the subsequent neural network. Based on the backpropagation analysis, it is found that the parameter optimization on Θ will produce a saddle point with respect to the overall loss as Eqs. (19) and (20) with convergence.

$$(\widehat{\theta}_f, \widehat{\theta}_y, \widehat{\theta}_r) = \underset{\theta_f,\theta_y,\theta_r}{\operatorname{argmin}} L(\theta_f, \theta_y, \theta_r, \widehat{\theta}_d, \widehat{\theta}_d^0, \widehat{\theta}_d^1) \tag{19}$$

$$(\widehat{\theta}_d, \widehat{\theta}_d^0, \widehat{\theta}_d^1) = \underset{\theta_d,\theta_d^0,\theta_d^1}{\operatorname{argmax}} L(\widehat{\theta}_f, \widehat{\theta}_y, \widehat{\theta}_r, \theta_d, \theta_d^0, \theta_d^1) \tag{20}$$

5 Experiments

5.1 Dataset

Following Zhang et al. [44], method-level defect prediction was conducted in this study. The source code, Git commit history, and bug reports are collected from four open-source projects Kafka,[1] Kylin,[2] Ant[3] and Tomcat.[4] Fifty-eight features for each method were extracted from the collected source code and Git commit history as code metrics, developer metrics, and development metrics using the method proposed by Zhang et al. [44]. The statistical information of the collected data is shown in Table 1. It can be seen that the defective methods were extremely imbalanced (i.e., fewer than 12% of the methods were defective in the collected data). To address the problem of class imbalance, the focal loss function was adopted to weigh the losses of the two classes, as specified in Sect. 4.5.

[1] GitHub-apache/kafka: Mirror of Apache Kalfka: https://github.com/apache/kafka.

[2] GitHub-apache/kylin: Apache Kylin: https://github.com/apache/kylin.

[3] GitHub-apache/ant: Apache Ant is a Java-based build tool: https://github.com/apache/ant.

[4] GitHub-apache/tomcat: Apache Tomcat: https://github.com/apache/tomcat.

Twelve pairs of tasks on cross-project defect prediction are devised to investigate the effectiveness of the proposed Adan approach. For instance, the Kafka→Kylin task is that Kafka is considered a source project with labeled data, and Kylin is a target project with unlabeled data. The proposed Adan approach is adopted on the Kafka→Kylin task to learn project-invariant features for both Kafka and Kylin projects. Meanwhile, a defect predictor G_y is trained using the data from Kafka, and its performance is investigated using the data from Kylin.

Each method is represented using 58 features in the original feature space as $x_k^{(\cdot)} = \{x_{k,C}^{(\cdot)}, x_{k,D}^{(\cdot)}, x_{k,P}^{(\cdot)}\}$ where these features can be categorized into three groups: 37 code features as $x_{k,C}^{(\cdot)} = \{x_{k,C}^{(\cdot),1}, ..., x_{k,C}^{(\cdot),37}\}$, 15 developer features as $x_{k,D}^{(\cdot)} = \{x_{k,D}^{(\cdot),1}, ..., x_{k,D}^{(\cdot),15}\}$, and 6 development process features as $x_{k,D}^{(\cdot)} = \{x_{k,P}^{(\cdot),1}, ..., x_{k,P}^{(\cdot),6}\}$. The source code metrics were derived from Understand.[5] Table 2 shows the code-level features that provide insights into the architecture and scale of the software system by quantifying its code size, code dependencies, and internal structure [6, 22]. Table 3 shows the developer-level features that reflect developers' personal characteristics and behaviors, including developers' coding experience, code ownership, coding style, programming ability, and coding attention [3]. Table 4 shows the development process-level features used to quantify developers' collaborations and the evolution patterns of source code during software development [11].

5.2 Experimental setting

For each pair of cross-project defect-prediction tasks, three subsets—training data, cross-validation data, and test data—were randomly partitioned from the whole dataset. By trial and error, 80% of the whole dataset was used for training, 10% of the whole dataset was used in cross-validation, and the remaining 10% of the whole dataset was used in the model test. The entire dataset was randomly partitioned 10 times to average the performances of the 10 repetitions.

For the feature extractor, as shown in Fig. 2, its parameters are the number of input artifacts as $(n_S + n_E)$ in each batch for model training, where n_S is the number of artifacts drawn from the source project and n_E is the number of artifacts drawn from the target project. In the experiments, both n_S and n_E are set as 128 for all 12 pairs of tasks on cross-project defect prediction. As described in Sect. 4.2, the numbers of neurons in H_f^1 and H_r^2 are set as $2 \cdot |x_k^{(\cdot)}|$ and the number of neurons in H_f^2 and H_r^1 are set as $|x_k^{(\cdot)}|$ by trial and error where $|x_k^{(\cdot)}|$ (i.e., 58) is the length of the vector of the input artifact $x_k^{(\cdot)}$.

[5] Understand Software Metrics. Online: https://documentation.scitools.com/pdf/metricsdoc.pdf. Accessed January 11, 2024.

Table 2 The 37 code-level features in the original feature space

Level	Structure/Scale	Description	Notation
Method	Structure	# of input parameters	$x_{k,C}^{(\cdot),1}$
		# of output parameters	$x_{k,C}^{(\cdot),2}$
		# of paths	$x_{k,C}^{(\cdot),3}$
		Log # of paths	$x_{k,C}^{(\cdot),4}$
		Cyclomatic complexity	$x_{k,C}^{(\cdot),5}$
		Modified Cyclomatic complexity	$x_{k,C}^{(\cdot),6}$
		Strict Cyclomatic complexity	$x_{k,C}^{(\cdot),7}$
		Essential complexity	$x_{k,C}^{(\cdot),8}$
		Knots	$x_{k,C}^{(\cdot),9}$
		Max essential knots	$x_{k,C}^{(\cdot),10}$
		Max nesting	$x_{k,C}^{(\cdot),11}$
		Min essential knots	$x_{k,C}^{(\cdot),12}$
		Sum Cyclomatic complexity	$x_{k,C}^{(\cdot),13}$
		Sum modified Cyclomatic complexity	$x_{k,C}^{(\cdot),14}$
		Sum strict Cyclomatic complexity	$x_{k,C}^{(\cdot),15}$
		Sum essential complexity	$x_{k,C}^{(\cdot),16}$
	Scale	# of lines	$x_{k,C}^{(\cdot),17}$
		# of blank lines	$x_{k,C}^{(\cdot),18}$
		# of source code lines	$x_{k,C}^{(\cdot),19}$
		# of code declaration lines	$x_{k,C}^{(\cdot),20}$
		# of code execution lines	$x_{k,C}^{(\cdot),21}$
		# of comment lines	$x_{k,C}^{(\cdot),22}$
		# of semicolons	$x_{k,C}^{(\cdot),23}$
		# of statements	$x_{k,C}^{(\cdot),24}$
		# of statement declarations	$x_{k,C}^{(\cdot),25}$
		# of statement execution	$x_{k,C}^{(\cdot),26}$
		Ratio of comment lines to code lines	$x_{k,C}^{(\cdot),27}$
Class	Structure	Conceptual coupling between objects	$x_{k,C}^{(\cdot),28}$
		Depth of inheritance tree	$x_{k,C}^{(\cdot),29}$
		# of Children	$x_{k,C}^{(\cdot),30}$
		Lack of cohesion in methods	$x_{k,C}^{(\cdot),31}$
		Coupling between objects	$x_{k,C}^{(\cdot),32}$
		Lack of modified cohesion in methods	$x_{k,C}^{(\cdot),33}$
		Coupling between modified objects	$x_{k,C}^{(\cdot),34}$
		Conceptual declared instance variable	$x_{k,C}^{(\cdot),35}$
		Conceptual declared method	$x_{k,C}^{(\cdot),36}$
		Conceptual declared method access	$x_{k,C}^{(\cdot),37}$

