Hindawi
Security and Communication Networks
Volume 2021, Article ID 5558561, 19 pages
https://doi.org/10.1155/2021/5558561

Research Article

Research on Cross-Company Defect Prediction Method to Improve Software Security

Yanli Shao, Jingru Zhao, Xingqi Wang, Weiwei Wu, and Jinglong Fang

Key Laboratory of Complex Systems Modeling and Simulation, School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China

Correspondence should be addressed to Jinglong Fang; fjl@hdu.edu.cn

Received 18 January 2021; Revised 9 April 2021; Accepted 6 August 2021; Published 24 August 2021

Academic Editor: Honghao Gao

As the scale and complexity of software increase, software security issues have become the focus of society. Software defect prediction (SDP) is an important means to assist developers in discovering and repairing potential defects that may endanger software security in advance and improving software security and reliability. Currently, cross-project defect prediction (CPDP) and cross-company defect prediction (CCDP) are widely studied to improve the defect prediction performance, but there are still problems such as inconsistent metrics and large differences in data distribution between source and target projects. Therefore, a new CCDP method based on metric matching and sample weight setting is proposed in this study. First, a clustering-based metric matching method is proposed. The multigranularity metric feature vector is extracted to unify the metric dimension while maximally retaining the information contained in the metrics. Then use metric clustering to eliminate metric redundancy and extract representative metrics through principal component analysis (PCA) to support one-to-one metric matching. This strategy not only solves the metric inconsistent and redundancy problem but also transforms the cross-company heterogeneous defect prediction problem into a homogeneous problem. Second, a sample weight setting method is proposed to transform the source data distribution. Wherein the statistical source sample frequency information is set as an impact factor to increase the weight of source samples that are more similar to the target samples, which improves the data distribution similarity between the source and target projects, thereby building a more accurate prediction model. Finally, after the above two-step processing, some classical machine learning methods are applied to build the prediction model, and 12 project datasets in NASA and PROMISE are used for performance comparison. Experimental results prove that the proposed method has superior prediction performance over other mainstream CCDP methods.

1. Introduction

With the increasing scale and complexity of software, software security, and quality issues are becoming more and more important. Generally, it is difficult for developers to directly develop a safe and reliable software system all at once. Due to the influence of many factors, such as irregular development process and excessive code complexity, it is inevitable that there are defects in the software system; attackers can use them to destroy the software confidentiality, damage the software integrity, and cause serious security problems and economic losses. The software system needs to be fully tested to ensure its security and reliability. In the software development process, the later the defect is discovered, the higher the repair cost. However, test resources such as test time and personnel are limited, and modules with greater probability of defects should be prioritized to improve software repair efficiency and shorten software development cycle. Software defect prediction (SDP) is a common method to detect potential defects that may endanger software security and reliability. Currently, SDP has become a hot issue in software engineering for a long time [1–3]. Traditional SDP aims to use historical defect dataset within or across projects with the same metrics to build a prediction model to predict potential defects in new projects [4, 5]. The general process of SDP is as follows: first, analyze the software code or development process, extract the metrics [6, 7] related to the software defect information,

and combine the defect label information to constitute the training dataset. Then, construct the defect prediction model based on the above defect dataset. Finally, use the prediction model to predict whether there are defects in the other modules to optimize the allocation of test resources. This can assist developers to discover and repair potential defects in the software code of new projects as early as possible, thereby reducing software testing costs and improving software security, reliability, and maintainability.

Generally, large-scale projects like Java projects often contain many classes or methods. Each class is taken as a sample, and the number of lines of code, the number of operands, the coupling and cohesion between objects, cyclomatic complexity, etc., are selected as the metrics. Moreover, the defect label is obtained through manually review of each class code, and it also needs to be verified by domain expert to ensure its correctness. However, due to the extremely low efficiency and long cycle of manually determining module defects, the historical defect dataset for a new project is usually insufficient to build an accurate prediction model. More importantly, accurately marking module defect information is a prerequisite to ensure the effectiveness and accuracy of the constructed model, but ensuring the accuracy of defect labels requires repeated verification by a large number of professionals. This work is very costly for some small- and medium-sized enterprises. Since there are often fewer labeled defect datasets for a new project, it is difficult to build an effective prediction model to predict potential defects to improve software quality. As a result, many researchers began to study the cross-project defect prediction (CPDP) and cross-company defect prediction (CCDP) with the aim to make full use of the existing defect datasets of other projects to build an effective and accurate prediction model. The difference is that the former uses cross-project datasets with consistent metrics but different data distributions as the source dataset, while the latter uses cross-company project datasets with both different metrics and different data distributions as the source dataset for prediction model construction.

CPDP generally refers to constructing a defect prediction model to predict the defect information in the target project, under the condition that the metric number and meaning of the source and target projects are the same. The main research work is how to solve the problem of inconsistent data distribution. Since the dataset for constructing the model and using the model comes from two projects with different data distributions, it may reduce the accuracy of defect prediction to a certain extent. Thus, the data distribution transformation method is studied to increase the data distribution similarity between projects to improve the prediction accuracy. However, most companies are only willing to provide researchers with extracted metrics and defect information, rather than complete source code to ensure that the project will not be maliciously attacked. This leads to the fact that the metric number and meaning between the source and target projects for different companies are quite different, CPDP will no longer be applicable and heterogeneity will occur. In fact, the meaning and number of metrics extracted by the same company are also very different from before as more and more defect-related metrics are proposed to describe the project information more completely [8]. Heterogeneous problems are common, especially when the dataset used for defect prediction comes from multiple companies. For a target project, there is often no or few source project datasets that have exactly the same metrics as the target project, and the samples from different projects are very different because of their diversity in size and metrics. This makes it difficult to build a prediction model directly based on the existing historical defect dataset to achieve the accurate defect prediction of target project.

