Cross-Project Defect Prediction Method based on Feature Distribution Alignment and Neighborhood Instance Selection

Yi Zhu[1,2], Yu Zhao[1*], Qiao Yu[1], Xiaoying Chen[1]

[1] School of Computer Science and Technology, Jiangsu Normal University, China
[2] Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, China
zhuy@jsnu.edu.cn, zhaoyu@jsnu.edu.cn, yuqiao@jsnu.edu.cn, cxy@jsnu.edu.cn

Abstract

In the practice of software project development, the developed project is a brand-new project. Defect prediction for this type of software project requires the use of other similar projects (i.e. source projects) to collect relevant data to build a defect prediction model, and make defect prediction for the project under development (i.e. target project). However, the prediction model built with the relevant data of the source project cannot achieve the ideal prediction performance when predicting the target project. The main reason is that there is a large data distribution difference between the source project and the target project. The data distribution difference is mainly in the distribution of features between projects and differences between instances. In response to the above problems, starting from both features and instances, a cross-project defect prediction method is proposed. This method first aligns the feature distribution based on the data of the existing target project and the source project data. Then, it selects the labeled instance that is similar to the unlabeled instance in the target project, and finally builds a defect prediction model based on the selected source project instances. Cross-project defect prediction experiments were carried out on the Relink datasets and the Promise datasets. Compared with the classic instance-based cross-project defect prediction method, significant improvements have been made in F-measure and AUC; compared with the prediction of within project defect prediction, it has achieved comparable performance.

Keywords: Cross-project defect prediction, Feature distribution alignment, Instance selection

1 Introduction

The importance and dependence of software in many application fields are increasing, so it is more and more important to ensure the reliability of software [1-5]. Predicting defects in a software project is very important to the software development process because the later errors in the software are discovered, the greater the cost of fixing the errors. The purpose of software defect prediction [6-12] is to help software developers find software defects in the early stages of development to allocate software testing resources reasonably to improve software reliability. The rapid development of machine learning technology allows software testers to build software defect prediction models based on existing data to focus on testing those classes or files that may have defects based on the prediction results [13]. Machine learning technology has been successfully applied in Within Project Defect Prediction (WPDP). Nevertheless, the actual software project under development is often a brand-new software project. In this case, WPDP may not be applicable because we do not have enough historical data to construct a defect prediction model.

In response to this situation, researchers have proposed the Cross-project Defect Prediction method [14-21]. The Cross-project Defect Prediction (CPDP) method is used to train the model based on the labeled data of other similar software projects (i.e. source projects) and to predict the defects of the software projects currently under development (i.e. target projects). However, due to the large distribution differences between the source project and the target project in features and instances, this leads to the failure of the defect prediction model built by the source project to achieve a good prediction performance on the target project. Therefore, how to reduce the difference in data distribution between the source project and the target project has become an urgent problem in the field of software defect prediction.

In the field of computer vision, image recognition and classification also have the problem of the difference in the feature distribution between the source domain and the target domain. There are already many methods to reduce the difference in feature distribution between image domains and improve the accuracy of image recognition [22-23], Sun et al. [24] proposed a feature distribution alignment method, which transformed and aligned the statistical features of image data, and then used the aligned data to construct a traditional machine learning classifier, which effectively improved the accuracy of image classification. The method has also achieved excellent results in the field of deep learning [25].

This paper proposes a cross-project defect prediction method based on feature distribution alignment and neighborhood instance selection (FDAIS). This method starts from the perspective of feature distribution alignment and neighborhood instance selection. On the one hand, it tries to solve the problem of feature distribution differences between projects. On the other hand, it tries to find the source project instances similar to the target project instances. Specifically, the existing methods on how to reduce the difference between the source project and the target project basically select source projects with similar feature distribution vectors from a large

*Corresponding Author: Yu Zhao; E-mail: zhaoyu@jsnu.edu.cn
DOI: 10.53106/160792642022072304011

number of source project datasets, and then use the selected source project data to build the model. Another solution is to first select a similar source project dataset, then select similar instances in the dataset, and finally build the model with the selected instances. However, the existing methods to measure the similarity between projects only rely on the statistical characteristics of the value distribution of the metrics. The similarity of the statistical characteristics does not accurately indicate that the projects are similar, that is to say, there are still differences in data distribution between the selected source project and the target project.

Furthermore, our method can preserve all feature information and data distribution and make full use of it. For example, existing feature selection methods for data feature processing may directly remove redundant features or irrelevant features that may exist in the dataset, which may lose important information. Because the feature selection method is judged according to the source project, some features are judged to be effective for classification according to the source project, but the actual feature that is effective for the classification of the target project of unknown label may be the feature discarded by the feature selection method.

From the perspective of feature distribution alignment, we perform second-order feature alignment on the source project and the target project, and then select a subset of neighborhood instances that are strongly related to the target project in the source project after feature alignment. Finally, a defect prediction model is constructed based on the subset of neighborhood instances. Compared with the classic instance selection method, our method achieved better and more stable performance in experiments on Relink and Promise datasets.