For the global discriminator G_d, the local discriminators G_d^0 and G_d^1, and the defect predictor G_y shown in Fig. 3, its parameters are H^{in},H^1 and H^2. By trial and error, the size of H^{in} is set equivalent to $|D^f|$ and the sizes of hidden dense layers H^1 and H^2 are set as $2 \cdot |D_f|$ and $|D_f|$, respectively. Thus, for the feature encoder (see Fig. 2) and the shared model structure (see Fig. 3), there is only one parameter needing to be tuned as the number of project-invariant features $|D^f|$. For the defect predictor, it has two other parameters, the balance factor α and the focusing factor γ as shown in Eq. (7), to be tuned in training the proposed Adan approach.

5.3 Baseline methods

The proposed Adan approach is compared with four state-of-the-art techniques in cross-project defect prediction, including data selection and sampling-based domain programming predictor (DSSDPP) [16], joint feature representation with double marginalized denoising autoencoder (DMDAJFR) [43], transfer Naïve Bayes (TNB) [21], TCA + [23], and CCA + [14].

DSSDPP is a parameter-free method for cross-project defect prediction. It first constructs a similar subset from a source project based on the distribution similarity measure between the data points in the source and target projects using maximum mean discrepancy. Second, it applies the metric value permutation-based oversampling to generate new artifacts from the minority class (defective artifacts) to balance the class distribution. Finally, a domain programming predictor is trained using the balanced source dataset for defect prediction.

The DMDAJFR comprises two types of autoencoders, domain adaptation global marginalized denoising autoencoder (DA-GMDA) and domain adaptation local subsets marginalized denoising autoencoder (DA-LMDA), to learn both global and local feature representations. It employs a progressive distribution-matching strategy that reassigns pseudolabels after each stack-layer learning, allowing for a more refined alignment of the feature space between source and target projects over iterations. By trial and error, the number of layers in the autoencoder is set as 8, the noise probability is 0.8, and the regularization parameter is 0.5 in comparative experiments.

The TNB method trains transfer Naïve Bayes for cross-project defect prediction. It first calculates the similarity of each sample in the source project to the sample in the target project. Then, large weights are assigned to the source samples, which are similar to the target samples. A weighted Naïve Bayes classifier is built on source-project data using these weights to integrate the target-project data distribution knowledge into cross-project defect prediction. Note that the

Table 3 Developer-level features in the original feature space

Category	Description	Notation
Experience	# of commits in the project of the developer	$x_{k,D}^{(\cdot),1}$
	# of activities in recent three months of the developer	$x_{k,D}^{(\cdot),2}$
	# of activities in the subsystem of the developer	$x_{k,D}^{(\cdot),3}$
Ownership	# of major commits of the developer	$x_{k,D}^{(\cdot),4}$
	# of minor commits of the developer	$x_{k,D}^{(\cdot),5}$
	# of major code snippets of the developer	$x_{k,D}^{(\cdot),6}$
	# of minor code snippets of the developer	$x_{k,D}^{(\cdot),7}$
	# of max commits of the developer	$x_{k,D}^{(\cdot),8}$
	# of max code snippets of the developer	$x_{k,D}^{(\cdot),9}$
Style	Working dispersion of the developer	$x_{k,D}^{(\cdot),10}$
	Ratio of # of comments to # of code lines of the developer	$x_{k,D}^{(\cdot),11}$
Ability	Average severity of the bugs resolved by the developer	$x_{k,D}^{(\cdot),12}$
	Average depth of inheritance tree of the objects touched by the developer	$x_{k,D}^{(\cdot),13}$
	Average number of children of the objects touched by the developer	$x_{k,D}^{(\cdot),14}$
Attention	Average on reciprocal of # of committed files in one commit	$x_{k,D}^{(\cdot),15}$

TNB method is a parameter-free approach, which means that it requires no parameter setting.

The TCA + method improves classical transfer component analysis by integrating automatic data normalization. It determines the optimal normalization technique for a given source-target project pair by exploring their data distributions using a set of predefined statistical tests and rules. It then applies TCA to find a latent feature space that reduces the difference between the normalized source and target data distributions. A software defect-prediction model (classifier) is trained on the labeled source-project data in the latent space and applied to the transformed unlabeled target project data. The parameter setting of the TCA + method includes the number of dimensions of the latent feature space and the classifier used. Following Nam et al. [23], the number of dimension of the latent feature space was set as 45, and the classifier used was Adaptive Boosting (Adaboost) in the experiment.

The CCA + method is a heterogeneous cross-project defect-prediction approach. It first represents the heterogeneous data in a unified metric representation (UMR) by combining common metrics, project-specific metrics, and zeros. Second, it uses CCA to align the data distributions of the source and target projects. Third, it learns a software defect-prediction model on the UMR of the source-project data and applies it to the target project. The parameter of the CCA + method includes the dimension of the shared feature space and the classifier used. Following Jing et al. [14], the dimension of the latent feature space is set as 40 and the classifier as a random forest in the experiment.

