A Cost-Sensitive Shared Hidden Layer Autoencoder for Cross-Project Defect Prediction

Juanjuan Li[1], Xiao-Yuan Jing[2(✉)], Fei Wu[1], Ying Sun[1], and Yongguang Yang[1]

[1] College of Automation, Nanjing University of Posts and Telecommunications, Nanjing, China
[2] School of Computer, Wuhan University, Wuhan, China
jingxy_2000@126.com

Abstract. Cross-project defect prediction means training a classifier model using the historical data of the other source project, and then testing whether the target project instance is defective or not. Since source and target projects have different data distributions, and data distribution difference will degrade the performance of classifier. Furthermore, the class imbalance of datasets increases the difficulty of classification. Therefore, a cost-sensitive shared hidden layer autoencoder (CSSHLA) method is proposed. CSSHLA learns a common feature representation between source and target projects by shared hidden layer autoencoder, and makes the different data distributions more similar. To solve the class imbalance problem, CSSHLA introduces a cost-sensitive factor to assign different importance weights to different instances. Experiments on 10 projects of PROMISE dataset show that CSSHLA improves the performance of cross-project defect prediction compared with baselines.

Keywords: Shared hidden layer autoencoder · Cost-sensitive learning · Cross-project software defect prediction

1 Introduction

Software defect prediction (SDP) has been a hot research topic in software engineering [23]. Its main goal is to discover defects exist in the software for improving the software quality. The previous research mainly focused on within-project defect prediction (WPDP) [19, 20], mainly using the historical data of one project to train a prediction model and testing the defect tendency of the same project software instance. However, when there is not enough historical data available in the same project, the performance of WPDP becomes significantly worse, and cross-project defect prediction (CPDP) can be considered.

Training a prediction model by using plenty of historical data from other project and predicting defects in a new project instances, is called cross-project defect

The first author is a student.

Z. Lin et al. (Eds.): PRCV 2019, LNCS 11859, pp. 491–502, 2019.
https://doi.org/10.1007/978-3-030-31726-3_42

prediction(CPDP) [6, 15]. However, its prediction performance is usually poor, because of the data distribution difference phenomenon between source and target projects, e.g., coding styles, programming language [4]. If the data distribution difference between source project and target project is small enough [8], CPDP model can achieve better results. To solve the problem of data distribution difference in CPDP, several CPDP methods have been developed [6, 15]. However, these methods [4, 8, 15] use the traditional features rather than deep features extracted by deep learning. Such as TCA+ [15], which maps source and target projects into a latent subspace, making the difference of data distribution between source and target projects is minimized. Deep learning has been successfully applied to the field of speech recognition [10] and image classification [1] due to its powerful feature learning capability. Stacked denoising autoencoders model [7] is applied in the field of SDP and proved that the deep features are more promising than the traditional software metric. Furthermore, the shared-hidden-layer autoencoder' method has solved the typical inherent mismatch between the two domains in the field of speech emotion recognition [10, 11].

Besides, class imbalance problem reduces the prediction performance of the CPDP model. That is, the number of defect-free instances is far greater than that of defective instances [2]. Thus the SDP model will more likely to identify defect-free instances. Especially for minority classes, imbalanced distribution is the main reason of poor performance of certain classification models [16]. In this paper, cost-sensitive technique is used to deal with class imbalance problem.

Similar to the idea of transfer learning, a cost-sensitive shared hidden layer autoencoder (CSSHLA) method is proposed for CPDP to solve the data distribution difference problem and the class imbalance problem. It mainly includes two phases: feature extraction stage and classifier learning phase. In the feature extraction stage, we extract a set of deep nonlinear features from the source and target projects by using shared hidden layer autoencoder. In the classifier learning phase, we build a cost-sensitive softmax classifier based on the deep features of source project data.

The main contributions of this paper can be summarized as follows:

1. We propose a shared hidden layer autoencoder for CPDP. It can extract deep feature representations from original features, making the data distribution of source and target projects be more similar in the nonlinear feature subspace to solve the data distribution difference problem. It can also make the instances of same class in source project be more compact.
2. To alleviate the class imbalance problem, we propose a cost-sensitive softmax classification technique. Different misclassification costs are assigned to instances from different class in the model building stage. In this way, the features of defective instances can be better learned.
3. Based on the above two techniques, a cost-sensitive shared hidden layer autoencoder method (CSSHLA) is proposed for CPDP. We evaluate CSSHLA with the baselines on the 10 projects from PROMISE. One conclusion is that we get better results on F-measure and Accuracy than other baselines.