Therefore, CCDP is more practical and can be applied to cross-company projects with different metrics, different dataset sizes, and different data distributions. It mainly studied how to build a more accurate defect prediction model by unifying metrics and adjusting data distribution. Currently, some researches build CCDP models by extracting common metrics of the source and target projects, but these methods are not informative when the source and target projects from different companies have few or no common metrics. Another part of the research first uses the feature selection method to filter out some metrics of the source project and then matches with the metrics of target project. After the metrics are unified, the homogeneous defect prediction method can be used to solve the cross-company problems. However, the selected metrics may not fully describe the defect characteristics of the software module. This method not only ignores the impact of relevant metrics of source project on the defect prediction model construction but also does not consider the degree of differences between the source and target samples. Furthermore, the data distribution spaces from different projects are different, resulting in the prediction model built directly using the source dataset is not suitable for the target project, thereby reducing the prediction accuracy to a certain extent.

To address the above issues, this study proposes a CCDP method based on metric matching and sample weight setting to improve the security and reliability of software. The innovations lie in：on the one hand, a clustering-based metric matching method is proposed to solve the metric inconsistent and redundancy problem. The multigranularity metric feature vector is extracted to unify the metric dimension between the source and target projects. Moreover, metric clustering is applied to eliminate metric redundancy, and representative metrics are extracted to facilitate subsequent one-to-one metric matching. In this way, the cross-company heterogeneous defect prediction problem is transformed into a homogeneous problem. On the other hand, a sample selection-based weight setting method is applied to adjust the source data distribution to make it as consistent as possible with target project dataset. It uses sample selection frequency information as the impact factor to increase the weight of source samples that are more similar to the target samples, so as to improve the data distribution similarity between projects to further improve the prediction accuracy. The rest of this study is organized as follows: Section 2 is the related work. Section 3 introduces the core work in detail, including the clustering-based metric matching method and sample selection-based weight setting

algorithm. Section 4 is the related experimental verification and performance analysis. Finally, the conclusions and future work are given in Section 5.

2. Related Work

Traditional SDP aims to construct a defect prediction model using historical defect dataset of within project to predict the defects in the same projects. This can accurately and timely predict whether the software module is defective or not in the early development stage, thereby improving the software security, reliability, and maintainability. However, this method has great limitations due to the lack of labeling defect dataset for new projects, and it is often difficult to use small amount of historical dataset to build an effective and accurate prediction model. Therefore, many researchers have begun to study CPDP and CCDP to make full use of historical within-project, cross-project, and cross-company defect dataset to construct an effective prediction model to improve the prediction accuracy of target project. The following will introduce related work from the perspective of CPDP and CCDP.

2.1. CPDP. As mentioned above, CPDP mainly refers to the defect prediction when the metric number and meaning of the source and target projects are basically the same. In recent years, many studies have been conducted on CPDP [9]. Zimmermann et al. [10] deeply analyzed the feasibility of CPDP through 28 datasets from 12 real-world large software projects and conducted 622 groups of cross-project prediction experiments and found that only 3.4% achieved satisfactory results. For example, although the defect prediction model based on the Firefox project can achieve good results in the IE project, but not vice versa. Many factors need to be considered, such as the development process, the data itself, and the belonged domain. Generally, the data distribution between the source and target projects is quite different, so the model trained directly based on the source project normally does not perform well in the target project. Even if the metrics are consistent, the defect dataset itself still has problems such as metric redundancy and class imbalance, which may affect the prediction accuracy to a certain extent. Therefore, the CPDP research work is roughly carried out from the above three aspects.

Rahman [11] analyzed 12 different open source projects from the aspects of defect prediction performance, prediction stability, and dataset characteristics and concluded that there are potential commonalities between different software projects. Moreover, the data distribution transformation can be used to increase the similarity between projects to improve the prediction accuracy. Some researchers studied data distribution transformation method to reduce data distribution differences to improve prediction accuracy. Pan et al. [12] proposed a transfer component analysis (TCA) algorithm to map the datasets of different projects to the latent feature space to minimize the distance between the source domain and target domain. Although this method reduces the data distribution differences between different projects, the selection of data normalization method has a great impact on the final prediction performance. After that, Nam et al. [13] proposed the TCA + method based on TCA, which analyzed the characteristics of the source and target projects and adaptively selected the most suitable data normalization method. This method further increases the data distribution similarity between different projects and improves the accuracy of prediction models. The performance of TCA + varies greatly when using different source projects to build prediction models. To find the source project that is most similar data distribution to the target project to construct the model, Liu et al. [14] proposed a two-phase transfer learning model (TPTL) for CPDP. A source project estimator (SPE) is proposed to automatically choose two source projects with the highest distribution similarity to a target project from candidates and leverage TCA + to build two prediction models based on the two selected projects and combine their prediction results to further improve the prediction performance. Jinyin et al. [15] proposed a collective training mechanism consists of two-phase source data expansion phase and adaptive weighting phase for defect prediction, which makes the feature distributions of source and target projects similar to each other by transfer learning, and uses the particle swarm optimization algorithm to comprehensively consider the multiple source projects to predict the target project. Sun et al.[16] proposed a Collaborative Filtering-based source Projects Selection (CFPS) method, which used collaborative filtering algorithm to recommend the appropriate source projects to filter out the project that is most similar to the target project. Jin et al. [17] used the kernel twin support vector machines (KTSVMs) to implement domain adaptation (DA) to match the distributions of training data for different projects. These methods solve the problem of inconsistent data distribution to a certain extent and improve the prediction accuracy.

Some researchers have studied the problem of metric redundancy. Menzies et al. [18] solved the feature redundancy problem through feature subsets selection, but they found that the feature subsets selected from different datasets are inconsistent, which means that it is meaningless to search for feature subsets that are applicable to all projects. Liu et al. [19] proposed a feature selection method based on clustering, which selected a specified number of features in each class by calculating the correlation with another feature in same cluster, and avoided selecting irrelevant features. Similarly, Wang et al. [20] applied genetic algorithms to select feature subsets and simultaneously detected outliers in the dataset to further improve the quality of feature subsets. These methods considered the impact of feature redundancy on model construction and improve the prediction accuracy to a certain extent. However, the selection criterion is to discard the metrics with relatively small relevance, which will cause the amount of information contained in the metrics to be insufficient to represent the entire project in some cases, making it difficult to build a better model.

