Cross-Project Software Defect Prediction Based on Class Code Similarity

WANZHI WEN\textsuperscript{1,2}, (Member, IEEE), CHENQIANG SHEN\textsuperscript{1}, XIAOHONG LU\textsuperscript{1}, ZHIXIAN LI\textsuperscript{1}, HAOREN WANG\textsuperscript{1}, RUINIAN ZHANG\textsuperscript{1}, AND NINGBO ZHU\textsuperscript{1}\textsuperscript{1}
School of Information Science and Technology, Nantong University, Nantong 226019, China
\textsuperscript{2}Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, Ministry of Industry and Information Technology, Nanjing 211106, China

Corresponding author: Wanzhi Wen (wenwanzhi@126.com)

This work was supported in part by the College Students’ Innovation and Entrepreneurship Training Program under Grant 202110304086Y; and in part by the Open Funds of Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, Ministry of Industry and Information Technology, under Grant NJ2018014.

ABSTRACT Software defect prediction techniques can help software developers find software defects as soon as possible. It can also reduce the cost of software development. This technique usually predicts the target project through the entire source project. However, the data distribution difference between the entire source project and the target project is generally large, so the software defect prediction accuracy is not high. We propose a cross-project software defect prediction technique based on class code similarity CCS-CPDP. Firstly, this technique converts the code set extracted by AST (Abstract Syntax Tree) into a vector set through the DTI (Doc2Bow and TF-IDF) strategy; Secondly, the similarity will be calculated between the vector set of target projects and training projects; Finally, according to the principle of the majority decision subordinate category in KNN, the number of most similar class instances of the training project is determined, the source project is refined by selecting the class instance, and then software defects are predicted and evaluated. We compared CCS-CPDP with software defect prediction methods based on four traditional classification models (KNN, Random Forest, Naive Bayes, and Logistic Regression). Experimental results show that CCS-CPDP can improve the effectiveness of CPDP in terms of recall and f1-score.

INDEX TERMS Source code similarity, source project selection, software defect prediction, machine learning.

I. INTRODUCTION

In recent years, with the development of Internet technology, the scale of software has expanded. While the complexity of functions and development difficulties had also increased rapidly, resulting that software defects had emerged. At the same time, the software code had certain defects. If the defects are serious, it may lead to economic losses and even security accidents. Therefore, in practical work, how to predict more software defects and minimize various problems caused by defects has become a research hotspot in the field of software engineering.

In the 1970s, research on software defect prediction began to flourish. Many researchers used machine learning models to predict defects based on software metrics [1], [2]. In terms of intra-project defect prediction (WPDP), for example, Catal and Diri [3] focused on machine learning and intelligent algorithms, aiming to explore the impact on defect prediction through data set size, metric sets and feature selection techniques. Since the training and testing metrics are extracted from the same project, the data distribution is relatively similar, and the prediction effect is generally better. In terms of cross-project defect prediction (CPDP), since most of the data...
come from new or different projects, the data distribution between projects is quite different, resulting in the inability of intra-project prediction methods to be transferred to cross-project prediction. Therefore, many researchers conducted research based on the data distribution characteristics. For example, Nam et al. [4] established a set of rules based on traditional metrics to select appropriate normalization options for a given source project pair, so as to conduct TCA+ Defect prediction research; Liu et al. [5] automatically selected two source projects with the highest distribution similarity to the target project from a set of candidate source projects according to traditional metrics, and established a two-stage cross-project prediction model through source project selection and transfer learning algorithm. Most of the research within these projects or across projects focused on the coarse-grained prediction at the file, package or module level [6], [7], [8]. However, the coarse-grained level had differences in the locating code scope and the data distribution between projects, which was affected by the current research perspective. Inspired, this paper started with class-level prediction, and determined the most suitable source project by selecting the most similar class instance files, so as to shorten the distribution difference between data in a fine-grained manner and improve the defect prediction performance.