2 Related Work

2.1 Cross-Project Defect Prediction

In recent years, CPDP is a hot topic in software engineering [22]. The most important problem is data distribution difference problem between source and target projects in CPDP. TCA+ [15] is an effective CPDP method that uses transfer component analysis (TCA) to map instance of source and target projects into a common latent subspace, and the difference of feature distributions between the source and the target is small enough. Dynamic cross-company mapped model learning (Dycom) is first used in web effort estimation [13], and transfer learning method Dycom [4] is successfully applied to CPDP, in which 10% of the labeled data comes from the target project in the training process. Log transformations (LT) [3] reduces the data distribution difference by log transformation of the feature values in the source and the target projects, and then aligns the median values of each source project and target project. Training data selection (TDS) [8] selects the most suitable training data related to the test data based on the similarity distance to improve the performance of cross-project defect prediction.

Most of the CPDP methods do not consider the class imbalance problem. In our approach, we consider the class imbalance problem and use cost-sensitive factor to solve the problem.

2.2 Deep Learning

In recent years, deep learning has been successfully applied in many fields because of its powerful feature generation ability, such as speech emotion recognition [9], image classification [1], face recognition [5], etc. Convolution neural network (CNN), deep belief network (DBN) and autoencoder play an important role in deep learning. Wang et al. [17] used DBN to learn the most relevant semantic features from the program's Abstract Grammar Tree (AST) and showed that the deep semantic features are better than the traditional features. Good results are obtained by using DBN to predict the defects for just-in-time defect prediction [21] than without deep feature representation. By integrating the similar feature learning technology and distance measurement learning technology, siamese dense neural network [14] is successfully applied to SDP. The autoencoder has been successfully applied to the field of speech recognition [10]. Some researchers applied autoencoder in the field of defect prediction [7].

Because of the strong feature extraction advantage of deep learning, a deep learning autoencoder method is introduced to solve the CPDP problem in our paper. And the improved autoencoder is used to solve the problem of data distribution difference in CPDP effectively.

3 Proposed Methodology

We design a cost-sensitive shared hidden layer autoencoder (CSSHLA) network, and the overall framework of CSSHLA is shown in Fig. 1. It is mainly summarized as two stages: (1) Data normalization. It makes the source data and target data have same order

of magnitude. (2) Training network. The network is composed of feature extraction model and classifier model. Feature extraction model takes the source data and target data as input and outputs their deep features. Classifier model uses the deep source features and its corresponding labels to build a classifier.

Let $x \in \{X_{tr} \cup X_{te}\}$, $X_{tr} = \{x_{tr}^i\}_{i=1}^{N_{tr}} \in R^{N_{tr} \times n}$ and $X_{te} = \{x_{te}^i\}_{i=1}^{N_{te}} \in R^{N_{te} \times n}$ mean feature sets from source and target projects, respectively, and $Y_s = \{y_s^i\}_{i=1}^{N_{tr}}$ is the corresponding labels, where x means X_{tr} and X_{te} scrambled sets, $y_{tr}^i \in \{1, 2\}$, 2 means the number of classes, and n is the number of corresponding data features. N_{tr} and N_{te} refer to the number of instances of source and target projects, and usually N_{tr} is not equal to N_{te}. Let θ_{all} denote a collection of parameters. $y(x^i) \in R^{N_{tr(te)} \times m}$ refers to the feature representations of hidden layer in the autoencoder, where m denotes the number of hidden layer neurons during the autoencoder training.