In addition, some researchers have conducted class imbalance studies, which is a common problem in CPDP and can affect the performance of prediction model. Sun et al. [21] proposed a coding-based ensemble learning (CEL)

method to solve the class imbalance problem. Jing et al. [22] proposed an improved subclass discriminant analysis (ISDA) method. The premise of the above methods is that the metrics between the source project and target project are the same or coincident in most cases. However, there is often no or few source projects that have exactly the same metrics as a certain target project, and CPDP method may not be applicable. In this case, CCDP is more important in practical applications due to the relatively limited dataset in the same projects and the different metrics between projects.

2.2. CCDP. CCDP aims to achieve accurate defect prediction between cross-company projects with different metrics, different dataset sizes, and different data distributions. Many researchers have done a lot of research based on this difference.

Some research works build the prediction model by extracting common metrics of the source and target projects. Turhan et al. [23] proposed a nearest neighbor filtering (NNFilter) algorithm. This method extracts the common metrics in the source and target projects and then uses a clustering algorithm to filter source samples similar to the target samples as the training dataset for prediction model construction. After that, Peters et al. [24] further proposed the Peter Filter method, which also first extracts the source samples similar to each target sample, and further improves the similarity between projects by filtering the training dataset. Ma et al. [25] proposed a transfer Naïve Bayes (TNB) algorithm. This method extracts the common metrics between the source and target projects, calculates their similarity by Euclidean distance, uses the gravitational formula to convert it into the weight of the training sample, and constructs a prediction model that is more suitable for the target datasets. The above methods are not universal since there are rarely the same metrics between the source and target projects. Furthermore, it may ignore unused metrics that may be useful for defect prediction. For example, there are 39 and 20 metrics in the NASA and PROMISE defect datasets, respectively, while their common metrics is only one. Therefore, the problem of few or no common metrics between the source and target projects has become a bottleneck for CCDP.

To solve this problem, some researchers have improved the data distribution similarity of the source and target projects from the perspective of transforming the data distribution, thereby improving the prediction accuracy. Jing et al. [26] proposed a canonical correlation analysis-based transfer learning (CCA+) algorithm. It uses the unified metric representation (UMR) method and typical association analysis to make the data distribution between source and target projects more similar, so as to improve the prediction accuracy through considering the linear separability of defect datasets. Ying et al. [27] proposed a kernel function-based typical association analysis method (Kernel CCA), which maps the defect dataset to the high-dimensional Hilbert space and then applies the CCA method to the projected transformation matrix of the source and target projects.

Another part of the research studies how to unify the metrics to convert the heterogeneous problem into homogeneous problem and use homogeneous defect prediction methods to solve the cross-company problem. Nam et al. [28] proposed a heterogeneous defect prediction (HDP) method, which not only solves the problem of feature redundancy during metrics selection but also realizes the metric matching between the source and target projects. However, this method only selects 15% of the source metrics, which may ignore the metrics that are strongly related to defect prediction in the source project and reduce the defect prediction accuracy. Similarly, feature matching and transfer method (FMT) proposed by Yu et al. [29] also performs metric matching based on feature similarity. But this method is only applicable to the filtered source project metrics that are less than the target project metrics, and the scale of the source and target projects depends on the companies with fewer dataset, so it cannot make full use of the richer source dataset.

Based on the above analysis, most studies only extracted part of the original metrics to build the defect prediction model without considering the impacts of unselected metrics on the prediction model. Meanwhile, the sample scale of the source and target projects from different companies was limited to the smaller of the two. Moreover, inconsistent data distribution has a certain impact on the prediction accuracy, and the different effects of samples with different degrees of similarity are not taken into account in most cases, which should also be considered to further improve the similarity of data distribution, thereby building a more accurate prediction model.

3. CCDP Method Based on Metric Matching and Sample Weight Setting

3.1. Method Overview. As a method to improve software security and reliability, SDP has always been a research hotspot in the field of software engineering. Building an accurate defect prediction model often requires sufficient historical defect dataset with defect labels, but there is usually not enough labeled defect dataset for a new project. Although the CPDP and CCDP methods are widely used to enrich defect dataset, there are still some problems in practical applications such as inconsistency of metrics and differences in the data distribution between source and target projects, which reduces the prediction performance. Thus, a CCDP method based on metric matching and sample weight setting is proposed in this study to address above issues. The method overview is shown in Figure 1. The specific process is as follows:

(1) Clustering-based metric matching: since the metric meaning and number between the source and target projects are quite different in the CCDP, it is necessary to match metrics between projects to facilitate the defect prediction model construction. Therefore, a clustering-based metric matching method is proposed here. First, extract multigranularity metric feature vectors from the source and target projects

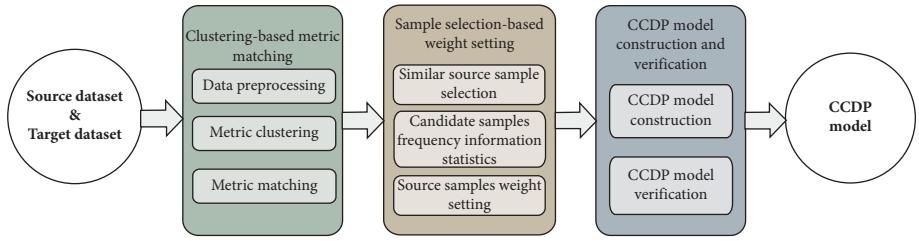

Figure 1: Method overview.

and unify them into the same dimension to calculate the similarity between metrics in the data preprocessing step. Then, apply metric clustering to obtain the representative feature vectors. Finally, perform one-to-one metric to solve the problem of inconsistent metrics. In this way, the cross-company heterogeneous defect prediction problem can be transformed into a homogeneous problem.

(2) Sample selection-based weight setting: even if the metrics are unified, the source and target datasets still have differences in data distribution, which reduces the effectiveness and accuracy of the defect prediction model. Hence, a sample weight setting method based on sample selection is applied to adjust the source data distribution. The key is to improve the data distribution similarity by increasing the weight of source samples similar to the target sample, thereby improving the defect prediction accuracy.