The main contributions of this paper are as follows:

(1) We propose a cross-project defect prediction method named FDAIS. This method can effectively reduce the difference in data distribution between the source project and the target project by second-order alignment of the features of the source project and the target project;

(2) Based on the empirical research on the Relink and Promise datasets, our method has better performance on a variety of indicators and more stable performance than the classic Burak filters; compared with WPDP, our method achieved comparable performance in the prediction of defect class.

The second section of the paper introduces the background knowledge and related works. The third section introduces specific implementation details of the cross-project defect prediction method based on feature distribution alignment and neighborhood instance selection. The fourth section introduces the empirical research of the paper. The fifth section analyzes the experimental results in detail, and finally summarizes the full text and the next work in the sixth section.

2 Related Work

In recent years, CPDP has attracted extensive attention from software testing researchers. There are many problems worth studying in CPDP research. At present, researchers mainly focus on three aspects: training data, data features and training models.

For the training data problem in CPDP research, many researchers have proposed methods of training data selection to select training data [26-28]. These methods select appropriate training data from the perspective of project, instance, and simultaneous selection of both. For example, He [26] et al. proposed a two-stage screening method. In the first stage, starting from a coarse-grained perspective, first select the source project with the most similar distribution to the target project, and then select the most similar instance set from the selected source projects in the second stage.

In view of the possible redundant features in the CPDP dataset, researchers have proposed some feature selection methods to process the data features before training the CPDP model. For example, Yu et al. [29] analyzed the importance of features and instances in cross-project defect prediction methods, conducted a lot of empirical research on feature selection methods [30], and proposed a feature matching and transferring cross-project defect prediction method [31].

In view of the problem that the current CPDP training data is large in quantity and belongs to different source projects, the researchers studied the applicability of the ensemble learning algorithm to CPDP from the perspective of training models [32-35]. For example, Zhang et al. [35] evaluated seven ensemble algorithms on the large-scale datasets. When building a defect prediction model, the ensemble algorithm is iteratively trained using a set of labeled instances from multiple source projects. Experimental results show that using bagging and boosting algorithms combined with appropriate classification models can improve the performance of CPDP.

3 Proposed Method

Turhan [36] et.al. believe that the difference in data distribution between the source project and the target project is mainly caused by the drift of the covariate, and the drift of the covariate causes the difference in the feature distribution between the projects. Therefore, we introduced a method of covariance alignment [24] by aligning the covariance matrix of the source project and the target project. This was done to align the feature distribution of the source project and the target project. Then, select the neighborhood instance in the source project dataset after the feature distribution is aligned. Figure 1 shows the framework of the proposed method. Initially, it receives an unlabeled target project and multiple labeled source projects as input. The feature distribution alignment module aligns multiple source projects with the target project for covariance alignment, and then the neighborhood instance selection module selects instances similar to the target project in the aligned source projects sets. Finally, the training model module receives the selected source project instance as training data for model training and returns the trained classifier.

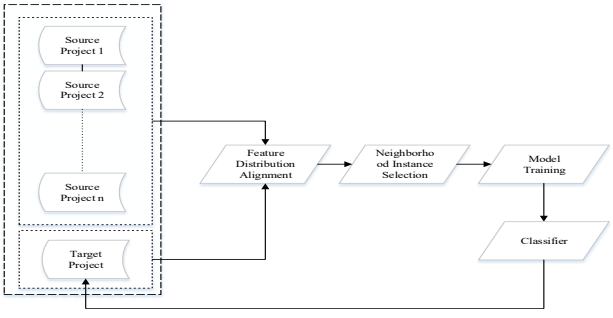


Figure 1. FDAIS method framework

Next, we introduce the definition of related symbols.

3.1 Related Definition

D_S indicates the labeled source project data and D_T indicates the unlabeled target project data; D_S and D_T have the same metric, C_S represents the covariance matrix of the source project, and C_T represents the covariance matrix of the target project. I^S represents all instances in the source project and I^T represents all instances in the target project. Every instance $I_i=\{f_1^i, f_2^i \ldots f_m^i, y_i\}$ indicates that the *i*-th instance has m attributes f_j^i and defect indicators y_i, f_j^i represents the *j*-th attribute of the *i*-th instance in a software project, defect indicators $y_i \in \{Y, N\}$, $y_i=N$ indicates that the *i*-th instance in the project has no defects, and $y_i=Y$ indicates that the *i*-th instance in the project has defects.

3.2 Feature Distribution Alignment

As mentioned above, we introduced a covariance alignment method, which aligns the feature distribution of the source project and target project data by aligning the covariance matrix of the source project and the target project.