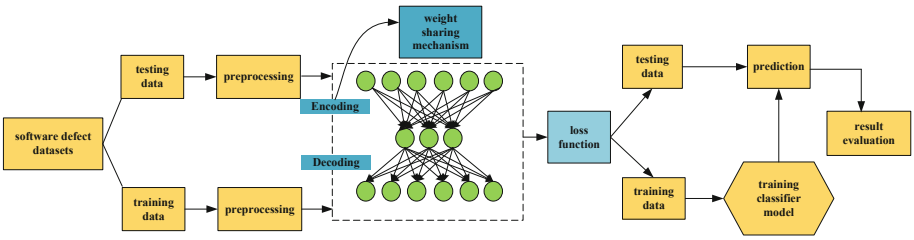

Fig. 1. The overall framework of CSSHLA. It mainly includes three parts: (1) Data normalization stage. Source data and target data can be preprocessed to the same order of magnitude. (2) Feature extraction stage. The source data features and target data features can be better converted into similar data distribution by weight sharing mechanism. (3) Classifier learning stage. The learned source features and its corresponding labels are used to learn a classifier model.

3.1 Data Normalization

We perform data normalization on these features due to the 20 basic metrics used are not the same order of magnitude. We use the 20 basic metrics [12] and min-max data normalization method [18] to convert all the values in the interval from 0 to 1 in this paper. Given feature x, its maximum and minimum values are $\max(x)$ and $\min(x)$, respectively. For each value x_i of the feature x, the normalized value P_i is computed as:

$$P_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{1}$$

3.2 Feature Extraction Model

In the feature extraction model, we use shared hidden layer autoencoder to extract features. Figure 2 shows the architecture of a basic autoencoder. Figure 3 shows the

architecture of shared hidden layer autoencoder, which is an improved version of the basic autoencoder.

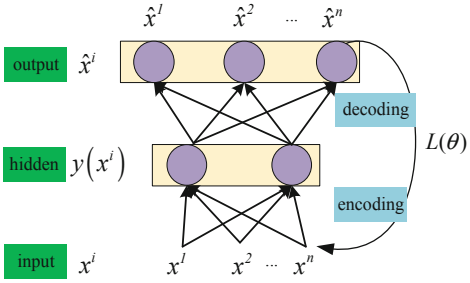

Fig. 2. Architecture of basic autoencoder. First, the original input x^i is mapped to $y(x^i)$. Then $\hat{x}^i$ tries to reconstruct x^i. The reconstruction error loss is expressed as $L(\theta)$.

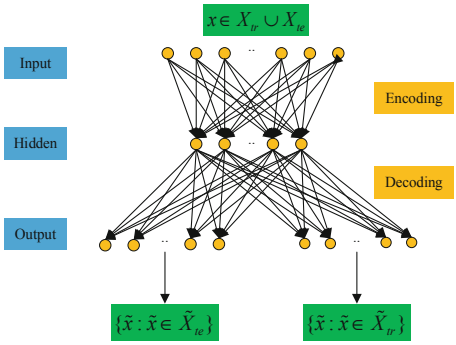

Fig. 3. Architecture of shared hidden layer autoencoder. It mainly includes two stages: (1) Encoding stage. Input data includes source data and target data, which are scrambled into the network to obtain feature representation of the hidden layer. The parameters of the input data adopt the parameter sharing mechanism in this stage. (2) Decoding stage. The feature representation of the hidden layer is decoded to get the reconstructed output data, making that the output data is equal to the input data as much as possible. In this phase, the source data and target data have different parameter settings, respectively.

Autoencoder. Autoencoder means that the output data is equal to the input data as much as possible, and it finds the common deep feature representation from input data. It mainly includes coding phase and decoding phase. Given an input data $x^i \in X_{tr}$, these two phases can be expressed as follows:

$$\text{Encoding phase}: \quad y(x^i) = f(w_1 x^i + b_1) \tag{2}$$

$$\text{Decoding phase}: \quad \hat{x}^i = f(w_2 y(x^i) + b_2) \tag{3}$$

where $f(\cdot)$ is a non-linear activation function, usually $f(\cdot)$ is sigmoid function, $w_1 \in R^{m\times n}$ and $w_2 \in R^{n\times m}$ are weight matrices, $b_1 \in R^m$ and $b_2 \in R^n$ are bias vectors. $\theta = \{w_1, b_1, w_2, b_2\}$ is included in the autoencoder network parameters, the optimization of parameters is actually to minimize the reconstruction error $L(\theta)$:

$$L(\theta) = \frac{1}{2}\sum_{x^i \in X} \left\|\hat{x}^i - x^i\right\|^2 \tag{4}$$

The implementation of minimizing $L(\theta)$ is achieved through the Adam optimizer during the autoencoder training.