(3) CCDP model construction and verification: through the above metric matching and data distribution adjustment, the training dataset is now obtained. Finally, common machine learning methods are used to build the prediction model, and the proposed method and the mainstream CCDP algorithms are applied to 12 projects in NASA and PROMISE for experimental comparison and performance analysis to verify its feasibility and superiority. Accurate defect prediction results can be used to optimize new software testing resources, thereby reducing testing costs and improving product quality.

3.2. Clustering-Based Metric Matching. In the CCDP, the original source project dataset cannot be directly used as training dataset to build a prediction model for target defect prediction due to the inconsistent metrics of source and target projects. Even if the metrics are unified, the metric redundancy problem will also increase the training time and affect the prediction accuracy. Therefore, a clustering-based metric matching method is proposed to solve the above problems, as shown in Figure 2, which is mainly carried out in the following three steps:

(1) Data preprocessing: data preprocessing is performed first to unify the metric dimensions. Here, after data normalization, a multigranularity metric feature vector representation method is proposed to convert the source and target metrics into the same dimension, while retaining the metric information as much as possible to describe the data characteristics of source and target projects.

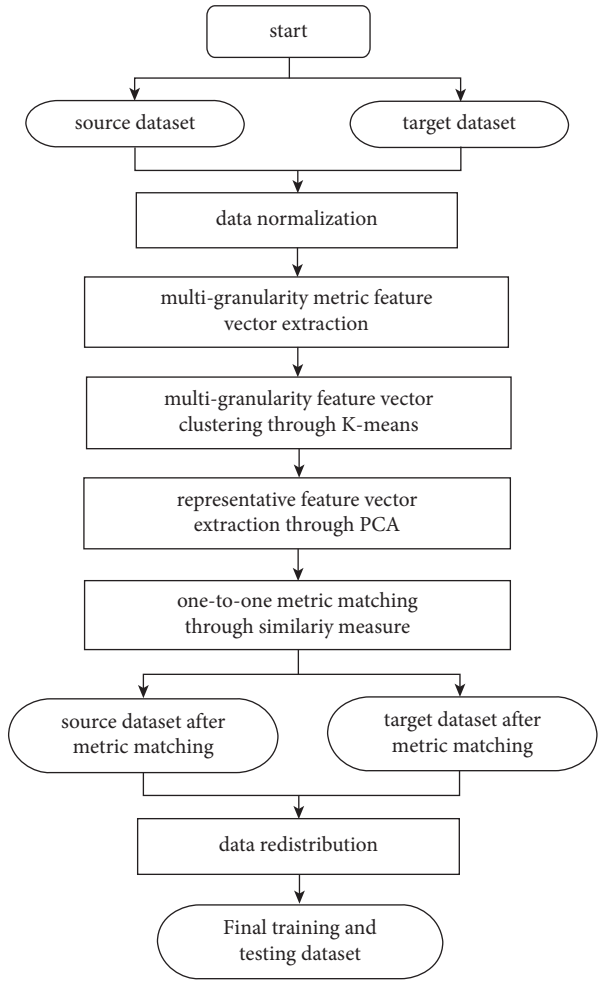

Figure 2: Flowchart of clustering-based metric matching.

(2) Metric clustering: after the metric dimensions are unified, the K-means clustering method with the same number of clusters is applied to the source and target projects, respectively. The Euclidean distance that can describe the similarity between metrics is used as a clustering measure. Meanwhile, the principal component analysis [30] (PCA) method is applied to extract the representative feature vector of each cluster in preparation for the following one-to-one metric matching.

(3) Metric matching: after unifying the metric dimension and number through above two steps, the metric pairs with highest similarity are matched in turn based on the Euclidean distance between the source and target representative metrics. This achieves one-to-one metric matching and eliminates the problem of inconsistent metrics. Even though the metrics have been unified through above operations, the extraction of multigranularity metric feature vector results in the loss of defect information of each sample in the dataset while the metric dimensions are unified. Thus, the datasets after metric matching cannot be used as training and testing dataset directly. Data redistribution should be performed based on the above clustering and matching results to obtain the final datasets for prediction model construction.

3.2.1. Data Preprocessing. To better understand the subsequent algorithm, some basic symbols are defined firstly, as shown in Table 1; the table shows the relevant information of the source and target project samples and metrics. Initially, the metric feature vectors of the source project and target project are represented as $S_{\text{original}} \in \{a_s^j|_{j=1}^{ls}\}$ and $T_{\text{original}} \in \{a_t^j|_{j=1}^{lt}\}$, respectively, where $a_s^j \in \Re^{n_s \times 1}$ and $a_t^j \in \Re^{n_t \times 1}$. There are three problems if the original metrics are directly used for metric matching. First, the dimensions of source and target metric vectors are quite different, but the premise of metric matching is that the metric dimensions of the source and target projects are consistent. For example, the sample number of source project CM1 in NASA is 327, and one of its metric vector dimensions is $a_s^{327\times 1}$. The sample number of target project Ant-1.7 is 745, so the metric vector dimension is $a_t^{745\times 1}$. It is difficult to directly calculate the metric similarity due to the different number of samples. Second, even if the same sample numbers of the source and target projects are selected to make the dimensions the same, the similarity between metrics is also affected by the order of the selected samples. The source metric vector varies with the sample positions. Thus, the similarity between metrics calculated in this way has great randomness. Third, if the metric feature vector is directly extracted, the data information may be greatly lost. For example, the metric dimension of the CM1 project in NASA is $a_s^{327\times 1}$. After extracting the metric feature vector, it becomes $\text{Feavecs}^{5\times 1}$, which may greatly reduce the data information contained in the metrics.