Specifically, our goal is to minimize the distance of the covariance matrix between the source project and the target project. We apply linear transformation *A* to the source project features, then use the Frobenius norm as the distance metric between the matrices. The optimization goal can be written as follows:

$$\min_A \left\| C_{\hat{S}} - C_T \right\|_F^2 = \min_A \left\| A^T C_S A - C_T \right\|_F^2 \quad (1)$$

Where $C_{\hat{S}}$ is the covariance of $D_S A$ and $\|\cdot\|_F^2$ represents the Frobenius norm of the matrix. According to the solution process in [14], we can get the optimal solution of *A*:

$$\begin{aligned} A^* &= U_S E \\ &= (U_S \Sigma_S^{\frac{+1}{2}} U_S{}^T)(U_{T[1:r]} \Sigma_{T[1:r]}^{\frac{1}{2}} U_{T[1:r]}{}^T) \end{aligned} \quad (2)$$

We did not use the optimal solution for calculation directly. In fact, the optimal solution is the product of two parts. The first part is actually processing the features of the source project, that is, eliminating the correlation between the features of the source project, and the second part is to fill the feature correlation of the target project into the source project. We use Algorithm 1 to complete the above functions. The input of Algorithm 1 is the source project data D_S and the target project data D_T. The third and fourth steps are to calculate the covariance matrix of the source project and the target project respectively. The fifth step is to eliminate the correlation between the features of the source project, and the sixth step is to fill the feature correlation of the target project into the source project. Finally, the adjusted source project data D_S^* is output.

Algorithm 1. Feature distribution processing algorithm

(1) Input: source project D_S, target project D_T
(2) Output: adjusted source project D_S^*
(3) $C_S = COV(D_S)$
(4) $C_T = COV(D_T)$
(5) $D_S = D_S * C_S^{-1/2}$
(6) $D_S^* = D_S * C_T^{1/2}$

3.3 Neighborhood Instance Selection

Neighborhood instance selection is to select the *k* instances that are most similar to each instance in the target project from the source project after the feature distribution is aligned, similar to the *knn* algorithm. If there are *N* test instances in the target project, for each instance, we add the *k* closest source project instances in the neighborhood of the instance to the training instance set, similar to the idea of Burak filters, so that we finally get *N*k* training instances. We selected 5 source project instances with the highest similarity for each instance in the target project, and calculate the degree of similarity between instances using the following formula:

$$similarity(I_i^S, I_j^T) = \sqrt{\sum_{k=1}^{m} (f_k^i - f_k^j)^2} \quad (3)$$

4 Experimental Design

This section conducts an empirical study on the effectiveness of the FDAIS method. We first introduce the datasets, performance evaluation indicators, classification models, and experimental settings used in the empirical research.

4.1 Datasets

This paper uses the defect data of some software projects in the Relink [37-38] datasets and the Promise datasets [39] for experiments. These datasets were widely used by researchers in this field. The basic information of the datasets is shown in Table 1. The first three projects belong to the Relink datasets, and the last ten projects belong to the Promise datasets.

Table 1. Datasets

Projects	Modules	Features	Defects%
Apache	194	26	51%
Safe	56	26	39%
Zxing	399	26	30%
berek	30	21	43%
ckjm	7	21	43%
pdftranslator	25	21	32%
skarbonka	30	21	50%
szybkafucha	18	21	39%
workflow	29	21	34%
ivy-1.1	111	21	57%
jedit-3.2	272	21	33%
log4j-1.1	109	21	34%
velocity-1.6	229	21	34%

4.2 Evaluation Indicator

Cross-project defect prediction is essentially a classification problem, so we use commonly used classification evaluation indicators to evaluate the performance of the method.

There are four types in the classification task: TP (True Positive) means that a defective software module is predicted to be defective; FP (False Positive) means that a non-defective software module is predicted to be defective; TN (True Negative) means that a non-defective software module is predicted to be non-defective; FN (False Negative) means that a defective software module is predicted to be non-defective. Combining these results yields some commonly used performance metrics. Precision is the ratio of true defects in all predicted defective test data, and recall is the ratio of true defects in all actual defective test data. The calculation method is shown in Equations 4 and 5.

$$precision = \frac{TP}{TP + FP} \tag{4}$$

$$recall = \frac{TP}{TP + FN} \tag{5}$$

The F-measure comprehensively considers Recall and Precision, which can fully reflect the actual performance of the method. Therefore, we mainly use the F-measure to evaluate the performance of each method, but we will also refer to other indicators in the specific analysis.

$$F\text{-}measure = \frac{2 * precision * recall}{precision + recall} \tag{6}$$

Since the experiment is carried out on the Weka platform, the indicator value can be obtained directly from the experimental results of the Weka platform.

4.3 Classification Models and Experimental Settings

This paper chooses the logistic regression model as the training model. The model has been automatically integrated on the Weka platform, and the relevant parameters are set as the default parameters of the Weka platform when the model is used. When performing feature distribution alignment and neighborhood instance selection, we use python language and related data processing packages.

5 Experimental Results and Analysis

We explore the experimental performance of the FDAIS method and the main factors affecting the performance of the FDAIS method from the following three questions.

RQ1: Can the FDAIS method effectively reduce the difference in data distribution between projects and achieve good performance? How does the performance compare with the classic instance selection method Burak filters?

RQ2: Compared with WPDP, what can be improved in the FDAIS method?

RQ3: How do different classification models and datasets with different defect rates affect the experimental performance of the FDAIS method?