Based on the above analysis, this paper proposes a cross-project software defect prediction method based on class code similarity, referred to as CCS-CPDP. Firstly, Using Doc2Bow and TF-IDF strategy, referred to as DTI, to quantify the source code nodes of each project version class file extracted by AST into a vector set; Secondly, according to the similarity model, each class instance file in the target file and each training project has been calculated the similarity of all class instance files; Thirdly, according to the principle of majority decision-making in K-Nearest Neighbors(KNN), the top n most similar class instances in the training project corresponding to a certain class of instance files in the target project were selected as the source selection index. Finally, according to the source selection index, the similarity of each selected n most similar class instances was accumulated to determine the similarity value of each training project, and then the project with the highest similarity was selected as the refined source project to obtain defect prediction results. In order to evaluate the effectiveness of the method, the experimental results showed that the method has improved CPDP by comparing it with the experimental results of four traditional machine learning models (KNN, Naive Bayes, Random Forest and Logistic Regression), Recall and F1-score results have been all improved.

The contributions of the paper are:
1) In view of the code node set extracted by AST, semantic information will inevitably be weakened in the process of quantization. This paper selects the DTI Strategy to highlight key codes and enhance semantic information.
2) This paper proposes a cross-project software defect prediction method (CCS-CPDP) based on class code similarity, and we compare the baseline methods in cross-project prediction to verify the effectiveness of the method.

II. RELATED WORK
After introducing a number of WDPD and CPDP methods, this section details software defect prediction based on class instance selection and source project selection.

A. DEFECT PREDICTION BASED ON INSTANCE SELECTION
In the field of software defect prediction, most researchers have found that dataset quality, such as data noise, class imbalance, etc., would have a certain impact on prediction performance. In order to improve the quality of the dataset and avoid problems such as numerical noise, fluctuation, and missing from affecting the prediction, researchers proposed corresponding optimization methods [9], [10].

Among them, class instance selection can effectively reduce redundant samples, and many typical methods include heuristic-based search methods and instance-specific domain methods.

Based on a heuristic search algorithm, the selection of key subsets is regarded as a combinatorial optimization problem, and a heuristic search strategy is used to find the optimal solution, such as a cluster-based cross-project software defect prediction feature proposed by Chao Ni et al. [11] The selection method consists of two stages. The first stage used the density-based clustering method to cluster the features, and the second stage designed three heuristic strategies. In the feature selection, the heuristic strategy was used to extract the features from each cluster select features.

Instance-specific domain methods are mainly based on the KNN, which classifies instances according to the relevant domains of the instances in the training set, such as the hybrid nearest-neighbor instance selection method for CPDP proposed by Duksan Ryu et al [12]. To solve the problem of differences in data distribution among different data, the two methods of KNN and Naive Bayes were used to learn local and global knowledge for mixed classification.

In the current class instance selection research, how to select the most similar class instance is still a basic problem, so it is necessary to explore a new class instance selection method to improve the effect of software defect prediction.

B. SOURCE PROJECT SELECTION TECHNIQUES
In the process of CPDP, if the selection of source projects is not considered, there may be instability. On this basis, if a source project with high similarity is identified, CPDP can be improved.

To determine the source project method with the highest similarity, a training data selection (TDS) method proposed by Herbold [13] is an unsupervised method that calculates the Euclidean distance between the source project and the target project directly. Then, the training projects with close data distribution were selected as the source projects.

Different from the above-mentioned TDS and SPE, this paper proposes a method to select the most similar class...
instances through source code similarity, and select the most suitable source project to optimize the CPDP.

III. RESEARCH METHODS
In this section, we elaborate on the proposed cross-project software defect prediction technique based on class code source code similarity. The framework of our method is as Figure 1.

A. DATA PROCESSING
1) VECTOR SET EXTRACTION
In order to express the code semantic information of the class file in each project, we used Abstract Syntax Tree (AST) to extract the class file source nodes of the source project [14], [15], [16] to form a node-set. All source projects in this article were extracted by AST to generate corresponding vector sets. In the construction process, each source code file was parsed into an AST, and its root node represented a complete source file, and each node’s information was represented by the node type, such as class-Declaration, expression Statement, Variable-Declaration, method Declaration, etc. The specific extraction process was shown in Figure 2 and Figure 3.