Table 4 Project-level features in the original feature space

Description	Notation
# of commits of the project	$x_{k,P}^{(\cdot),1}$
# of committers of the project	$x_{k,P}^{(\cdot),2}$
# of commits in the latest release of the project	$x_{k,P}^{(\cdot),3}$
# of committers in the latest release of the project	$x_{k,P}^{(\cdot),4}$
# of lines removed from the latest release of the project	$x_{k,P}^{(\cdot),5}$
# of lines added to the latest release of the project	$x_{k,P}^{(\cdot),6}$

5.4 Evaluation measures

For unbalanced defective and nondefective methods in the experimental dataset, the area under the receiver operating characteristic curve (AUC) and F-measure were adopted to evaluate the software defect-prediction performance of the proposed Adan approach [16, 40]. The AUC is a comprehensive measure commonly used for imbalanced classification and is widely accepted as indicating the classifier's ability to distinguish each class in a balanced manner [8]. The larger the AUC value, the better the classifier. The calculation of AUC is shown in Eqs. (21), where M is the number of defective artifacts, N is the number of nondefective artifacts and $rank_i$ is the rank of the i-th defective artifact predicted by the classifier. The F-measure provides a balanced and informative assessment of classification model performance by harmonizing precision and recall. The larger the F-measure value, the better the classifier. The calculation of the F-measure is shown in Eqs. (22), (23), and (24):

$$AUC = \frac{\sum_{i \in defectiveClass} rank_i - \frac{M(M+1)}{2}}{M \times N} \tag{21}$$

$$Recall = \frac{|\{predicted\ defective\ methods\} \cap \{defective\ methods\}|}{|\{defective\ methods\}|} \tag{22}$$

$$Precision = \frac{|\{predicted\ defective\ methods\} \cap \{defective\ methods\}|}{|\{predicted\ defective\ methods\}|} \tag{23}$$

$$F - measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{24}$$

5.5 Experiment results

Five research questions were devised to investigate the performance of the proposed Adan approach:

- RQ1: How is the performance of the proposed Adan approach for within-project and cross-project defect prediction?
- RQ2: How is the performance of the proposed Adan approach compared with state-of-the-art techniques for cross-project defect prediction?
- RQ3: What is the sensitivity of the parameter setting of the proposed Adan approach for cross-project defect prediction?
- RQ4: What are the effects of the autoencoder, the global discriminator, and the local discriminator on the performance of the proposed Adan approach for cross-project defect prediction?
- RQ5: What is the effect of the focal loss function on dealing with imbalanced defect distribution in the proposed Adan approach for cross-project defect prediction?

To answer RQ1, a series of experiments was conducted on the collected data to investigate the performance of the proposed Adan approach for within-project and cross-project defect prediction. Table 5 shows the performances of the proposed Adan approach, where the diagonal AUC indicates the performances for within-project defect prediction, and the off-diagonal AUC indicates the performances for cross-project defect prediction. Table 5 shows that the performance of the proposed Adan approach for within-project prediction was better than that of cross-project prediction.

This outcome can be explained as both training and test data are drawn from the same project. Although transfer learning can alleviate the discrepancy between the distributions (i.e., the global distribution and the local distribution) of the defect data in the source and target projects, it cannot completely remove the distribution difference. In cross-project defect prediction, the training data and testing data are drawn from different software projects. Thus, a great discrepancy still exists between the distributions of the source and target projects in the shared feature space. It was also found that in some cross-project defect-prediction tasks, the proposed Adan approach can perform comparable to that of within-project defect prediction, such as Kafka→Kylin and Kylin→Kafka. This outcome demonstrates that the proposed Adan approach can produce favorable performance in defect prediction when the data distributions of the source and target projects are similar.

Table 5 The AUC of the proposed Adan approach in the scenarios of within-project and cross-project. The best outcomes are marked in bold

Target project					
		Kafka	Kylin	Ant	Tomcat
Source project	Kafka	**0.9031**	0.8840	0.8425	0.7535
	Kylin	0.8507	**0.9156**	0.8855	0.8499
	Ant	0.7381	0.8346	**0.9192**	0.7894
	Tomcat	0.7477	0.8400	0.7502	**0.9235**

To answer RQ2, a series of experiments was conducted to compare the proposed Adan and baseline methods on the 12 pairs of tasks on cross-project defect prediction mentioned in Sect. 5.1. Table 6 shows the experimental results and demonstrates that the proposed Adan approach can significantly outperform state-of-the-art techniques in most cases, with an average improvement of 8.42% in terms of AUC. Figure 4 shows a box plot of the performance of the proposed Adan approach and the baseline methods for 12 pairs of tasks for cross-project defect prediction.

The outcomes in Table 6 and Fig. 4 are explained as follows: First, the proposed Adan approach takes into account the alignment of both global and local distributions between the source and target projects. The global discriminator and local discriminators collaboratively align the data distributions of source and target projects in the project-invariant feature space. This is achieved by the adversarial learning inherent in feature extractors and domain discriminators. The global distribution alignment is used to match the global distribution of the features in the dataset, while the local distribution alignment is used to match the class-specific decision boundaries. Existing methods primarily concentrate on aligning either the global distribution or the local distribution between the source and target projects. Nevertheless, they ignore the relative importance of global and local distributions when extracting project-invariant features.

Second, deep learning is adopted to construct a project-invariant feature space that can effectively determine nonlinear correlations between source and target distributions. Baseline methods using linear transformation (e.g., CCA+) or kernel transformation (e.g., TCA+) can only capture the

Table 6 The AUC was derived from the proposed Adan approach and the baseline methods for 12 pairs of tasks for cross-project defect prediction. The best outcomes are marked in bold

Source	Target	Adan	TNB	TCA+	CCA+	DSSDPP	DMDAJFR	Improvement
Kafka	Kylin	**0.8840**	0.6049	0.6849	0.6036	0.7958	0.8155	8.39%
	Ant	**0.8425**	0.6049	0.6849	0.6036	0.7404	0.6043	13.78%
	Tomcat	**0.7535**	0.6608	0.6606	0.7237	0.7436	0.5816	1.33%
Kylin	Kafka	**0.8507**	0.7159	0.7057	0.5876	0.7297	0.6252	16.58%
	Ant	**0.8855**	0.7586	0.6192	0.6179	0.7522	0.5954	16.72%
	Tomcat	**0.8499**	0.6391	0.7529	0.7037	0.7408	0.5794	12.8%
Ant	Kafka	**0.7381**	0.6901	0.6127	0.5651	0.7330	0.6561	0.69%
	Kylin	**0.8346**	0.7031	0.7082	0.6022	0.7942	0.5106	5.08%
	Tomcat	**0.7894**	0.7049	0.7035	0.7032	0.7489	0.5081	5.40%
Tomcat	Kafka	**0.7477**	0.6497	0.6586	0.6733	0.6921	0.6090	8.03%
	Kylin	**0.8400**	0.6221	0.7734	0.7430	0.7694	0.6058	8.61%
	Ant	**0.7502**	0.5602	0.7033	0.6453	0.7267	0.6169	3.23%