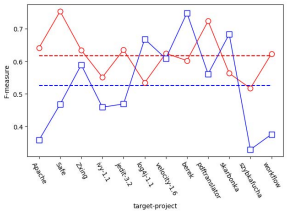


Figure 2. Overall F-measure

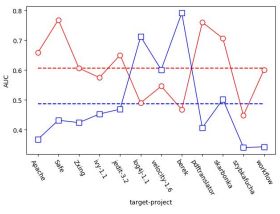


Figure 3. Overall AUC

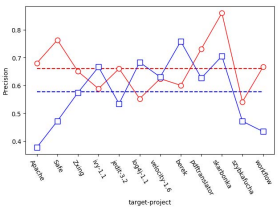


Figure 4. Overall Precision

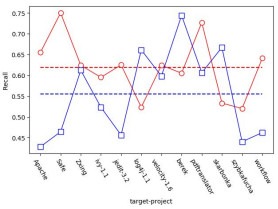


Figure 5. Overall Recall

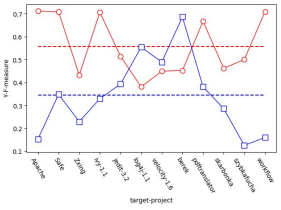


Figure 6. Class-Y F-measure

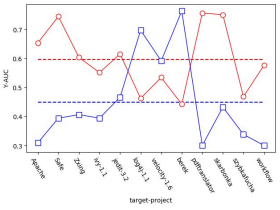


Figure 7. Class-Y AUC

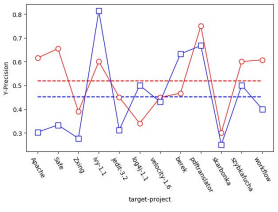


Figure 8. Class-Y Precision

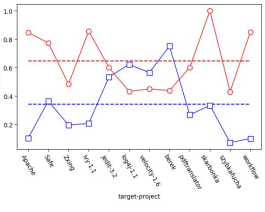


Figure 9. Class-Y Recall

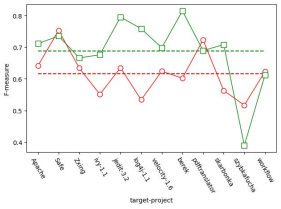


Figure 10. Overall F-measure

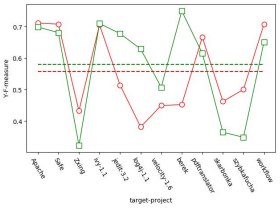


Figure 11. Class-Y F-measure

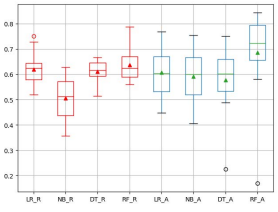


Figure 12. Impact on FDAIS using different basic classifier

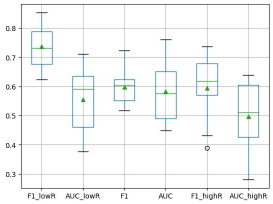


Figure 13. Overall performance on different defect rate datasets

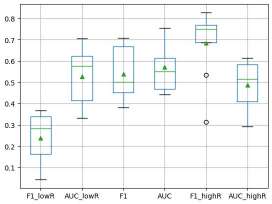


Figure 14. Performance of class-Y on different defect rate datasets

5.1 Answer to RQ1

For RQ1, we made the experimental results into the line graphs shown in Figure 2 to Figure 9. The line chart not only shows the indicator F-measure, but we also show the indicator values of Precision and Recall. In addition, because the AUC takes into account the change of the threshold, it is more comprehensive to use AUC value to evaluate the performance of the model, so we also plot the value of the AUC indicator. As shown in Figure 2 to Figure 9, the red solid line and red circle indicate the performance of FDAIS on each project, and the blue solid line and blue square indicate the performance of the Burak filters on each project. The red dotted line represents the average performance of FDAIS on all projects, and the blue dotted line represents the average performance of Burak filters on all projects.

From the performance on the Relink datasets and Promise datasets in Figure 2 to Figure 5, we can see that the overall performance of FDAIS is good. The F-measure, Precision, Recall, and AUC are between 0.5 and 0.8, especially the Relink datasets which only have three projects. FDAIS has shown excellent results with F-measure, Precision, Recall, and AUC between 0.6 and 0.8. The performance on the Promise datasets is also very good, so FDAIS can effectively reduce the difference between projects and achieve good performance.

Herbold [40] et al. conducted large-scale experimental investigations on all reproducible CPDP methods and found that the performance of the classic Burak filters method surpassed most of the current more advanced methods. Therefore, we compared the performance of the FDAIS with the Burak filters in an all-round way. We compare from two aspects. On the one hand, we choose the overall classification performance (i.e. Weka's weighted result). The overall classification performance is to measure the total classification ability of the method for defective and non-defective instances, including predicting defective instances as non-defective classes and predicting non-defective instances as defective classes. The specific indicators values are shown in Figure 2 to Figure 5. On the other hand, we only compare the performance of the two methods on the positive class (i.e. Weka's Y class), because the defect of a software project is more important in practical application. If a non-defective instance is predicted to be defective, the impact may not be significant, and it will only cause a waste of part of the test resources. If the defective instance is predicted to be non-defective, the defective instance will always be hidden in the software project in this case, and it will eventually lead to serious problems. Therefore, we also made a comparison in the defective class, and the specific indicators values are shown in Figure 6 to Figure 9.