2) VECTOR TRANSFORMATION STRATEGY
In the existing space vector conversion strategies, including Word2Vec [17], GloVe [18], Doc2Bow and TF-IDF method [1], since Word2Vec generates corresponding word vectors in an unsupervised way, all words are represented as unified...
meanings and dimension vectors but will weaken the semantic information of the text. While the GloVe is a tool based on global word frequency statistics, and it achieves similarity by capturing semantic features, which is a complement to the Word2Vec global corpus. But it cannot highlight the text. Under comprehensive consideration, in order to highlight key codes, enhance code semantic information, and reduce the influence of code-independent words, this paper decides to choose DTI (Doc2Bow and TF-IDF). The specific steps are as follows:

1) Create a dictionary: In order to calculate the frequency of source words and ensure the accuracy of creating a dictionary, it is stipulated that words with a frequency greater than 1 are included, and words less than 1 are discarded. The construction format of the dictionary is key and value, such as dictionary [“if”,1], [“i”,2], [“test2”,3].

2) Bag-of-words representation vector: The Doc2Bow model is selected, and its process is shown in Figure 4. The code keywords are placed in a specific environment to highlight their relevance, so as to reduce code redundancy problems and improve the quality of code similarity prediction. We used this method to count the word frequency of each different word, and then convert it into a vector using the bag-of-words representation method according to the dictionary established in the previous step.

3) Construct the corpus: Based on the constructed dictionary, it is stipulated that words with a frequency greater than 1 are included, and words less than 1 are discarded. The construction format of the dictionary is key and value, such as dictionary [“if”,1], [“i”,2], [“test2”,3].

4) Initialize the TF-IDF model: we built the TF-IDF model based on the corpus in the previous step and constructed the corpus.

B. SELECTION OF THE MOST SIMILAR CLASS INSTANCE

1) SIMILARITY CALCULATION
After obtaining the class instance file vector of each project, the cosine similarity was used to measure the similarity between class instances. Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. If the words between the instances are more similar, the code similarity would be higher. Similarity values between all class files of the training projects correspond to the instance files, so as to form a plurality of corresponding similarity value sets.

2) DETERMINE THE RANGE OF SELECTED VALUES
According to the set of similarity values calculated in step 1, the set was processed in descending order. In view of this method, the idea of KNN was adopted and improved. Specifically, after the K value was determined, the subordinate category rule was determined according to the majority decision rule, and the most similar class instances were selected. In order to determine the n value, the experiment was carried out by enumerating them one by one. To make the selected n value more representative, considering the size of datasets which was selected, and because of the KNN idea of the CCS-CPDP, in order to avoid the disaster of the KNN dimension, the value is limited to 2 ∼ 20, etc. The CCS-CPDP determined the class instances of the final n most similar training projects.

3) SELECTION OF SOURCE PROJECTS
In order to further ensure the prediction effect of the selected training project and the target project, the similarity of n class instances selected according to the target project was accumulated to represent the training project as the basis for the selection of the source project. After generating the corresponding source project selection basis for all training projects, we selected the training project with the highest similarity, and refined the source project according to the selected n class instances. Since the refined source project was similar to the source project of the target project, Thereby, the data distribution difference between the source project and the target project could be narrowed.

4) PREDICTION AND EVALUATION
The CCS-CPDP is mainly based on the algorithm idea of KNN that predicts software defects, which can ensure the selection of the most similar class instances of the project. In view of the traditional KNN idea is to determine the subordinate category according to the K adjacent samples near a sample in the feature space, so the CCS-CPDP drew on the idea of this algorithm, and determined the defects of the target project class instance according to the n most similar class instances by the selected source project similarity value set.

According to the defect situation of the target project class instance determined by the CCS-CPDP, it is compared with the traditional machine learning model (KNN, Naive Bayes, Random Forest and Logistic Regression), and the refined source project selection is also compared with the current one. The source project selection strategies are compared to verify the effectiveness of the CCS-CPDP for software defect prediction.
IV. EXPERIMENT AND RESULT ANALYSIS

A. RESEARCH QUESTION
We first studied the number of class instances selected by the CCS-CPDP. By selecting the number of class instances, we could obtain better prediction performance, and then compared other current classification model methods and selected refined source projects to improve performance.

In the following experiments, we validated a cross-project software defect prediction technique based on class code similarity, including class instance selection and refined source project selection. Then, we compared the current classification models with the CCS-CPDP as the main method. Finally, we analyzed and compared the performance of CCS-CPDP to select refined source projects and other mainstream techniques.