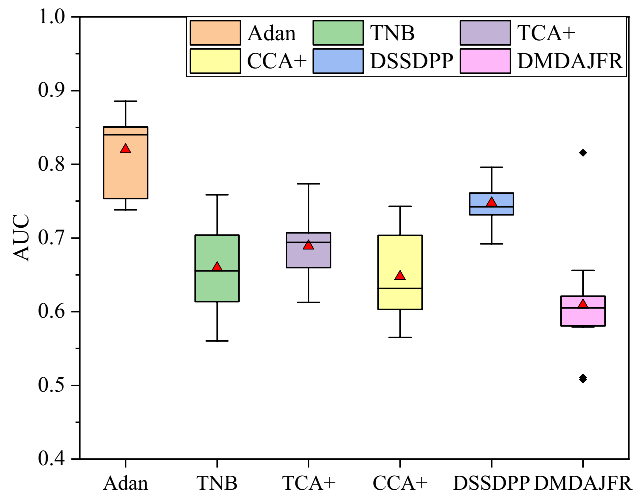

Fig. 4 Box plot of the performance of the proposed Adan approach and the baseline methods for 12 pairs of tasks on cross-project defect prediction

linear or low-order correlations between source and target distributions, and this would limit the distribution alignment capability of baseline methods. The DSSDPP method tends to make type I errors that judge nondefective artifacts as defective, while the DMDAJFR method tends to make type II errors that misclassify defective artifacts as nondefective. The results show that the DSSDPP attempts to balance defective and nondefective artifacts using upsampling, which results in a biased learning process for the classifier. The DMDAJFR completely ignores the imbalanced label distribution in cross-project defect prediction, resulting in serious prediction bias toward nondefective artifacts.

The baseline method CCA + holds that there is a linear relationship between the data distribution in different projects that constrains the CCA + method from modeling higher-order correlations between different defect data and constructing nonlinear feature mapping. The TCA + method suffers from significant shifts in data distribution when kernel transformation is adopted. The TNB approach weighs training instances based on their similarity to that of the target project and lacks consideration of the relationships between features. As a result, the TNB method aligns only the data distribution of individual features between the source and target projects, while there is still a significant discrepancy in the joint distribution of the features between the source and target projects.

To answer RQ3, two kinds of sensitivity analysis are conducted on the parameters of the proposed Adan approach as the dimension of the extracted project-invariant feature space D^f and, the balance factor α and focusing factor γ in the focal loss function. The dimension of the extracted features D^f (i.e., $|G_f(x_k^{(\cdot)})|$) is a critical parameter in the proposed Adan approach. To investigate the influence of D^f on the performance of the proposed Adan approach, the parameter D^f is tuned from 25 to 55 at an interval of 5 in the 12 pairs of cross-project defect-prediction tasks.

Table 7 shows the results for tuning the dimension of the extracted project-invariant features in 12 pairs of CPDP tasks. Firstly, the performance of the proposed Adan approach, as measured by the AUC in each task, initially rises and then declines with increasing feature dimension D^f. Second, it can be seen that the optimal feature dimension is dependent on a specific source project and target project. The optimal feature dimension is asymmetric for source and target projects. Taking the Kafka and Tomcat project as an instance, the optimal feature dimension D^f is 30 in the task Kafka → Tomcat, and it is 55 in the task Tomcat → Kafka. This result can be understood as a defect predictor trained on the labeled data that comes from a source project; thus, a feature extractor tends to extract discriminative features from the source project.

Different software projects have different discriminative features arising from the differences in its design patterns, software framework, coding style, and so on. Therefore, the

Table 7 The cross-project defect-prediction performance of the proposed Adan approach in terms of AUC when varying the dimensions of extracted features from 25 to 55 at an interval of 5. The best performances are marked in bold

Source	Target	25	30	35	40	45	50	55
Kafka	Kylin	0.8058	0.8441	0.8482	0.8400	0.8521	**0.8840**	0.8500
	Ant	0.7366	0.7615	0.7576	**0.8425**	0.7288	0.7768	0.7427
	Tomcat	0.7496	**0.7535**	0.6705	0.7035	0.6504	0.7100	0.7122
Kylin	Kafka	0.7103	0.7122	**0.8507**	0.7498	0.7193	0.7258	0.7200
	Ant	0.8840	**0.8855**	0.8822	0.8580	0.8709	0.8667	0.8678
	Tomcat	0.8121	0.8102	**0.8499**	0.8057	0.7265	0.8026	0.7849
Ant	Kafka	0.6961	**0.7381**	0.6589	0.6845	0.6262	0.6441	0.6424
	Kylin	0.7934	0.8023	**0.8346**	0.7874	0.7891	0.7589	0.7375
	Tomcat	0.7279	0.7475	0.7541	**0.7894**	0.7243	0.6966	0.7108
Tomcat	Kafka	0.6933	0.6705	**0.7477**	0.6820	0.6599	0.6151	0.6044
	Kylin	0.7295	0.7400	0.6804	0.7893	0.7769	**0.8400**	0.8205
	Ant	0.7183	0.7280	0.6896	**0.7502**	0.6733	0.7175	0.6732

optimal feature dimension varies across different source projects. However, the number of samples and class imbalances vary across different software projects (see Table 1). The defective proportion was 11.72% in Kafka (with 10,504 samples) and 2.86% in Tomca (with 19,487 samples). To prevent the learned defect predictor bias toward the majority class and facilitate the convergence of Adan, the optimal dimension of extracted features should be set differently in symmetric cross-project defect-prediction tasks, such as Kafka → Tomcat and Tomcat → Kafka.

The balance factor α in the focal loss in Eq. (9) assigns different weights to defective or nondefective classes, making the training focus on the samples of defective class by increasing its loss contribution. The focusing factor γ in the focal loss in Eq. (7) modulates the loss contribution from easy-classified and hard-classified samples. A large value for γ will make the loss function L_y focus more on the hard-classified samples than on the easy-classified samples. In training the proposed Adan approach, the defect predictor G_y is trained on the labeled data from the source project. Table 5 shows that the defect predictor achieves satisfactory defect-prediction performance within the source project, and this serves as a foundation for its application to the target project. Therefore, the parameters α and γ are tuned in the focal loss on four within-project defect-prediction tasks. To find the optimal parameter settings, the balance factor α is tuned from 0.2 to 0.8 at an interval of 0.2, and the focusing factor γ from 0 to 2 at an interval of 0.25.