In terms of overall classification, Figure 2 to Figure 5 show that FDAIS is generally better than Burak filters in various indicators. From the F-measure and AUC in Figure 2 to Figure 3, 9 groups are significantly better than Burak filters, and 3 groups are significantly inferior to Burak filters. The average value of FDAIS is about 0.1 higher than that of traditional methods. We analyze the three projects of log4j-1.1, berek, and skarbonka with low F-measure. We found that log4j-1.1 and berek whose Precision and Recall were lower than Burak filters, while skarbonka is higher than Burak filters in Precision and lower than Burak filters in Recall. This shows that the performance of FDAIS on log4j-1.1 and berek is indeed worse than Burak filters. On skarbonka, since the F-measure is the harmonic mean of precision and recall, it is affected by the lower Recall, which leads to the decrease of F-measure. In general, FDAIS is much more stable than Burak filters. The broken line statistical chart from Figure 2 to Figure 5 shows that the performance of Burak filters fluctuates greatly, ranging from 0.2 to 0.8, while FDAIS is basically between 0.5 and 0.8. This is another advantage compared to Burak filters.

When only comparing the positive class, Figure 6 to Figure 7 show the F-measure and AUC values of the FDAIS and Burak filters on the defective class. It can be seen that FDAIS is better than the Burak filters as a whole. Similarly, there are 9 groups of indicator values that are significantly better than the Burak filters, and 3 groups are significantly worse than the Burak filters; on average, FDAIS is about 0.2 higher than the Burak filters. We analyze the three projects of log4j-1.1, berek, and velocity-1.6 with low F-measure. We found that log4j-1.1 and berek whose Precision and Recall were lower than Burak filters, while velocity-1.6 is higher than Burak filters in Precision and lower than Burak filters in Recall. This also shows that the performance of FDAIS in judging defective classes on log4j-1.1 and berek is indeed worse than the Burak filters and the F-measure is reduced by the lower Recall on the velocity-1.6. Therefore, on the whole, FDAIS has better performance on defective classes, that is, the ability to identify defective instances is stronger and the method is more reliable.

Therefore, we give a brief answer to RQ1, FDAIS method can effectively reduce the difference of data distribution between projects and achieve good performance. Compared with the classical instance selection method Burak filter, our method has advantages in multiple metrics in terms of comprehensive performance and performance on defect classes.

5.2 Answer to RQ2

For RQ2, in order to explore the performance comparison between FDAIS and WPDP, we use the Weka software platform, select the logistic regression model, and use the default parameters of Weka to carry out WPDP experiments on all software projects.

Figure 10 and Figure 11 are the comparison of FDAIS and WPDP on F-measure. Figure 10 is the overall classification performance, and Figure 11 is the prediction performance on the defect class. The red solid line and red circle represent the F-measure of FDAIS on each dataset, and the green solid line and green square represent the F-measure of WPDP. The red dotted line represents the average F-measure of FDAIS on all projects, and the green dotted line represents the average F-measure of WPDP on all projects.

According to Figure 11, it can be found that the prediction performance of FDAIS on defect instances is very close to that of WPDP. The average difference of F-measure between them is no more than 0.02, The performance of each project is also very close. However, in Figure 10, the average of F-Measure on the whole is about 0.08 different, and the F-Measure of WPDP is also high in each project. This shows that the WPDP can not only correctly identify the defect instances, but also has a better prediction effect on the non-defect instances.

Therefore, compared with WPDP, although the performance of FDAIS on defect instances can be compared with that of WPDP, there is still some misjudgment on non-defect instances, which is worth improving.

5.3 Answer to RQ3

RQ3 is to explore the factors that affect the experimental performance of the FDAIS method. In general, different classification models tend to have an impact on the experimental performance, so we use different models for the experiment. In addition to the classification model we used at the beginning, we also used the probability-based classification model NB (Naive Bayes), the rule-based classifier DT (Decision Tree), and the ensemble learning classifier RF (Random Forest). Figure 12 shows the experimental results.

Here we use two common indicators to evaluate different classifiers, one is Recall, the other is AUC. In Figure 12, the four red box graphs on the left represent the Recall values predicted by the four classifiers on the datasets in Table 1, and the four on the right represent the AUC values. The horizontal line in each box graph represents the median of experimental results obtained using the classifier, the triangle represents the average of experimental results, and the box represents the data distribution of experimental results. As can be seen from Figure 12, the Recall obtained by using NB is relatively low, while the Recall obtained by using LR and DT is similar, and the Recall obtained by using RF is generally better. In terms of AUC, the median AUC obtained by using LR, NB, and DT is the same. The average value and overall distribution of AUC obtained by using LR are slightly better, but the overall distribution of AUC obtained by using RF is much better than the other three models, with the highest mean value and median value.

Based on the above analysis, different types of classifiers will have a certain impact on the performance of FDAIS method, and ensemble learning classifier can significantly improve the performance of the method.