Due to the different scale of source code between different projects and different modules, the metric dimension is quite different, which makes it difficult to build prediction model directly. The core of data preprocessing in Step (1) is multigranularity metric feature vector extraction, which aims to unify the metric dimension between source and target projects. The MaxMinNormalization method is used to first standardize the sample metrics to minimize the impact of the dataset itself on the model performance. Generally, the numerical statistical attributes, such as minimum, maximum, average, median, and standard deviation, can briefly describe the data characteristics. But only using these values to describe the metric is too simple. To fully represent the metric, here each metric a_s^j is sorted from small to large according to their values and divided into $m = 5$ parts evenly. Supposing that x^{ij} represents the jth metric value of the ith sample, the jth metric of S can be described as $a_s^j = \{x_s^{1j}, x_s^{2j}, \ldots, x_s^{ij}, \ldots, x_s^{n_s j}\}$. The corresponding minimum, maximum, average, median, and standard deviation in each part are calculated (as shown in equation (1)) and combined to form the final metric feature vector $(\text{Fea vec}_s^j)_{25\times 1}$.

$$\text{Fea vec}_s^{j-m} = \left\{\min\left(a_s^{j-m}\right),\ \max\left(a_s^{j-m}\right), \text{avg}\left(a_s^{j-m}\right), \text{median}\left(a_s^{j-m}\right), \text{std}\left(a_s^{j-m}\right)\right\} m \in [1, 5]. \tag{1}$$

Now the original dimension of source metrics $(n_s * 1)$ and target metrics $(n_t * 1)$ are transformed into the same dimension $(25 * 1)$. Repeat the above steps for all the metrics, the multigranularity metric feature vectors of S and T can finally be expressed as $S_{\text{feavec}} \in \{\text{Feavec}_s^j|_{j=1}^{ls}\}$ and $T_{\text{fea vec}} \in \{\text{Fea vec}_t^j|_{j=1}^{lt}\}$, respectively, where $\text{Fea vec}^j \in \Re^{25\times 1}$.

Multigranularity metric feature vector can not only represent the metrics more comprehensively by characterizing the original metric information as much as possible but also unify the dimensions of the source and target metrics. After that, the Euclidean distance of the above two feature vectors can be directly calculated as a similarity measure between metrics to facilitate the subsequent metric clustering and matching. The Euclidean distance between the ith source metric (Fea vec_s^i) and jth target metric (Fea vec_t^j) is

$$\text{Dist}\left(\text{Fea vec}_s^i, \text{Fea vec}_t^j\right) = \sqrt{\sum_{k=1}^{25}\left(\text{Fea vec}_s^{ik} - \text{Fea vec}_t^{jk}\right)^2}. \tag{2}$$

3.2.2. Metric Clustering. Through the above data preprocessing step, the multigranularity metric feature vectors of S and T are $S_{\text{fea vec}} \in \{\text{Fea vec}_s^j|_{j=1}^{ls}\}$ and $T_{\text{fea vec}} \in \{\text{Fea vec}_t^j|_{j=1}^{lt}\}$, respectively, where each metric $\text{Fea vec}^j \in \Re^{25\times 1}$. Data preprocessing unifies the metric dimensions of the source and target projects, but the number and meaning of their metrics are still different, making it difficult to perform one-to-one metric matching directly. Therefore, metric clustering and PCA in Step (2) are performed on the source and target projects to eliminate the

Table 1: Basic symbol definition.

Symbol	Definition
S	Source project dataset
T	Target project dataset
x_s^i, x_t^i	ith sample of source and target projects
a_s^j, a_t^j	jth metric of source and target projects
n_s, n_t	Number of samples in the source and target projects
l_s, l_t	Number of metrics in the source and target projects

redundancy between metrics, while unifying the number of metrics to facilitate subsequent metric matching. First, the K-means clustering method divides all the metrics in the $S_{\text{fea vec}}$ and $T_{\text{fea vec}}$ into K ($l_s<K$ and $l_t<K$) clusters ($C = \{C_1, C_2, \ldots, C_K\}$, respectively, which makes the metrics in the same cluster have a high correlation, and the metric correlation between different clusters is not large. Fea vec_s^{ij} and Fea vec_t^{ij} represent the ith multigranularity metric of cluster C_j of project S and T. During the clustering process, the Euclidean distance is used as clustering measure. The Euclidean distance between the pairwise metrics of projects is calculated based on equation (2).

$$\begin{aligned} &\text{Fea vec}_s^{ij} \in C_{sj} \text{ while} i \in [1, l_s], \quad j \in [1, K], \\ &\text{Fea vec}_t^{ij} \in C_{tj} \text{ while} i \in [1, l_t], \quad j \in [1, K]. \end{aligned} \tag{3}$$

During the clustering process, the Euclidean distance is used as a clustering measure. The Euclidean distance between the pairwise metrics of projects is calculated based on equation (2). Moreover, the PCA method is used to extract the principal component of each cluster as the representative metrics $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i | _{i=1}^K\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_t^i | _{i=1}^K\}$, where $\text{Rep vec}^i \in \mathfrak{R}^{25\times 1}$. This step aims to eliminate metric redundancy and retain clustering information as much as possible since the highly correlated metrics are clustered in a cluster. The specific flow is shown in Algorithm 1.

Note that the number of clusters of the source and target metric needs to be set the same to facilitate the following one-to-one metric matching. Furthermore, when the clustering number is different, the clustering results on the same dataset will be different, and it is difficult to define the optimal cluster number directly. If the parameter setting is too small, it will be difficult to construct an effective model due to the loss of information. But if the parameter setting is too large, the clustering result is meaningless, which makes it impossible to eliminate strong correlation features. Since different cluster numbers will greatly affect the final results, in the experiment part, some parameters within the appropriate range will be selected as reference parameters first, and then inappropriate parameter will be excluded by filtering out abnormal predicted values. Refer to Section 4.2 for more details.

3.2.3. Metric Matching. After metric clustering in Step (2), the source and target project dataset features are now represented as $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i | _{i=1}^K\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_s^i | _{i=1}^K\}$ (K is the number of metrics after clustering with the same metric dimension). Metric matching in Step (3) refers to matching K metrics of project S and T in pairs to improve the data distribution similarity of the source and target projects, so as to further improve the versatility and accuracy of defect prediction model constructed. The metric matching process is shown in Figure 3.