In total, we investigated the following three questions:

RQ1: How many class instances can the CCS-CPDP method select to achieve optimal performance?

RQ2: Does the CCS-CPDP outperform the methods based on other classification models?

RQ3: Does CCS-CPDP outperform other source project selection methods?

B. DATASETS
In this experiment, since the Promise project data set contains data from different open source Java projects, each version contains multiple class instance files, and the project’s defect prediction data set metrics are open source and public, which can be used for the text. The experiments provide good metric information and code, the following is the dataset table of the Promise project.

| TABLE 1. Datasets. |
| --- | --- | --- |
| Datasets | #Version | #Instances |
| synapse | 1.0/1.1/1.2 | 635 |
| lucene | 2.0/2.2/2.4 | 782 |
| ant | 1.3/1.4/1.5/1.6/1.7 | 1692 |
| camel | 1.0/1.2/1.4/1.6 | 2784 |
| poi | 2.0/2.5/3.0 | 1378 |
| xalan | 2.4/2.5/2.6 | 2411 |
| xerces | 1.2/1.3/1.4 | 1481 |

C. EVALUATION
In this experiment, the common evaluation metrics Accuracy, Precision, Recall and F1-Score are used to evaluate the prediction model. The calculation of the four indicators involves the following indicators: 1. tp (true) refers to the positive samples predicted by the model as positive, that is, the correct rate of judgment as true. 2. tn (true negative) is a negative sample predicted by the model to be negative, that is, the correct rate of judgment as negative. 3. fp (false positive) is predicted by the model as a positive negative sample, that is, the false positive rate. 4. fn (false negative) is a positive sample predicted by the model to be negative, that is, the false negative rate.

Accuracy is the correct rate, indicating the proportion of correctly classified samples to the total number of samples, which is positively related to the performance of the classifier. The specific formula is as follows:

\[ \text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \]  \hspace{1cm} (1)

Precision is the precision rate, which means the ratio of the number of times the correct judgment is positive to all positive cases. The specific formula is as follows:

\[ \text{Precision} = \frac{tp}{tp + fp} \]  \hspace{1cm} (2)

Recall is the recall rate, which indicates how many positive examples in the overall example are predicted correctly, that is, the prediction is tp (true). There are two cases of positive examples. One is to predict the original positive example (tp) as a positive example (tp), and the other is to predict the original positive example (tp) as a negative example (fn). The specific formula is as follows shown:

\[ \text{Recall} = \frac{tp}{tp + fn} \]  \hspace{1cm} (3)

F1-score is the F1 score, that is, the harmonic mean of precision and recall. It includes two important indicators Precision and Recall. The specific formula is as follows:

\[ F1 - \text{score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \]  \hspace{1cm} (4)

D. PROCEDURE
This experiment is a cross-project software defect prediction based on class code similarity. The following experimental steps will be set up to ensure the accuracy and validity of the experimental results.

1) Select the version of any project as the target project.
2) For cross-project experiments, select any version of different projects as the test set.
3) Select a class instance of the target project, calculate the similarity with all class instances of a certain training project, select the n most similar class instances (the determination of n is enumerated one by one, 2 ∼ 20), and the selected class instance similarity is accumulated as an indicator for source project selection.
4) The model trains the selected n training project instances to predict the instance defect situation of the selected target project.
5) Repeat the process from 3) to 4) until all instances of the target project are predicted to be completed, and the final defect situation is obtained.
6) Compare the predicted defect situation with the actual defect situation, and get the experimental indicators Accuracy, Precision, Recall and F1-score.
7) Repeat the process of 2) to 6) until all training projects have completed the prediction of all class instances of the target project.
8) Sort according to the indicators selected in 3) Obtain all source projects, and select the source project with the highest indicator corresponding to the current target project.

9) Repeat the process from 1) to 8) until the selection of the highest index source project corresponding to all target projects is completed.