In general, the performance of CPDP is also affected by imbalanced datasets. At the beginning of the experiment, we chose some projects with balanced defect rate in Promise datasets, but there may be class imbalance in practice. To further determine the scope of our method, we performed an experimental comparison on two other projects with extreme defect rates in the Promise datasets. The first set of low defect rates datasets is shown in Table 2. The defect rate of each project is between 0% and 30%. The second set of high defect rate datasets is shown in Table 3. The defect rate of each project is between 60% and 100%. We performed experiments on low defect rate datasets and high defect rate datasets using the FDAIS method. The box plots in Figure 13 and Figure 14 show the experimental results of FDAIS on different defect rate datasets. Figure 13 shows the overall classification performance, while Figure 14 shows the classification performance only on the defect class.

F1_lowR and AUC_lowR represent the F-measure and AUC values on the datasets shown in Table 2. F1 and AUC represent the performance of the FDAIS method on the datasets shown in Table 1, which are obtained under the condition of a balanced defect ratio. F1_highR and AUC_highR represent the F-measure and AUC values on the datasets shown in Table 3. The green horizontal lines in the box plot represent the median of the group's experimental results, and the green triangles represent the average of the group’s experimental results.

From the overall classification results shown in Figure 13, the F-measure of the FDAIS method is the highest on datasets with low defect rates, and it is better than the other two conditions whether it is the distribution of F-measure, the mean value of F-measure, or the median value of F-measure. However, the AUC is very low. The reason for our analysis is that the defect rate of the datasets is very low which causes the model to classify most of the non-defect instances correctly, while a small number of defect instances are not correctly classified, resulting in the situation of high F-measure and low AUC. The mean value, median value, and distribution range of F-measure in high defect rate datasets are similar to those in defect rate balanced datasets, but the AUC on high defect rate datasets is poor. We analyze that at this time, most of the defective instances are successfully classified, while only a small part of the non-defect instances in the datasets are successfully classified. However, it is slightly better than the performance of the low defect rate datasets. This shows that the FDAIS method is more suitable for a balanced defect ratio of the datasets. Our method may not be suitable for low defect rate datasets. For high defect rate datasets, FDAIS also cannot achieve satisfactory results.

This conclusion can also be proved in the performance of the defect class in Figure 14. The value of F1_lowR is basically lower than 0.5, indicating that the model's Precision and Recall on the defect class are very low. Moreover, the F-measure and AUC on the high defect rate datasets are not commensurate, indicating that the model also has a large degree of misjudgment. Therefore, we suggest that FDAIS should be used when the defect rate is relatively balanced, that is, a dataset with a defect rate of 30%-60% can get better results.

As a result, we make a summary answer to RQ3. Different types of classifiers will have a certain impact on the

performance of FDAIS method, and ensemble learning classifier can significantly improve the performance of the method. Datasets with different defect rates will affect the experimental performance of the FDAIS method, and the FDAIS method is more suitable for datasets with balanced defect rates.

Table 2. Low ratio promise datasets

Projects	Modules	Features	Defects%
arc	234	21	12%
camel-1.4	872	21	17%
ivy-2.0	352	21	11%
jedit-4.2	367	21	13%
poi-2.0	314	21	12%
prop-6	660	21	10%
redaktor	176	21	15%
tomcat	858	21	9%
xalan-2.4	723	21	15%
xerces-1.2	440	21	16%

Table 3. High ratio promise datasets

Projects	Modules	Features	Defects%
log4j-1.2	205	21	92%
lucene-2.4	340	21	60%
pbeans1	26	21	77%
poi-2.5	385	21	64%
sklebagd	20	21	60%
szybkafucha	25	21	56%
velocity-1.4	196	21	75%
wspomaganiepi	18	21	67%
xalan-2.7	909	21	99%
xerces-1.4	588	21	74%

6 Conclusion

This paper proposes a cross-project defect prediction method based on feature distribution alignment and neighborhood instance selection. This method combines feature distribution alignment and neighborhood instance selection. From the perspective of project covariate drift, first align the second-order statistics between the source project and the target project, then select a subset of neighborhood instances that are strongly related to the target project in the aligned source project. Finally, build a defect prediction model based on the neighborhood instance subset. Empirical research shows that our method can effectively reduce the difference in data distribution between the source project and the target project. Compared with the classic Burak filters, our method has achieved better and more stable performance on the Relink and Promise datasets. Compared with the WPDP method, our method has achieved considerable performance on the defect classes of the Relink and Promise datasets.

There are still many next steps worth discussing in this article. First of all, the method uses the traditional Euclidean distance to measure the similarity between instances when selecting instances after feature alignment. In fact, Euclidean distance may not necessarily get similar instances in this case, we can try other distance measures later. Secondly, in the experiment, we selected software projects with two datasets, and our method can be further extended to other suitable datasets in the later period; at the same time, it can be seen from the experimental results that not all methods can achieve excellent results on all datasets. Our method can achieve better results on the class-balanced datasets, and it can also predict the defect class very well. But in fact, unbalanced datasets are more common, so designing appropriate methods for different datasets is also a problem worthy of research, and we will continue to design new methods for exploration in the later period. Finally, different classifiers have a certain impact on the method. Ensemble learning can improve the performance of the method a lot. In the later stage, the effect of ensemble learning on the performance of cross-project defect prediction methods can be studied.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No. 62077029); the National Natural Science Foundation of China Youth Project (Grant No. 61902161); the Open Project Fund of Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology (Grant No. NJ2020022); the Future Network Scientific Research Fund Project (Grant No. FNSRFP-2021-YB-32); the Applied Basic Research Program of Xuzhou (Grant No. KC19004); the Graduate Science Research Innovation Program of Jiangsu Province (Grant No. KYCX20_2384 and KYCX20_2380).