Firstly, the similarity between the source and target metrics is measured through the Euclidean distance based on equation (2), which can be represented by a K^*K dimensional matrix W, as shown in Figure 3(a). Each element W_{ij} represents the similarity measure between Rep vec_s^i and Rep vec_t^i. Then, the subscript of the smallest value in the matrix W would be selected as the matching pair between the source and target metric, i.e., if W_{ij} is the minimum value in W, it means that the metric Rep vec_s^i and the metric Rep vec_t^j are the most similar, so they are matched and the corresponding rows and columns in the matrix are deleted, as shown in Figure 3(b), where gray means deleted. Similarly, next if W_{12} is the minimum value after the first matching, the metric Rep vec_s^1 and Rep vec_t^2 are matched, and the 1st row and 2nd column will be deleted as well, as shown in Figure 3(c). According to the matching order, all the metrics in the project S and T are, respectively, matched in pairs until the matching process is completed.

Figure 4 shows the entire clustering-based metric matching process, and the details are as follows.

As shown in Figure 4(a), there are the original source project S and target project T contains n_s and n_t samples, respectively. The corresponding number of metrics is also different, namely l_s and l_t. Initially, the original source and target metrics are represented as $S_{\text{original}} \in \{a_s^j | _{j=1}^{ls}\}$ and $T_{\text{original}} \in \{a_s^j | _{j=1}^{ls}\}$, respectively, where $a_s^j \in \mathfrak{R}^{n_s\times 1}$ and $a_t^j \in \mathfrak{R}^{n_t\times 1}$. Lines of the same color in the same project represent the metrics with similar meaning. The metrics cannot be matched directly because the metric dimensions from different projects are different and the similarity between metrics cannot be directly calculated. In other words, there are metric inconsistencies and redundancy problems in cross-company defect prediction. Thus, metric matching must be performed between the original source and target project dataset to facilitate the construction of subsequent prediction model. As shown in Figure 4(b), after MaxMinNormalization, the multigranularity metric feature vectors of SNorm and TNorm, represented as $S_{\text{fea vec}} \in \{\text{Fea vec}_s^j | _{j=1}^{ls}\}$ and $T_{\text{fea vec}} \in \{\text{Fea vec}_t^j | _{j=1}^{lt}\}$, are extracted in the data preprocessing step. In this way, although the number of metrics remains unchanged, the metric dimension is unified to 25 ($\text{Fea vec}^j \in \mathfrak{R}^{25\times 1}$), while retaining the metric information as much as possible. Then, as shown in Figure 4(c), the K-means clustering method is applied to unify the metric number of different projects by setting the same number of clusters (here K is 3). Metrics with similar meaning (same color) are clustered together, and then the PCA method is performed on each cluster to extract the representative metrics $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i | _{i=1}^K\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_t^i | _{i=1}^K\}$, where $\text{Repvec}^i \in \mathfrak{R}^{25\times 1}$. This step solves the metric redundancy problem and prepares for the next step of metric matching. As shown in Figure 4(d), after calculating the Euclidean distance of different metrics

Input: source and target multigranularity metric feature vector $S_{\text{fea vec}} \in \left\{\text{Fea vec}_s^j \middle|_{j=1}^{ls}\right\}$ and $T_{\text{feavec}} \in \left\{\text{Fea vec}_t^j \middle|_{j=1}^{lt}\right\}$; number of clusters K;
Output: source and target representative vector $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i|_{i=1}^K\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_t^i|_{i=1}^K\}$;
(1) For each project $\in$ {Source project S, Target project T}
(2) Randomly select K metrics as the starting centroid of the project $C = \{C_1, C_2, \ldots, C_K\}$;
(3) Repeat the following process until convergence
(4) For each metric Fea $\text{vec}^i \in \Re^{25\times 1}$:
(5) Calculate the Euclidean distance to each starting centroid based on eq.(2);
(6) Assign the metric to its nearest cluster with minimum distance;
(7) End for
(8) For each cluster C_j ($j \in [1, K]$):
(9) Calculate the mean value of the cluster C_j and update its centroid;
(10) End for
(11) For each cluster C_j ($j \in [1, K]$):
(12) Use PCA method to extract the corresponding representative vector Rep vec^i;
(13) End for
(14) End for
(15) Output $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i|_{i=1}^K\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_s^i|_{i=1}^K\}$;

Algorithm 1: Metric clustering algorithm.