E. BASELINE

To evaluate the effectiveness of the CCS-CPDP for software defect prediction, considering the size of datasets we selected, this paper will compare it with these baseline methods under cross-project defect prediction:

1) K-Nearest Neighbors (KNN): Find the k nearest samples near a sample, most of which belong to a certain category, then the sample also belongs to this category. For a known sample, the sample has n class possibilities, select k samples similar to the sample, \( x(i) \) \((i = 1, \ldots, k)\) is the class corresponding to the k samples, \( Y(x(i)) = Y(x(i)) + 1 \) \((i = 1, \ldots, k)\), \( Y(x(j)) = \max Y(x(i)) \) \((i, j = 1, \ldots, k)\), then \( x(j) \) is the category of the sample.

2) Naive Bayes [20]: It is a probabilistic classifier. It is assumed that all predictors are independent of each other and the attributes of the samples are independent of each other. The probability distribution is learned through the given training set. Based on this model, calculate the probability that the sample belongs to each class so that the sample belongs to the class with the highest probability.

3) Random Forest [21]: the technique, proposed by Breiman, contains multiple decision trees, when deciding which class the sample belongs to, it uses voting to decide which class it belongs to. For a known sample with n class possibilities, each node of each tree has at most n children. The sample walks through the nodes of a tree according to certain rules and order to obtain a category, and the category with the most independent results of multiple trees is the category of the sample.

4) Logistic Regression [22]: the technique, based on regression, is usually suitable for binary classification problems. According to the training set, we find a straight line, input the X of the sample, output the Y value, and Y follow certain rules to obtain the category of the sample. For example, the rule is that Y is greater than or equal to 0.5, and the sample is category A; otherwise, the sample is category B.

F. RESULTS AND ANALYSIS

In order to verify the effectiveness of the CCS-CPDP, from cross-project experiments, Promise open source data were selected for experiments, and Accuracy, Precision, Recall and F1-score were selected as evaluation indicators. The above four indicators were expressed by acc, pre, rec and f1 respectively, imp represents the improve (percentage of

| TABLE 2. CPDP performance evaluation based on KNN. |  |
|---|---|---|---|---|
| n=2 | 0.546 | 0.559 | 0.376 | 0.368 |
| n=3 | 0.541 | 0.539 | 0.536 | 0.458 |
| n=4 | 0.550 | 0.559 | 0.438 | 0.397 |
| n=5 | 0.544 | 0.543 | 0.533 | 0.451 |
| n=6 | 0.552 | 0.570 | 0.465 | 0.412 |
| n=7 | 0.549 | 0.563 | 0.532 | 0.448 |
| n=8 | 0.551 | 0.567 | 0.476 | 0.415 |
| n=9 | 0.553 | 0.566 | 0.535 | 0.449 |
| n=10 | 0.555 | 0.570 | 0.493 | 0.424 |
| n=11 | 0.553 | 0.556 | 0.534 | 0.445 |
| n=12 | 0.554 | 0.564 | 0.497 | 0.424 |
| n=13 | 0.554 | 0.576 | 0.537 | 0.448 |
| n=14 | 0.557 | 0.577 | 0.504 | 0.429 |
| n=15 | 0.554 | 0.566 | 0.537 | 0.445 |
| n=16 | 0.558 | 0.575 | 0.510 | 0.433 |
| n=17 | 0.555 | 0.562 | 0.538 | 0.446 |
| n=18 | 0.559 | 0.583 | 0.513 | 0.435 |
| n=19 | 0.556 | 0.574 | 0.538 | 0.446 |
| n=20 | 0.556 | 0.580 | 0.508 | 0.428 |
| Avg. | 0.552 | 0.566 | 0.505 | 0.432 |
improvement), impall represents the improve all (the total percentage of improvement of the four evaluation indicators), base represents the experimental results of traditional metrics.

1) RQ1: HOW MANY CLASS INSTANCES CAN THE CCS-CPDP SELECT TO ACHIEVE OPTIMAL PERFORMANCE?
In view of the fact that the CCS-CPDP continues to use and improve the idea of the KNN, it is based on the principle of majority decision-making of KNN. It determined the number of the most similar class instances of this method, through the experimental method of enumerating one by one, and using KNN. The model selection experiment results were compared.