References

[1] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, M. F. Siok, Recent Catastrophic Accidents: Investigating how Software was Responsible, *Proceedings of the Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement*, Singapore, 2010, pp. 14-22.

[2] W. E. Wong, X. Li, P. A. Laplante, Be more Familiar with our Enemies and Pave the Way Forward: A Review of the Roles Bugs Played in Software Failures, *Journal of Systems and Software*, Vol. 133, pp. 68-94, November, 2017.

[3] Y. Zhu, Z. Q. Huang, H. Zhou. Modeling and Verification of Web Services Composition based on Model Transformation, *Software: Practice and Experience*, Vol. 47, No. 5, pp. 709-730, May, 2017.

[4] Y. T. Chang, W Gunarathne, T. K. Shih, Deep Learning Approaches for Dynamic Object Understanding and Defect Detection, *Journal of Internet Technology*, Vol. 21, No. 3, pp. 783-790, May, 2020.

[5] B. Khan, R. Naseem, M. Binsawad, M. Khan, A. Ahmad, Software Cost Estimation Using Flower Pollination Algorithm, *Journal of Internet Technology*, Vol. 21, No. 5, pp. 1243-1251, September, 2020.

[6] P. Mahesha, D. Gupta, Performance of Genetic Programming-based Software Defect Prediction Models, *International Journal of Performability Engineering*, Vol. 17, No. 9, pp. 787-795, September, 2021.

[7] X. Chen, Q. Gu, W. S. Liu, S. L. Liu, C. Ni, Survey of Static Software Defect Prediction, *Journal of Software,* Vol. 27, No. 1, pp. 1-25, January, 2016.

[8] Q. Wang, S. J. Wu, M. S. Li, Software Defect Prediction, *Journal of Software,* Vol. 19, No. 7, pp. 1565-1580, July, 2008.

[9] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A Systematic Literature Review on Fault Prediction Performance in Software Engineering, *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, pp. 1276-1304, November-December, 2012.

[10] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, N. Ziviani, Understanding Machine Learning Software Defect Predictions, *Automated Software Engineering*, Vol. 27, No. 3-4, pp. 369-392, December, 2020.

[11] S. K. Pandey, D. Rathee, A. K. Tripathi, Software Defect Prediction Using K-PCA and Various Kernel-Based Extreme Learning Machine: An Empirical Study, *IET Software*, Vol. 14, No. 7, pp. 768-782, December, 2020.

[12] D. Li, W. E. Wong, W. Wang, Y. Yao, M. Chau, Detection and Mitigation of Label-Flipping Attacks in Federated Learning Systems with KPCA and K-Means, *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, Yinchuan, China, 2021, pp. 551-559.

[13] G. Xie, S. Xie, X. Peng, Z. Li, Prediction of Number of Software Defects based on SMOTE, *International Journal of Performability Engineering*, Vol. 17, No. 1, pp. 123-134, January, 2021.

[14] Y. Li, W. E. Wong, S. Y. Lee, F. Wotawa, Using Tri-relation Networks for Effective Software Fault-proneness Prediction, *IEEE Access*, Vol. 7, pp. 63066-63080, May, 2019.

[15] X. Chen, L. P. Wang, Q. Gu, Z. Wang, C. Ni, W. S. Liu, Q. P. Wang, A Survey on Cross-project Software Defect Prediction Methods, *Chinese Journal of Computers*, Vol. 41, No. 1, pp. 254-274, January, 2018.

[16] L. N. Gong, S. L. Jiang, L. Jiang, Research Progress of Software Defect Prediction, *Journal of Software,* Vol. 30, No. 10, pp. 3090-3114, October, 2019.

[17] S. Hosseini, B. Turhan, D. Gunarathna, A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction, *IEEE Transactions on Software Engineering*, Vol. 45, No. 2, pp. 111-147, February, 2019.

[18] X. Xia, D. Lo, S. J. Pan, N. Nagappan, X. Wang, HYDRA: Massively Compositional Model for Cross-Project Defect Prediction, *IEEE Transactions on Software Engineering*, Vol. 42, No. 10, pp. 977-998, October, 2016.

[19] Y. Li, Z. D. Liu, H. J. Zhang, Review on Cross-project Software Defects Prediction Methods, *Computer Technology and Development*, Vol. 30, No. 3, pp. 98-103, March, 2020.

[20] Y. Zhao, Y. Zhu, Q. Yu, X. Y. Chen, Cross-Project Defect Prediction Method Based on Manifold Feature Transformation, *Future Internet,* Vol. 13, No. 8, 216, pp. 1-17, August, 2021.

[21] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, B. Xu, How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction, *ACM Transactions on Software Engineering and Methodology*, Vol. 27, No. 1, pp. 1-51, January, 2018.

[22] B. Fernando, A. Habrard, M. Sebban, T. Tuytelaars, Unsupervised Visual Domain Adaptation Using Subspace Alignment, *2013 IEEE International Conference on Computer Vision*, Sydney, Australia, 2013, pp. 2960-2967.