Figure 5 shows the performance of the defect predictor in the four within-project defect-prediction tasks measured by the F-measure when varying the parameters α and γ. In the heat map, brighter colors indicate superior model performance, while darker colors indicate the opposite. The optimal settings were $\alpha = 0.6$ and $\gamma = 0.7$ in the Kafka project, $\alpha = 0.6$ and $\gamma = 0$ in the Kylin project, $\alpha = 0.6$ and $\gamma = 0.75$ in the Ant project, $\alpha = 0.4$ and $\gamma = 0.25$ in the Tomcat project.

The results can be explained as follows: To improve the performance of the defect predictor G_y, the focal loss L_y was used to evaluate the prediction loss in the training process of Adan. As shown in Eqs. (2) and (4), the balance factor $\alpha \in (0,1)$ was adopted to dynamically adjust the loss contribution of defective or nondefective classes. It assigned different weights to these two classes, focusing more on the defective samples by increasing their loss contribution. This parameter mitigated the influence of class imbalance in defect data distribution on the performance of the learned defect predictor. The focusing factor γ enabled the focal loss to downweight the loss contribution from easily classified examples and focused more on hard and misclassified examples in the training process. Large values of gamma rapidly reduced the weight of loss from well-classified examples to focus training on correcting hard misclassifications.

To answer RQ4, a comparative experiment was conducted between the proposed Adan approach and its three variants, Adan-G-L, Adan-A-L, and Adan-A-G. The variant Adan-G-L did not include the autoencoder (i.e., the feature decoder G_r in Fig. 1 was removed). The variant Adan-A-G did not include the global discriminator (i.e., G_d in Fig. 1 was removed). The variant Adan-A-G did not include the local discriminator (i.e., G_l in Fig. 1 is removed). This ablation study was conducted to investigate the effects of different components, such as the autoencoder, the global discriminator, and the local discriminator, on the performance of the proposed Adan approach for cross-project defect prediction.

Table 8 shows the performance comparison between the proposed Adan approach and its three variants. First, the comparison between the proposed Adan approach and the variant Adan-G-L shows that the absence of autoencoder leads to a consistent performance decrease across all 12 cross-project defect-prediction tasks. The autoencoder plays a critical role in learning the project-invariant feature space to ensure that the representations of the source and target

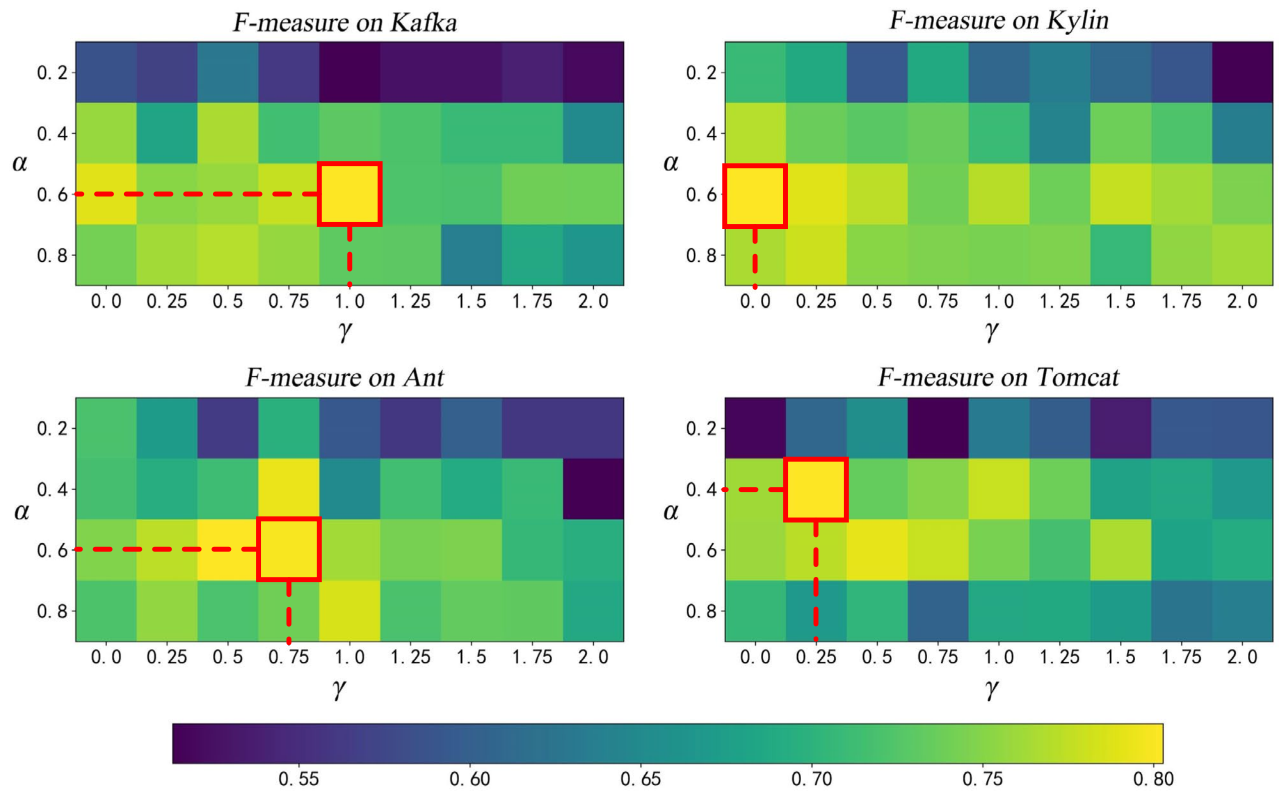

Fig. 5 Performances of the defect predictor were measured by F-measure for within-project defect prediction when varying the balance factor α and focusing factor γ

Table 8 Performances were measured by the AUC of the proposed Adan approach and its variants in the 12 pairs of cross-project defect prediction tasks. The best outcomes are marked in bold

Source	Target	Adan	Adan-G-L	Adan-A-L	Adan-A-G
Kafka	Kylin	**0.8840**	0.8073	0.8154	0.8200
	Ant	**0.8425**	0.6724	0.5441	0.5519
	Tomcat	**0.7535**	0.7180	0.6764	0.6837
Kylin	Kafka	**0.8507**	0.7318	0.7566	0.7559
	Ant	**0.8855**	0.8804	0.8833	0.8738
	Tomcat	**0.8499**	0.7684	0.7839	0.7783
Ant	Kafka	**0.7381**	0.7314	0.6907	0.5949
	Kylin	**0.7894**	0.6895	0.6742	0.6724
	Tomcat	**0.8346**	0.7057	0.8014	0.8289
Tomcat	Kafka	**0.7477**	0.7065	0.6995	0.6153
	Kylin	**0.8100**	0.7753	0.7070	0.7022
	Ant	**0.7502**	0.7384	0.7293	0.7196

artifacts in the project-invariant feature space will not cause too much information loss with respect to the representations in their own spaces (i.e., the source space and the target space).