Shared Hidden Layer Autoencoder. To a certain extent, it is similar to autoencoder, except that some improvements have been made in the setting of parameters. In order to solve the data distribution difference problem in CPDP, shared hidden layer autoencoder is used to obtain the advanced deep feature representation of the hidden layer by minimizing the reconstruction error loss $L(\theta_{all})$. $L(\theta_{all})$ loss consists of two parts: $L(\theta_{tr})$ and $L(\theta_{te})$. $L(\theta_{tr})$ is defined as the Euclidean distance between the input source data and the output source data. We add the label information of the source data to make the source data with the same label more compact in the decoding phase. $L(\theta_{te})$ is defined as the Euclidean distance between the input target data and the output target data. $L(\theta_{tr})$ and $L(\theta_{te})$ can be expressed as follows:

$$L(\theta_{tr}) = \frac{1}{2}\sum_{x_{tr}^i \in X_{tr}} \left\|\hat{x}_{tr}^i - x_{tr}^i\right\|^2 + \sum_{\hat{x}_{tr}^i \in X_{tr}^0} \left\|\hat{x}_{tr}^i - \bar{\hat{x}}_{tr_0}^i\right\|^2 + \sum_{\hat{x}_{tr}^i \in X_{tr}^1} \left\|\hat{x}_{tr}^i - \bar{\hat{x}}_{tr_1}^i\right\|^2 \tag{5}$$

$$L(\theta_{te}) = \frac{1}{2}\sum_{x_{te}^i \in X_{te}} \left\|\hat{x}_{te}^i - x_{te}^i\right\|^2 \tag{6}$$

where $\hat{x}_{tr}^i$ refers to the source data features obtained after the decoding phase, $\hat{x}_{te}^i$ refers to the target data features obtained after the decoding phase. X_{tr}^0 refers to all the instances in the source project are 0 and X_{tr}^1 refers to all the instances in the source project are 1. $\bar{\hat{x}}_{tr_0}^i$ and $\bar{\hat{x}}_{tr_1}^i$ are the mean values of all source project instances labeled 0 and all source project instances labeled 1 after decoding the source project data, respectively.

Combined with the above two formulas, optimizing $L(\theta_{tr})$ and $L(\theta_{te})$ two formulas at the same time, the final objective function can be expressed as:

$$L(\theta_{all}) = L(\theta_{tr}) + rL(\theta_{te}) \tag{7}$$

The network needs to optimize parameter θ_{all}: $\theta_{all} = \{w^1, b^1, w_{tr}^2, b_{tr}^2, w_{te}^2, b_{te}^2\}$. r is a regularization parameter, it can help to regularize the functional behavior of the autoencoder. The goal of this term is to make the source as similar as possible to the distribution of the target by changing the value of r.

3.3 Cost-Sensitive Softmax Classifier Model

To better learn features of minority class, cost-sensitive softmax classifier model is used to alleviate the class imbalance problem by assigning different misclassification costs to instances from different classes in the model building stage. In the trained autoencoder above, the deep feature representations of source data learned from the hidden layer are used to build a classifier.

To calculate the classification loss C, we usually measure the similarity between the real label and the predicted label by using the cross-entropy loss function, which is expressed as follows:

$$C = -\frac{1}{N_{tr}} \sum_{i=1}^{N} \sum_{c=1}^{k} \left(\left(y_s^i\right)_c * \log\left(g(x_s^i)\right)_c \right) \tag{8}$$

where N_{tr} is the number of source project instances, c refers to class of label, k is number of label class, which is set as 2 in this paper. y_s^i is ground-truth label, $g(x_s^i)$ is the final predicted label, $g(\cdot)$ is softmax activation function.

Table 1. Cost matrix for CSSHLA.