[23] B. Sun, K. Saenko, Subspace Distribution Alignment for Unsupervised Domain Adaptation, *British Machine Vision Conference*, Swansea, UK, 2015, pp. 1-10.

[24] B. Sun, J. Feng, K. Saenko, Return of Frustratingly Easy Domain Adaptation, *2016 AAAI Conference on Artificial Intelligence*, Phoenix, AZ, US, 2016, pp. 2058-2065.

[25] B. Sun, K. Saenko, Deep Coral: Correlation Alignment for Deep Domain Adaptation, *European Conference on Computer Vision*, Amsterdam, The Netherlands, 2016, pp. 443-450.

[26] Z. He, F. Peters, T. Menzies, Y. Yang, Learning From Open-source Projects: An Empirical Study on Defect Prediction, *Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement*, Baltimore, MD, USA, 2013, pp. 45-54.

[27] F. Peters, T. Menzies, A. Marcus, Better Cross Company Defect Prediction, *2013 10th Working Conference on Mining Software Repositories*, San Francisco, CA, USA, 2013, pp. 409-418.

[28] P. He, B. Li, X. Liu, J. Chen, Y. T. Ma, An Empirical Study on Software Defect Prediction with a Simplified Metric Set, *Information and Software Technology*, Vol. 59, pp. 170-190, March, 2015.

[29] Q. Yu, S. Jiang, Y. Zhang, A Feature Matching and Transfer Approach for Cross-company Defect Prediction, *Journal of Systems and Software*, Vol. 132, pp. 366-378, October, 2017.

[30] Q. Yu, S. Jiang, J. Qian, L. Bo, L. Jiang, G. Zhang, Process Metrics for Software Defect Prediction in Object-oriented Programs, *IET Software*, Vol. 14, No. 3, pp. 283-292, June, 2020.

[31] Q. Yu, J. Qian, S. Jiang, Z. Wu, G. Zhang, An Empirical Study on the Effectiveness of Feature Selection for Cross-project Defect Prediction, *IEEE Access*, Vol. 7, pp. 35710-35718, January, 2019.

[32] G. Adomavicius, A. Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-art and Possible Extensions, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 6, pp. 734-749, June, 2005.

[33] Z. Sun, J. Li, H. Sun, L. He, CFPS: Collaborative Filtering Based Source Projects Selection for Cross-project Defect Prediction, *Applied Soft Computing*, Vol. 99, Article No. 106940, February, 2021.

[34] J. Chen, K. Hu, Y. Yang, Y. Liu, Q. Xuan, Collective Transfer Learning for Defect Prediction, *Neurocomputing*, Vol. 416, pp. 103-116, November, 2020.

[35] Y. Zhang, D. Lo, X. Xia, J. Sun, An Empirical Study of Classifier Combination for Cross-project Defect Prediction, *IEEE Computer Software & Applications Conference*, Taichung, Taiwan, 2015, pp. 264-269.

[36] B. Turhan, On the Dataset Shift Problem in Software Engineering Prediction Models, *Empirical Software Engineering*, Vol. 17, No. 1-2, pp. 62-74, February, 2012.

[37] R. Wu, H. Zhang, S. Kim, S. C. Cheung, ReLink: Recovering Links Between Bugs and Changes, *19th ACM SIGSOFT Symposium and the 13th European*

Conference on Foundations of Software Engineering, Szeged, Hungary, 2011, pp. 15-25.

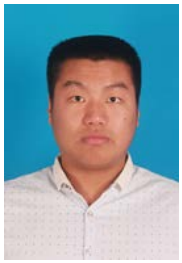
[38] M. D'Ambros, M. Lanza R. Robbes, Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison, *Empirical Software Engineering*, Vol. 17, No. 4-5, pp. 531-577, August, 2012.

[39] M. Jureczko, L. Madeyski, Towards Identifying Software Project Clusters with regard to Defect Prediction, *2010 International Conference on Predictive Models in Software Engineering*, Timisoara, Romania, 2010, pp. 1-10.

[40] S. Herbold, A. Trautsch J. Grabowski, A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches, *IEEE Transactions on Software Engineering*, Vol. 44, No. 9, pp. 811-833, September, 2018.

Biographies

Yi Zhu received his Ph.D. degree in computer science and technology from Nanjing University of Aeronautics and Astronautics. He is currently the Professor in the School of Computer Science and Technology, Jiangsu Normal University. His current research interests include software engineering, software defect prediction, and formal methods and applications.



Yu Zhao received the B.S. degree in software engineering from Jiangsu Normal University. He is currently pursuing the M.S. degree in software engineering at Jiangsu Normal University. His current research interests include software testing and software defect prediction.

Qiao Yu received her Ph.D. degree in computer science and technology from China University of Mining and Technology. She is currently the Associate Professor in the School of Computer Science and Technology, Jiangsu Normal University. Her current research interests include software testing and software defect prediction.

Xiaoying Chen received the B.S. degree in software engineering from Jiangsu Normal University. She is currently pursuing the M.S. degree in software engineering at Jiangsu Normal University, China. Her current research interests include software engineering, formal methods and applications.