We used the KNN as the benchmark model to predict software defects across projects. As shown in TABLE 2, it can be seen that the prediction effect of the KNN, when n is between 2 and 20, is gradually increasing and gradually decreasing. Firstly, Since the value of n accounts for a small proportion of the sample size of each project, which can effectively avoid the disaster of KNN dimension, it is reasonable to choose 2~20 as the experimental value; Secondly, based on the CCS-CPDP, in the same way as the class instance selection method of KNN, cross-project software defect prediction is performed, as shown in TABLE 3; Thirdly, comparing the cross-project defect prediction results based on the KNN, as shown in Figure 5 and Figure 6,
although the CCS-CPDP has a slight decrease in the effect of the Accuracy and Precision values, the Recall value except \( n = 3 \) are all better than the comparison value, and the Recall value is 11.3% higher than the KNN, and its F1-score value is 8.8% higher than the KNN; Finally, the improvement rate of Accuracy, Precision, Recall and F1-score values except for \( n = 3 \), the total value of the accumulated improvement is better than the reference value, so the CCS-CPDP using the DTI vector transformation strategy is effective in CPDP.

In view of the fact that the CCS-CPDP is significantly better than the reference value of KNN, and referring to TABLE 2, when \( n = 12 \), the total improvement rate of Accuracy, Precision, Recall and F1-score reaches the highest value of 23.7%, which is \( n \) Peak total boost rate from 2 to 20. Based on the above analysis, it is reasonable for the CCS-CPDP to select 12 most similar class instances.

2) RQ2: DOES THE CCS-CPDP OUTPERFORM THE METHODS BASED ON OTHER CLASSIFICATION MODELS?

In order to prove that the CCS-CPDP can improve the performance of software defect prediction across projects, and in view of the fact that this method continues to use and improves the idea of the KNN, this method was first compared with the experimental results of the KNN, and as shown in the boxplot in FIGURE 7, it effectively proves that the CCS-CPDP is significantly better than the KNN.

In order to further ensure the effectiveness of the CCS-CPDP, considering the large number of classification models for CPDP, based on the selected values of \( n = 12 \) class instances determined in RQ1, the selected corresponding class instances were compared with the remaining traditional classification models (Naive Bayes, Logistic Regression and Random Forest) are compared, where base represents the experimental results of traditional metrics in various classification models.

Compared with the Naive Bayes, as shown in TABLE 4, although the Accuracy and Precision have decreased, the Recall has increased by 17%, and the F1-score has increased by 8.4%. The overall improvement rate is positive, so the CCS-CPDP improves the cross-project software defect prediction results compared to the naive Bayesian model.

Compared with the Random Forest, as shown in TABLE 5, although the Precision has decreased, the Accuracy has increased by 1.1%, the Recall has increased by 18.1%, and the

| TABLE 4. Performance comparison between CCS-CPDP and CPDP based on Naive Bayes. |
|---------------------------|-----------|---------|-----------|---------|---|---|---|---|
|                           | acc        | imp      | pre       | imp      | rec      | imp    | fl        | imp    |
| base                      | 0.592      | —        | 0.586     | —        | 0.527    | —      | 0.488     | —      |
| CCS-CPDP                  | 0.558      | -3.4%    | 0.548     | -3.8%    | 0.697    | 17%    | 0.572     | 8.4%   | 18.2% |

| TABLE 5. Performance comparison between CCS-CPDP and CPDP based on Random Forest. |
|---------------------------|-----------|---------|-----------|---------|---|---|---|---|
|                           | acc        | imp      | pre       | imp      | rec      | imp    | fl        | imp    |
| base                      | 0.547      | —        | 0.604     | —        | 0.516    | —      | 0.433     | —      |
| CCS-CPDP                  | 0.558      | 1.1%     | 0.548     | -5.6%    | 0.697    | 18.1%  | 0.572     | 13.9%  | 27.5% |

| TABLE 6. Performance comparison between CCS-CPDP and CPDP based on Logistic Regression. |
|---------------------------|-----------|---------|-----------|---------|---|---|---|---|
|                           | acc        | imp      | pre       | imp      | rec      | imp    | fl        | imp    |
| base                      | 0.517      | —        | 0.474     | —        | 0.507    | —      | 0.372     | —      |
| CCS-CPDP                  | 0.558      | 7.1%     | 0.548     | 7.4%     | 0.697    | 19%    | 0.572     | 20%    | 61%   |
F1-score has increased by 13.9%. The overall improvement rate is positive, so the CCS-CPDP improves the cross-project software defect prediction results compared with the Random Forest.