Second, the global discriminator ensures that the representations of the source and target artifacts in the project-invariant feature space are of approximate global distribution, which is indispensable for transferring the global knowledge of defect prediction in the source project to the target project. Particularly when the global distributions of the artifacts between the source and target projects are of great difference, the removal of the global discriminator will bring about a large decrease in the performance of cross-project defect prediction. For instance, as demonstrated in the Tomcat→Kylin task where the Tomcat project has the largest number of artifacts at 19,487, and the Kylin project has the smallest number of artifacts at 7,077, the AUC score declines markedly from 0.8100 to 0.7070 from the proposed Adan approach to the variant Adan-A-L.

Third, the local discriminator contributes to local knowledge transfer from the source project to the target project by aligning its feature distribution condition with defects in the project-invariant feature space. When the distributions of defects in the artifacts between the source and target projects are of great difference, there will be a large performance decrease without the local discriminator. For instance, in the Ant→Kafka task where the Ant and Kalka projects have defective proportions of 2.81% and 11.72%, respectively, the

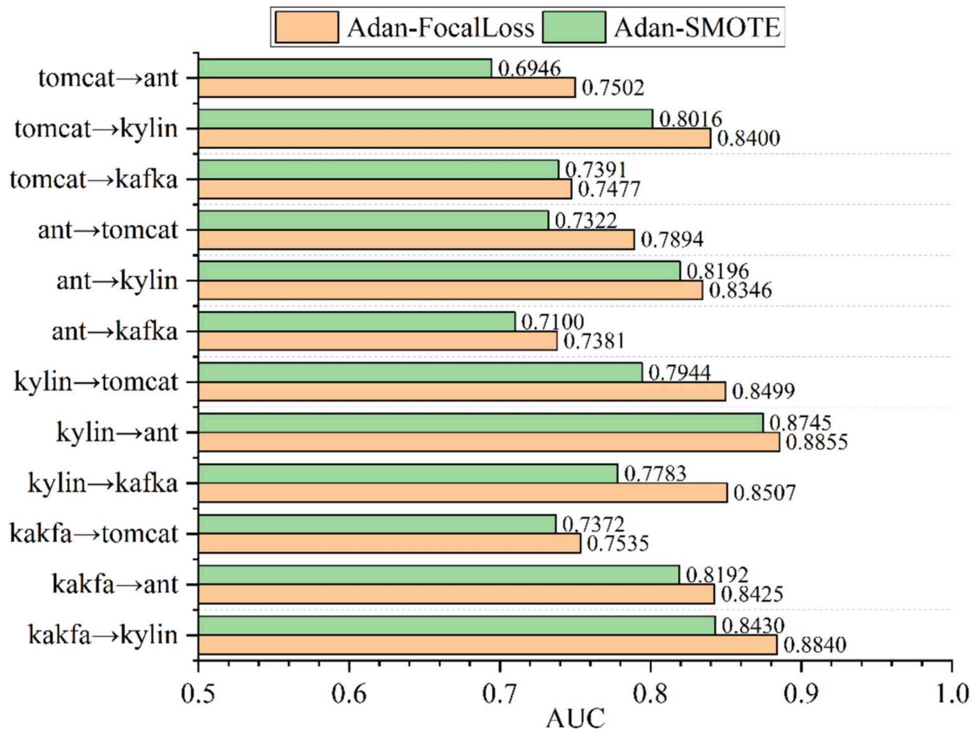

Fig. 6 The performances of the proposed Adan approach with focal loss and cross-project defect prediction with SMOTE were measured by AUC on 12 pairs of cross-project defect prediction tasks

AUC of the proposed Adan approach is dropped from 0.7381 to 0.5946 without the local discriminator.

To answer RQ 5, the SMOTE (Synthetic Minority Oversampling Technology) algorithm [5] is employed to sample the defective artifacts from the source projects and use the cross entropy with logits[6] as the loss function for the defector predictor as shown in Eq. (25). The basic idea of SMOTE is to generate synthetic artifacts of the defective class by interpolating existing defective artifacts. Note that in Eq. (25), it has $\hat{y}_i^0 + \hat{y}_i^1 = 1$:

$$L_y = \frac{1}{n_S} \sum_{i=1}^{n_S} -\hat{y}_i^0 \log(\hat{y}_i^0) - \hat{y}_i^1 \log(\hat{y}_i^1) \tag{25}$$

Figure 6 shows the experimental results of the performance comparison of the proposed Adan approach with focal loss and SMOTE. It can be seen that, in all pairs of tasks, the proposed Adan approach with focal loss significantly outperforms that with SMOTE. Moreover, the more defects in the source project, the better the performance of the proposed Adan approach with SMOTE. That is, the proposed Adan approach with SMOTE performs better in using source projects such as Kafka and Kylin than in using source projects such as Tomcat and Ant. The outcome can be explained by highly imbalanced scenarios, such as using Tomcat and Ant as the source projects. SMOTE can lead to overfocusing on defective artifacts at the expense of prediction accuracy for nondefective artifacts. The SMOTE only generates synthetic defective artifacts through the linear interpolation of existing defective artifacts, which cannot guarantee the quality and diversity of the generated defective artifacts and may even amplify the noise and outliers in the original dataset, failing to effectively deal with the problem of imbalanced data distribution.

6 Conclusion

Data scarcity poses a great challenge in software defect prediction, especially for a small project or one in its early development stage. This paper proposes a novel approach called Adan to make use of an autoencoder and dynamic adversarial adaptation for cross-project defect prediction. The proposed Adan approach makes use of an autoencoder to produce latent space as a project-invariant feature space of the source artifacts and the target artifacts while taking into account the alignment of global and local distributions between source and target projects. The dynamic adversarial factor ω was adopted to dynamically adjust the relative importance of global and local distributions in model training. The dynamic adversarial adaptation enabled the proposed Adan approach to extract a project-invariant feature space with global and local distribution alignment between source and target projects to improve defect-prediction

[6] Tensorflow API, online: https://tensorflow.google.cn/api_docs/python/tf/nn/weighted_cross_entropy_with_logits?hl=en.

knowledge transfer from a source project to a target project. The focal loss function was also devised in the proposed Adan approach to deal with the imbalanced distribution of defective artifacts in source projects. The source code, Git commit history, and bug reports were collected from four open-source software projects, and 12 pairs of tasks on cross-project defect prediction were devised to investigate the performance of the proposed Adan approach. The experimental results validate the superiority of the proposed Adan approach over state-of-the-art techniques.