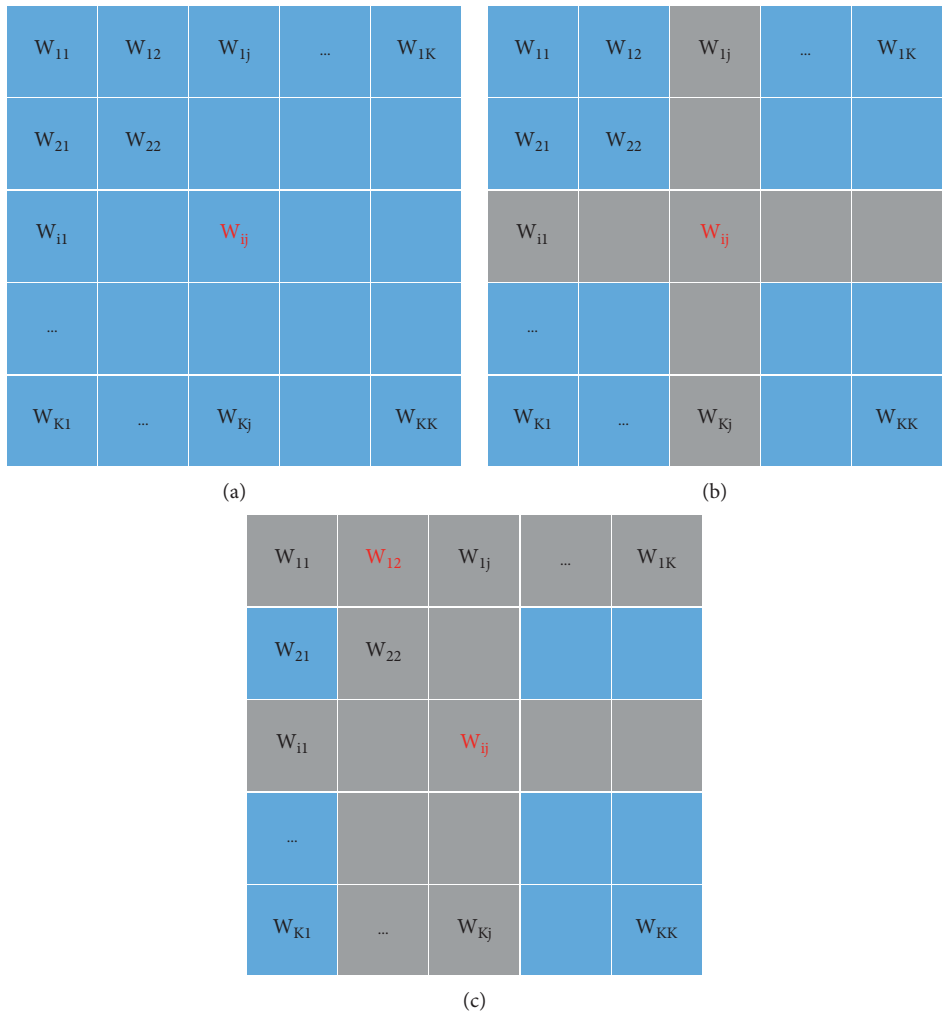

Figure 3: Metric matching process. (a) Original matrix, (b) matrix after first matching, and (c) matrix after second matching.

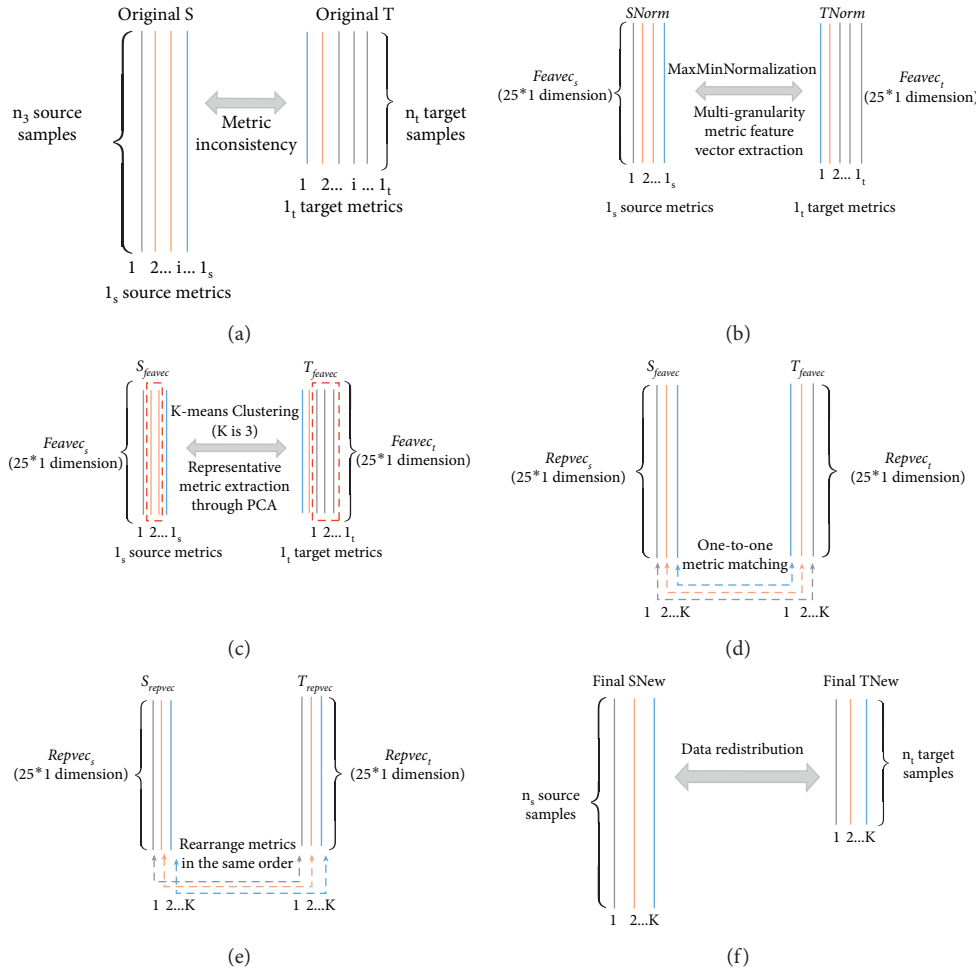

FIGURE 4: Schematic diagram of clustering-based metric matching. (a) Original datasets, (b) data preprocessing, (c) metric clustering, (d) metric matching, (e) metric arrangement, and (f) final datasets.

between different projects, one-to-one metric matching is performed based on the metric similarity information. The specific metric matching process is shown in Figure 3. Next, as shown in Figure 4(e), the matched metrics $S_{\text{rep vec}}$ and $T_{\text{rep vec}}$ are rearranged in the same order to make the data distribution of the source and target project as similar as possible. Finally, the original datasets are redistributed based on the above clustering results and metric order to obtain the final training and testing dataset, expressed as $SNew \in \mathfrak{R}^{n_s \times K} TNew \in \mathfrak{R}^{n_t \times K}$, as shown in Figure 4(f). The features of the source and target project datasets are restored to $S_{\text{rep vec}}{}' \in \left\{a_s^j \middle| {}_{j=1}^{K}\right\}$ and $T_{\text{rep vec}}{}' \in \left\{a_s^j \middle| {}_{j=1}^{K}\right\}$, respectively, where $a_s^j \in \mathfrak{R}^{n_s \times 1}$ and $a_t^j \in \mathfrak{R}^{n_t \times 1}$. This step is to restore the lost defect information of each sample in the dataset due to the extraction of multigranularity metric feature vector, so as to ensure the authenticity of the dataset used for prediction model construction and the fairness of comparison with other mainstream CCDP methods. The above operations have successfully unified the metrics and enhanced the data distribution similarity between source and target projects, so that the defect prediction model constructed using such source project dataset can achieve satisfactory results.