Compared with the Logistic Regression, as shown in Table 6, the Accuracy is increased by 7.1%, the Precision is increased by 7.4%, the Recall is increased by 19%, and the F1-score is increased by 20%, all of which are improved. So the CCS-CPDP improves the cross-project software defect prediction results compared with the Logistic Regression.

To sum up, within the cross-project, the experimental results of the CCS-CPDP are significantly improved compared with the four traditional models, except for the slight decrease in Accuracy and Precision, and the experimental results of the Recall and F1-score are significantly improved, and the total improvement rates are all positive, therefore, the CCS-CPDP outperforms the current methods of other classification models.

3) RQ3: DOES CCS-CPDP OUTPERFORM OTHER SOURCE PROJECT SELECTION METHODS?

In this problem, it can be seen from the RQ1 that when the value of n = 12 is selected for the most similar class instances, the CCS-CPDP has the best prediction effect. Based on this, the class instances and target projects selected by the CCS-CPDP. The top (12) of the similarity of each class instance is accumulated, and the similarity between all training project instances and the target project instance was calculated to obtain the average similarity of the instance with the target project as a whole. The average similarity of each training project is as follows as shown in Table 7, the most suitable source project after refinement corresponding to the target project is determined.

As shown in Table 7, for each target project, the similarity value of each training project is accumulated, and the source project with the highest average value is selected according to the average value of the accumulated value of the training project. Precision values are slightly less effective, but the Recall and F1-score of the selected source project are effective. Analyze the Recall and F1-score of the selected source projects: in terms of ranking, the ant-1.7 project, the selected source project Recall and F1-score rank first; the synapse-1.2 project, the selected source project Recall value ranks first, F1-score ranks second; the lucene-2.4 project, the selected source project Recall and F1-score ranks first; the poi-3.0 project, the selected source project Recall value rank third, ranked third in F1-score; the xalan-2.6 project, selected source project Recall and F1-score ranked first; the xerces-1.4 project, selected source project Recall and F1-score ranked first; the camel-1.6 project, the selected source project is ranked first in Recall value and second in F1-score.

The CCS-CPDP selects the refined source projects in the average experimental results, as shown in Table 7. In order to verify the effectiveness of the refined source project strategy selected by the CCS-CPDP, this paper selects

| Source project selection strategy | acc | pre | rec | F1 |
|----------------------------------|-----|-----|-----|----|
| Mean_log                         | 0.632 | 0.659 | 0.449 | 0.483 |
| Std_log                          | 0.684 | 0.644 | 0.546 | 0.563 |
| Median_log                       | 0.619 | 0.646 | 0.478 | 0.475 |
| CCS-CPDP                        | 0.551 | 0.537 | 0.867 | 0.634 |
| Improve1                        | -8.1% | -12.2% | 41.8% | 15.1% |
| Improve2                        | -13.3% | -10.7% | 32.1% | 7.1% |
| Improve3                        | -6.8% | -10.9% | 38.9% | 15.9% |

TABLE 8. Performance comparison based on source projects selected by the CCS-CPDP and other three source project selection strategies.
three source project selection strategies, Mean_log, Std_log and Median_log for comparison, as shown in TABLE 8, although the selection effect of the three source projects is slightly lower in terms of Accuracy and Precision, the Recall in Improve1 (and the Mean_log strategy) is improved by 41.8%, F1-score increased by 15.1%; in Improve2 (with Std_log strategy) Recall increased by 32.1%, F1-score increased by 7.1%; in Improve3, Recall increased by 38.9%, F1-score increased by 15.9%.

In general, in cross project, the CCS-CPDP selects the refined source project to predict the target project, which is better than the other three source project selection effects in terms of Recall and F1-score.

G. THREAT TO VALIDITY

Internal validity is related to the chosen contrasting model. We only selected the current traditional machine learning models (KNN, Random Forest, Naive Bayes and Logistic Regression) to compare with the CCS-CPDP, and its effect is only better than the four models, not all machine learning models. In comparison, the effect may not be improved significantly; and the experimental data has not been subjected to a lot of preprocessing work, such as feature value selection, class sampling, etc., the potential threat to the effectiveness is the potential error in the implementation of our method.