Although the proposed Adan approach has shown promising prospects for cross-project defect prediction compared to state-of-the-art techniques, we admit that it can be further improved in three directions. First, we will investigate the artificial network structure of the defect predictor G_y to improve its predictive performance. Also, it is possible to make the neural network structures of G_y, G_d, G_d^0, and G_d^1 different from each other to further improve the performance of the proposed Adan approach. Second, we will attempt to classify defects into more categories according to their severity and priority rather than merely whether they are defective and nondefective (as was done in this current study) and extend the proposed Adan approach to multi-class cross-project prediction [1, 35]. Third, we will extend the proposed Adan approach to application domains other than cross-project defect prediction, such as bioinformatics [33], recommendation systems [45], and deceptive review identification [46].

Acknowledgements This research was supported in part by the National Natural Science Foundation of China under Grant Nos. 72174018 and 71932002; the Beijing Natural Science Fund under Grant No. 9222001; the Philosophy and Sociology Science Fund from Beijing Municipal Education Commission (SZ202111000500l); Beijing Postdoctoral Research Foundation under Grant No. 2023-ZZ-176.

Data Availability The data that support the findings of this study is available from the corresponding author upon reasonable request.

Declarations

Competing Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Arokiam J, Bradbury JS (2020) Automatically predicting bug severity early in the development process. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER ’20. Association for Computing Machinery, New York, NY, USA, pp. 17–20. https://doi.org/10.1145/3377816.3381738
2. Berahmand K, Daneshfar F, Salehi ES et al (2024) Autoencoders and their applications in machine learning: a survey. Art Intell Rev 57(2):28. https://doi.org/10.1007/s10462-023-10662-6
3. Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don’t touch my code! examining the effects of ownership on software quality. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE ’11. Association for Computing Machinery, New York, NY, USA, pp. 4–14. https://doi.org/10.1145/2025113.2025119
4. Bhat NA, Farooq SU (2023) An empirical evaluation of defect prediction approaches in within-project and cross-project context. Software Qual J 31:917–946. https://doi.org/10.1007/s11219-023-09615-7
5. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intell Res 16:321–357. https://doi.org/10.1613/jair.953
6. Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Software Eng 20:476–493. https://doi.org/10.1109/32.295895
7. Chen J, Hu K, Yang Y, Liu Y, Xuan Q (2020) Collective transfer learning for defect prediction. Neurocomputing 416:103–116
8. Fawcett T (2006) An introduction to ROC analysis. Pattern Recogn Lett 27(8):861–874. https://doi.org/10.1016/j.patrec.2005.10.010
9. Ganin Y, Ustinova E, Ajakan H, Germain P, Larochelle H, Laviolette F, Marchand M, Lempitsky V (2016) Domain-adversarial training of neural networks. The journal of machine learning research 17:2096–2030
10. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. diacriTech, Chennai
11. Hassan AE (2009) Predicting faults using the complexity of code changes. In Proceedings of IEEE 31st International Conference on Software Engineering, ICSE2009, Vancouver, pp 78–88. https://doi.org/10.1109/ICSE.2009.5070510
12. Hosseini S, Turhan B, Gunarathna D (2019) A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans Software Eng 45:111–147. https://doi.org/10.1109/TSE.2017.2770124
13. Jiang W, Qiu S, Liang T, Zhang F (2023) Cross-project clone consistent-defect prediction via transfer-learning method. Inf Sci 635:138–150. https://doi.org/10.1016/j.ins.2023.03.118
14. Jing X, Wu F, Dong X, Qi F, Xu B (2015) Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015. Association for Computing Machinery, New York, NY, USA, pp. 496–507. https://doi.org/10.1145/2786805.2786813
15. Li Z, Niu J, Jing X, Yu W, Qi C (2021) Cross-project defect prediction via landmark selection-based kernelized discriminant subspace alignment. IEEE Trans Reliab 70:996–1013. https://doi.org/10.1109/TR.2021.3074660
16. Li Z, Zhang H, Jing X, Xie J, Guo M, Ren J (2023) DSSDPP: data selection and sampling based domain programming predictor for cross-project defect prediction. IEEE Trans Software Eng 49:1941–1963. https://doi.org/10.1109/TSE.2022.3204589
17. Lin T, Goyal P, Girshick R, He K, Dollár P (2017) Focal loss for dense object detection. In Proceedings of the IEEE International Conference on Computer Vision, ICCV’17, Venice, pp 2980–2988. https://doi.org/10.1109/ICCV.2017.324
18. Liu J, Li J, Lu K (2018) Coupled local–global adaptation for multi-source transfer learning. Neurocomputing 275:247–254. https://doi.org/10.1016/j.neucom.2017.06.051
19. Liu C, Yang D, Xia X, Yan M, Zhang X (2019) A two-phase transfer learning model for cross-project defect prediction. Inf Softw