	Actual defective	Actual defect-free
Predict defective	$cost(i,i)$	$cost(i,j)$
Predict defect-free	$cost(j,i)$	$cost(j,j)$

Furthermore, we add the cost-sensitive method to the classifier, so we propose a cost-sensitive softmax classifier. The goal of cost-sensitive learning is to take the cost matrix into consideration and generate a prediction model with minimum misclassification cost. The cost matrix as shown Table 1, $cost(i,j)$ is the cost value $f(c)$ of classifying a instance from the $i{-}th$ class as the $j{-}th$ class, a correct classification will be no cost in the cost matrix, that is $cost(i,i) = 0$ and $cost(j,j) = 0$. Because more defective modules should be found, the cost of defective modules should be higher. The setting of the remaining cost value is set according to the works of [24]. $f(c)$ is defined as:

$$f(c) = \begin{cases} \dfrac{N_0}{N_1}, & c = 1 \\ 1, & c = 0 \end{cases} \tag{9}$$

Based on this, the final cost-sensitive cross-entropy loss can be defined as:

$$C = -\frac{1}{N_{tr}} \sum_{i=1}^{N} \sum_{c=1}^{k} \left(f(c) * \left(y_s^i\right)_c * \log\left(g(x_s^i)\right)_c \right) \tag{10}$$

where N_0 is the number of the defective instances, N_1 is the number of the defect-free instances. $f(c)$ means the cost of instance of class c.

4 Experiment

4.1 Datasets

In this experiment, we chose 10 projects from the PROMISE repository [12]. Table 2 lists the project name, the number of instances (#instance), the number of defective instances (#defect) and the percentage of defective instances in all instances (%defect).

Table 2. Datasets in our experiment.

Datasets	#instance	#defect	%defect
ant-1.7	745	166	22.28
camel-1.6	965	188	19.48
jedit-3.2	272	90	33.09
log4j-1.0	135	34	25.19
lucene-2.0	195	91	46.67
poi-1.5	237	141	59.49
redaktor	176	27	15.34
synapse-1.0	157	16	10.19
xalan-2.6	885	411	46.44
xerces-1.3	453	69	15.23

4.2 Evaluation Metrics

In order to evaluate the performance of proposed method, the evaluation metrics F-measure and Accuracy are widely used in SDP. As shown in Table 3, they can be defined by the confusion matrix.

Table 3. Confusion matrix.

	Predicted as defective	Predicted as defect-free
True defective	TP	FN
True defect-free	FP	TN

where *TP* is the number of defective instances that are predicted as defective, *FP* is the number of defect-free instances that are predicted as defective, *TN* is the number of defect-free instances that are predicted as defect-free, *FN* is the number of defective instances that are predicted as defect-free. So F-measure and Accuracy can be defined as:

$$precision = TP/(TP + FP),\ recall = TP/(FP + FN) \tag{11}$$

$$F\text{-}measure = (2 * precision * recall)/(precision + recall) \tag{12}$$

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{13}$$

4.3 Implementation Detail

In the training process, the CSSHLA model has 4 hidden layers and the number of nodes in each layer is 20-15-10-10-2, where 20 is the dimension of the input data, 2 is the dimension of the data that enters the softmax classifier. Each layer uses rectified linear unit (ReLU) activation function and the setting of layers is empirically obtained. CSSHLA using Adam optimizer performs the parameter optimization of during the training process. The mini-batch is set 64, and the hyper-parameter r is the following: $r \in \{0.1, 0.5, 1, 5, 10, 15\}$, the good results are obtained at $r = 10$.

4.4 Experiment Setup

In this paper, to prove the effectiveness of the proposed method CSSHLA for CPDP, we compare CSSHLA with prior CPDP methods: TCA+ [15], TDS [8], Dycom [4], LT [3] and SHLA(shared hidden layer autoencoder without cost-sensitive). We use 10 projects from PROMISE datasets as our experiments data. And we select one project from 10 projects as target, select one of the remaining nine projects and take their turn as source. We have nine possible combinations for each target project, in total, we have 90 possible CPDP combinations from 10 projects of PROMISE datasets. For example, we chose ant 1.7 as target, and our combination of CPDP is as follows: camel 1.6 - ant 1.7, jedit 3.2 - ant 1.7, camel 1.6 - ant 1.7, log4j 1.0 - ant 1.7, etc.