External validity is related to the chosen dataset size. Although we use the dataset from Promise, which contains 14 projects, it does not apply to other projects. If we use datasets from different repositories, the experimental results may be different. In addition, our model only uses the Java language, and no experiments are performed on other languages such as Python and C. To minimize these threats, we plan to analyze our model on more diverse defect datasets in the near future.

Construct validity is the applicability of our performance metrics. Given that Recall and F1-score have been widely used in previous software engineering research, we adopt these two commonly used evaluation metrics, Recall and F1-score, to evaluate whether the proposed CCS-CPDP improves significantly over the baseline method. In this experiment, multiple tests are carried out to take the average value to avoid the randomness of the experiment and to ensure the correctness of the experimental results.

V. CONCLUSION AND FURTHER WORK

In this paper, a cross-project software defect prediction method CCS-CPDP based on class code similarity is proposed. This method is based on the principle of the majority decision-making class in KNN and improved the effectiveness of CPDP. We first studied that the CCS-CPDP can achieve optimal performance by selecting a reasonable number of class instances, and then we compared the selected class instances with the current four classification models to verify the effectiveness of this method; finally, we refined the source projects, and the performance of the popular source project selection techniques are analyzed and compared.

The experimental results showed that the CCS-CPDP is better than four popular classification methods, and the selected refined source projects are also better than the popular source project selection techniques. It can further improve the performance of the current class instance and source project selection technology in cross project defect prediction.

In the future, we plan to use more project source code datasets to evaluate the method, and carry out data validation tests from various aspects to further verify the performance of the method proposed in this paper; In addition, we also plan to improve this method. This method follows the KNN idea and belongs to the single view graph for processing. We may start with the multi view graph and use the strong integration model to parallel processing to improve the defect prediction performance; Finally, in view of the size of the dataset selected in this paper, machine learning model will have better effects, if the dataset is expanded, we try to use the deep learning model for comparison, and then verify the performance of this method.
[14] J. Lin and L. Lu, “Semantic feature learning via dual sequences for defect prediction,” IEEE Access, vol. 9, pp. 13112–13124, 2021, doi: 10.1109/ACCESS.2021.3051957.

[15] T. Shippey, D. Bowes, and T. Hall, “ Automatically identifying code features for software defect prediction: Using AST N-grams,” Inf. Softw. Technol., vol. 106, pp. 142–160, Feb. 2019, doi: 10.1016/j.infsof.2018.10.001.

[16] J. Deng, L. Lu, S. Qiu, and Y. Ou, “A suitable AST node granularity and multi-kernel transfer convolutional neural network for cross-project defect prediction,” IEEE Access, vol. 8, pp. 66647–66661, 2020, doi: 10.1109/ACCESS.2020.2985780.

[17] J. Tian and Y. Tian, “A model based on program slice and deep learning for software defect prediction,” in Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN), Aug. 2020, pp. 1–6, doi: 10.1109/ICCCN49398.2020.9209658.

[18] H. Wang, W. Zhuang, and X. Zhang, “Software defect prediction based on gated hierarchical LSTMs,” IEEE Trans. Rel., vol. 70, no. 2, pp. 711–727, Jun. 2021, doi: 10.1109/TR.2020.3047396.

[19] S. Patil and B. Ravindran, “Predicting software defect type using concept-based classification,” Empirical Softw. Eng., vol. 25, pp. 1341–1378, Mar. 2020, doi: 10.1007/s10664-019-09779-6.

[20] T. Mori and N. Uchihira, “Balancing the trade-off between accuracy and interpretability in software defect prediction,” Empirical Softw. Eng., vol. 24, no. 2, pp. 779–825, Apr. 2019, doi: 10.1007/s10664-018-9638-1.

[21] L. Breiman, “Random forests,” Mach. Learn., vol. 2, pp. 199–228, Oct. 2001, doi: 10.1023/A:1010933404324.

[22] N. Li, M. Shepperd, and Y. Guo, “A systematic review of unsupervised learning techniques for software defect prediction,” Inf. Softw. Technol., vol. 122, Jun. 2020, Art. no. 106287, doi: 10.1016/j.infsof.2020.106287.