Technol 107:125–136. https://doi.org/10.1016/j.infsof.2018.11.005
20. Luo L, Wang W, Bao S, Peng X, Peng Y (2024) Robust and sparse canonical correlation analysis for fault detection and diagnosis using training data with outliers. Expert Syst Appl 236:121434. https://doi.org/10.1016/j.eswa.2023.121434
21. Ma Y, Luo G, Zeng X, Chen A (2012) Transfer learning for cross-company software defect prediction. Inf Softw Technol 54:248–256. https://doi.org/10.1016/j.infsof.2011.09.007
22. McCabe TJ (1976) A complexity measure. IEEE Trans Software Eng SE-2:308–320. https://doi.org/10.1109/TSE.1976.233837
23. Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In Proceedings of 35th International Conference on Software Engineering (ICSE), ICSE2013, San Francisco, pp 382–391. https://doi.org/10.1109/ICSE.2013.6606584
24. Ni C, Xia X, Lo D, Chen X, Gu Q (2022) Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans Software Eng 48:786–802. https://doi.org/10.1109/TSE.2020.3001739
25. Pan S, Yang Q (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359. https://doi.org/10.1109/TKDE.2009.191
26. Pei Z, Cao Z, Long M, Wang J (2018) Multi-adversarial domain adaptation. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI'18. New Orleans, pp 3934–3941. https://doi.org/10.5555/3504035.3504517
27. Sharma BU, Sadam R (2023) How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model. J Syst Softw 195:111522. https://doi.org/10.1016/j.jss.2022.111522
28. Singh M, Chhabra JK (2024) Machine learning based improved cross-project software defect prediction using new structural features in object oriented software. Appl Soft Comput 165:112082. https://doi.org/10.1016/j.asoc.2024.112082
29. Sotto-Mayor B, Kalech M (2021) Cross-project smell-based defect prediction. Soft Comput 25(22):14171–21418. https://doi.org/10.1007/s00500-021-06254-7
30. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learning Res 15:1929–1958. https://doi.org/10.5555/2627435.2670313
31. Sun Z, Li J, Sun H, He L (2021) CFPS: collaborative filtering based source projects selection for cross-project defect prediction. Appl Soft Comput 99:106940. https://doi.org/10.1016/j.asoc.2020.106940
32. Tabassum S, Minku LL, Feng D (2023) Cross-project online just-in-time software defect prediction. IEEE Trans Software Eng 49:268–287. https://doi.org/10.1109/TSE.2022.3150153
33. Theodoris CV, Xiao L, Chopra A, Chaffin MD, Al Sayed ZR, Hill MC, Mantineo H, Brydon EM, Zeng Z, Liu XS, Ellinor PT (2023) Transfer learning enables predictions in network biology. Nature 618:616–624. https://doi.org/10.1038/s41586-023-06139-9
34. Tong H, Liu B, Wang S, Li Q (2019) Transfer-learning oriented class imbalance learning for cross-project defect prediction, ArXiv, pp 1–38. https://doi.org/10.48550/arXiv.1901.08429
35. Umer Q, Liu H, Illahi I (2020) CNN-based automatic prioritization of bug reports. IEEE Trans Reliab 69:1341–1354. https://doi.org/10.1109/TR.2019.2959624
36. Wang S, Liu T, Nam J, Tan L (2020) Deep semantic feature learning for software defect prediction. IEEE Trans Software Eng 46:1267–1293. https://doi.org/10.1109/TSE.2018.2877612
37. Wang D, Li T, Deng P, Zhang F, Huang W, Zhang P, Liu J (2023) A generalized deep learning clustering algorithm based on non-negative matrix factorization. ACM Trans Knowl Discov Data 17(7):1–20. https://doi.org/10.1145/3584862
38. Xie J, Huang B, Dubljevic S (2022) Transfer learning for dynamic feature extraction using variational Bayesian inference. IEEE Trans Knowl Data Eng 34:5524–5535. https://doi.org/10.1109/TKDE.2021.3054671
39. Yu C, Wang J, Chen Y, Huang M (2019) Transfer learning with dynamic adversarial adaptation network. In: Proceedings of the 2013 IEEE International, Conference on Data Mining (ICDM), ICDM'13. IEEE, Beijing, pp 778–786. https://doi.org/10.1109/ICDM.2019.00088
40. Zhang W, Yan S, Li J, Tian X, Yoshida T (2022) Credit risk prediction of SMEs in supply chain finance by fusing demographic and behavioral data. Transp Res Part E: Logistics Trans Rev 158:102611. https://doi.org/10.1016/j.tre.2022.102611
41. Zhou K, Liu Z, Qiao Y, Xiang T, Loy CC (2023) Domain generalization: a survey. IEEE Trans Pattern Anal Mach Intell 45:4396–4415. https://doi.org/10.1109/TPAMI.2022.3195549
42. Zhuang F, Qi Z, Duan K, Xi D, Zhu Y, Zhu H, Xiong H, He Q (2021) A comprehensive survey on transfer learning. Proc IEEE 109:43–76. https://doi.org/10.1109/JPROC.2020.3004555
43. Zou Q, Lu L, Yang Z, Gu X, Qiu S (2021) Joint feature representation learning and progressive distribution matching for cross-project defect prediction. Inf Softw Technol 137:106588. https://doi.org/10.1016/j.infsof.2021.106588
44. Zhang W, Li Z, Wang Q, Li J (2019) FineLocator: a novel approach to method-level fine-grained bug localization by query expansion. Inf Softw Technol 110:121–135. https://doi.org/10.1016/j.infsof.2019.03.001
45. Zhang W, Li X, Li J, Yang Y (2020) A two-stage rating prediction approach based on matrix clustering on implicit information. IEEE Trans Comp Social Syst 7(2):517–535. https://doi.org/10.1109/TCSS.2019.2960858
46. Zhang W, Zhang X, Chen J, Li J, Ma Z (2024) Stacking GA2M for inherently interpretable fraudulent reviewer identification by fusing target and non-target features. Int J Gen Syst 1–36. https://doi.org/10.1080/03081079.2024.2384404

Wen Zhang received his Ph.D. degree in knowledge science from the Japan Advanced Institute of Science and Technology, Japan, in 2009. He is currently a full Professor with College of Economics and Management, Beijing University of Technology, China. He is the author of more than 60 publications including Annals of Operations Research, Decision Support Systems, Information Processing and Management, Knowledge and Information Systems. His recent research interests include mining software repository, big data analytics, recommendation system and deep learning.

Jiangpeng Zhao received his B.M. degree in information management and systems from the University of Shanghai for Science & Technology, China, in 2020. He is currently a PhD candidate in Management Science and Engineering at the School of Economics and Management, Beijing University of Technology. His recent research interests include mining software repository, and recommendation system.

Guangjie Qin received his M.S. degree in management science and engineering from the Beijing University of Technology, China, in 2022. He is currently a senior software engineer at Inspur New Infrastructure Technology with focus on design and development of advanced software systems for large-scale infrastructure projects. His research interests include software system reliability, opensource software ecosystems, and smart cities.

Song Wang received his Ph.D. degree in computer engineering from the University of Waterloo in 2018, M.S. degree from Chinese Academy of Sciences in 2014. Before that, he got both his B.E. degree in Software Engineering and BHRM degree in Human Resource Management from Sichuan University in 2011. He is currently an associate professor with department of electrical engineering and computer science, York University, Canada. He has published more than 50 papers in leading journals and conferences including IEEE Transaction on Software Engineering, ACM Transactions on Software Engineering and Methodology and International Conference on Software Engineering. His research interests include software engineering, software reliability, program analysis, software testing, and machine learning.