4.5 Experiment Result and Analysis

Through the above experimental settings, we made a comparison between CSSHLA and baselines(TCA+ , TDS, Dycom, LT, HLA). Tables 4 and 5 present the F-measure and Accuracy performance of CSSHLA compared with the five baselines, respectively. We can see that the average of F-measure of CSSHLA exceeds 5 baseline methods from Table 4, the F-measure values of CSSHLA range from 0.257 to 0.647, and CSSHLA improves F-measure results at least by 0.015 = (0.433−0.418). Table 5 shows that CSSHLA gets an average Accuracy of 0.652. Accuracy results in an improvement of at least 0.002 = (0.652−0.650).

CSSHLA can effectively solve class imbalance by using cost-sensitive learning technology compared with SHLA. The F-measure and Accuracy of CSSHLA were increased by 0.056 and 0.017, respectively. There are two reasons why our results are better than the baselines: First, to learn more about the features of minority class, we consider the influence of the class imbalance problem on the model learning by assigning different importance weights to different instances. Second, we use the advanced deep features, which are more efficient than traditional features. The results

Table 4. F-measure comparison of CSSHLA model versus 5 baselines.

Target	TDS	TCA+	Dycom	LT	SHLA	CSSHLA
ant-1.7	**0.530**	0.463	0.408	0.447	0.361	0.440
camel-1.6	0.160	**0.321**	0.070	0.260	0.233	0.288
jedit-3.2	0.444	0.510	0.415	**0.532**	0.481	0.530
log4 g-1.0	0.373	0.466	0.428	0.413	0.416	**0.487**
lucene-2.0	0.288	0.530	0.508	0.316	0.492	**0.549**
poi-1.5	0.225	0.596	0.579	0.423	0.611	**0.647**
redaktor	**0.387**	0.235	0.197	0.367	0.223	0.257
synapse-1.0	0.333	0.265	**0.336**	0.097	0.212	0.287
xalan-2.6	0.404	0.481	**0.546**	0.405	0.432	0.533
xerces-1.3	0.345	0.317	0.299	**0.360**	0.291	0.308
Average	0.349	0.418	0.379	0.365	0.377	**0.433**
Improved	0.084	0.015	0.054	0.068	0.056	–

Table 5. Accuracy comparison of CSSHLA model versus 5 baselines.

Target	TDS	TCA+	Dycom	LT	SHLA	CSSHLA
ant-1.7	0.680	0.684	0.674	0.675	0.631	**0.701**
camel-1.6	0.742	0.618	**0.769**	0.722	0.731	0.609
jedit-3.2	0.593	0.663	0.710	0.599	0.702	**0.722**
log4 g-1.0	0.715	0.657	**0.763**	0.726	0.711	0.716
lucene-2.0	0.538	0.621	0.600	0.533	0.621	**0.636**
poi-1.5	0.559	0.576	0.435	0.527	0.611	**0.616**
redaktor	0.579	0.556	0.386	**0.648**	0.361	0.494
synapse-1.0	0.761	0.641	**0.796**	0.643	0.592	0.603
xalan-2.6	0.417	0.591	0.603	0.531	0.582	**0.611**
xerces-1.3	0.714	0.627	0.764	0.757	0.810	**0.815**
Average	0.630	0.623	0.650	0.636	0.635	**0.652**
Improved	0.022	0.029	0.002	0.016	0.017	–

of F-measure and Accuracy of CSSHLA are better than baseline results. According to the evaluation metrics, the proposed method CSSHLA outperform better than baseline methods.

5 Conclusion

In this paper, we present a cost-sensitive shared hidden layer autoencoder (CSSHLA) method for cross-project defect prediction. To solve the problem of data distribution difference in CPDP, we use autoencoder with shared parameter mechanism. It can make the network adapt to source and target projects, and make the distribution of source and target projects more similar to each other by minimizing the reconstruction

error loss. Besides, we use cost sensitive learning technology to solve the class imbalance problem. CSSHLA takes into account the different misclassification costs, different weights are assigned to instances of different class. The average values of F-measure and Accuracy of CSSHLA are at least 0.015 and 0.002 better than the baseline methods, respectively. Empirical results show that CSSHLA can achieve better prediction performance than baselines.

Acknowledgements. The work described in this paper was supported by National Natural Science Foundation of China (No. 61702280), Natural Science Foundation of Jiangsu Province (No. BK20170900), National Postdoctoral Program for Innovative Talents (No. BX20180146), Scientific Research Starting Foundation for Introduced Talents in NJUPT (NUPTSF, No. NY217009), and the Postgraduate Research & Practice Innovation Program of Jiangsu Province KYCX17_0794.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
2. Boehm, B.W.: Industrial software metrics top 10 list. IEEE Softw. **4**(5), 84–85 (1987)
3. Camargo Cruz, A.E., Ochimizu, K.: Towards logistic regression models for predicting fault-prone code across software projects. In: International Symposium on Empirical Software Engineering and Measurement, pp. 460–463 (2009)
4. Liu, C., Yang, D., Xia, X., Yan, M., Zhang, X.: A two-phase transfer learning model for cross-project defect prediction. Inf. Softw. Technol. **107**, 125–136 (2019)
5. Wu, F., et al.: Intraspectrum discrimination and interspectrum correlation analysis deep network for multispectral face recognition. IEEE Trans. Cybern. 1–14 (2018)
6. Wu, F., et al.: Cross-project and within-project semisupervised software defect prediction: a unified approach. IEEE Trans. Reliab. **67**(2), 581–597 (2018)
7. Tong, H., Liu, B., Wang, S.: Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf. Softw. Technol. **96**, 94–111 (2018)
8. Herbold, S.: Training data selection for cross-project defect prediction. In: International Conference on Predictive Models in Software Engineering, p. 6 (2013)
9. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
10. Deng, J., Xia, R., Zhang, Z., Liu, Y., Schuller, B.: Introducing shared-hidden-layer autoencoders for transfer learning and their application in acoustic emotion recognition. In: International Conference on Acoustics, Speech and Signal Processing, pp. 4818–4822 (2014)
11. Deng, J., Zhang, Z., Eyben, F., Schuller, B.: Autoencoder-based unsupervised domain adaptation for speech emotion recognition. IEEE Signal Process. Lett. **21**(9), 1068–1072 (2014)
12. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: International Conference on Predictive Models in Software Engineering, p. 9 (2010)
13. Minku, L., Sarro, F., Mende, E., Ferrucci, F.: How to make best use of cross-company data for web effort estimation? In: International Symposium on Empirical Software Engineering and Measurement, pp. 1–10 (2015)

14. Zhao, L., Shang, Z., Zhao, L., Qin, A., Tang, Y.Y.: Siamese dense neural network for software defect prediction with small data. IEEE Access **7**, 7663–7677 (2019)
15. Nam, J., Pan, S.J., Kim, S.: Transfer defect learning. In: International Conference on Software Engineering, pp. 382–391 (2013)
16. Wang, S., Yao, X.: Using class imbalance learning for software defect prediction. IEEE Trans. Reliab. **62**(2), 434–443 (2013)
17. Wang, S., Liu, T., Tan, L.: Automatically learning semantic features for defect prediction. In: International Conference on Software Engineering, pp. 297–308 (2016)
18. Kotsiantis, S.B., Kanellopoulos, D., Pintelas, P.E.: Data preprocessing for supervised learning. Int. J. Comput. Sci. **1**(2), 111–117 (2006)
19. Kim, S., Zhang, H., Wu, R., Gong, L.: Dealing with noise in defect prediction. In: International Conference on Software Engineering, pp. 481–490 (2011)
20. Liu, W., Liu, S., Gu, Q., Chen, J., Chen, X., Chen, D.: Empirical studies of a two-stage data preprocessing approach for software fault prediction. IEEE Trans. Reliab. **65**(1), 38–53 (2016)
21. Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J.: Deep learning for just-in-time defect prediction. In: International Conference on Software Quality, Reliability and Security, pp. 17–26 (2015)
22. Gao, Y., Yang, C., Liang, L.: Software defect prediction based on geometric mean for subspace learning. In: Advanced Information Technology, Electronic and Automation Control Conference, pp. 225–229 (2017)
23. Yang, Y., et al.: Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. IEEE Trans. Softw. Eng. **41**(4), 331–357 (2015)
24. Li, Z., Jing, X., Wu, F., Zhu, X., Xu, B., Ying, S.: Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Autom. Softw. Eng. **25**(2), 201–245 (2018)