3.3. Sample Selection-Based Weight Setting. Although the clustering-based metric matching largely unified the source and target metrics, the data distribution difference between the source and target datasets makes the built defect prediction model still insufficient to achieve the accurate defect prediction of the target project. Therefore, how to select source data samples similar to the target ones and make the two data distributions as similar as possible to further improve the prediction accuracy is the focus of this section. Here, a sample selection-based weight setting algorithm, as shown in Algorithm2, is proposed to adjust the source data distribution to make it more similar to the target data distribution to build a more accurate defect prediction model.

The specific method steps are as follows:

(1) First, sample selection is performed to select the source samples similar to target samples. Similar to NNFilter [23], for each target sample, the Euclidean distance of all source samples is calculated and sorted, and then top N source samples with the smallest distance are selected as the candidate samples for each target one. In this study, N is set to 10 based on the practical experience [31].

Input: source and target $SNew \in \Re^{n_s \times K} TNew \in \Re^{n_t \times K}$, number of candidate samples N;
Output: the final training dataset $SNew\prime \in \Re^{n_{S\prime} \times K}$
(1) SCandidate = ∅ //array, which is used to store all the candidate training source samples
(2) For each sample x_t^j of target project T New:
(3) Advanced = ∅;//Array, which is used to store the selected samples in each loop
(4) For each sample x_s^i of source project SNew:
(5) Calculate the Euclidean distance between x_t^j and x_s^i based on equation (2);
(6) Sort source samples according to the above Euclidean distance information;
(7) Select the Top N source samples with the smallest distance and store them in advanced;
(8) SCandidate ← SCandidate + Advanced;
(9) End
(10) End
(11) Array Of Weight$^i \in \Re^{n_s \times 1}$ //array, which is used to store sample weight
(12) For each sample x_s^i of source project SNew:
(13) Statistic the sample frequency of x_s^i in SCandidate and update the Array Of Weight^i
(14) End
(15) Use MaxMinNormalization method to normalize ArrayOfWeight;
(16) Set the source sample weight based on ArrayOfWeight to obtain the final $SNew\prime \in \Re^{n_{S\prime} \times K}$

Algorithm 2: Sample selection-based weight setting algorithm.

(2) Second, the frequency information of each candidate sample is counted based on the aforementioned top N selection information. For the datasets with n target samples, $n * N$ source samples will be selected as candidate training samples, and some of source samples will be selected multiple times. It is assumed that the samples that are repeatedly selected are more similar to the target samples. Therefore, the frequency of each source sample being selected is counted as the basis for sample weight setting.

(3) Finally, the frequency information is used as a reference for sample weight setting. The maximum and minimum frequency value of entire candidate source samples are counted, and MaxMinNormalization is used to normalized the frequency information first, and then the weighted source samples with defect label information constitute the training dataset.

The above sample weight setting operation can accurately filter the source samples that are similar to the target sample and then use the calculated frequency information to set the source sample weight to further improve the data distribution similarity between the source and target projects, thereby improving the defect prediction performance.

3.4. CCDP Model Construction and Verification. Through the above metric matching and sample weight setting processing, the source and target project datasets with consistent metrics and similar data distribution are obtained, and the heterogeneity problem in CCDP is transformed into a homogeneous issue. Next, the commonly machine learning methods are used in the source dataset to build the defect prediction model to predict the defect information of target dataset. Extensive experiments should be performed on the experimental datasets for performance verification based on the evaluation indicators to prove the superiority of the proposed method.

3.4.1. Model Construction. Generally, there are many excellent machine learning models that perform well in defect prediction field. Here, the commonly machine learning methods such as logistic regression model (LR) [32], Naïve Bayes (NB) [33], and K-Nearest Neighbor (KNN) are used to build the defect prediction model. The model details are as follows:

(1) LR: when the dependent variable is dichotomous, LR is more suitable [34]. The method avoids the Gaussian assumption used in standard Naive Bayes.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k, \quad (4)$$

where p is the probability that the defective module was found and x_1, x_2, ..., x_k are the independent variables. $\beta_0, \beta_1, \ldots, \beta_k$ are the regression coefficients estimated using maximum likelihood.

(2) NB: a statistical learning scheme that assumes that metrics are equally important and statistically independent.

$$P(c_k|x) = P(c_k) \times \frac{P(x|c_k)}{P(x)}, \quad (5)$$

where c_k is a member of the set of values for the dependent metric and x represents unknown sample. NB finds the conditional probability of that sample being labeled c_k to classify the test samples. The c_k with the highest probability is chosen as the label for x.

(3) KNN: KNN is a classic nonparametric decision procedure that classifies x, an unknown sample in the category of its nearest neighbor. As one of the simplest defect prediction models, it is usually used as baseline.

3.4.2. Model Verification. After the construction of CCDP models, extensive experiments are performed for performance verification and analysis. Some evaluation indicators that are commonly used in software defect prediction performance verification are selected to evaluate the prediction performance of the CCDP models, and the specific definitions are as follows, where TP, TN, FP, and FN refer to the number of true positive, true negative, false positive, false negative, respectively.

(1) Pd (probability of detection or recall) is the percentage of defects that are predicted correctly within the defect class. A higher Pd means more defects are detected, so the ideal case is Pd = 1, where all the defects are detected:

$$\text{Pd} = \frac{\text{TP}}{(\text{TP} + \text{FN})}. \tag{6}$$

(2) Pf (probability of false alarm) refers to the percentage of nondefective samples that are incorrectly predicted within the nondefect class. The higher the Pf, the more time and cost are wasted to predict the true defect:

$$\text{Pf} = \frac{\text{FP}}{(\text{FP} + \text{TN})}. \tag{7}$$

(3) Precision is what percentage of samples predicted as defective are actually such. However, it does not tell us anything about the number of samples that the classifier mislabeled:

$$\text{precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}. \tag{8}$$

(4) F-measure ($F1$) is the harmonic mean of precision and recall. Precision is a measure of exactness, whereas recall is a measure of completeness. But there tends to increase one at the cost of reducing the other. F-measure is an alternative way to use precision and recall by combining them into a single measure:

$$\text{F1} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}. \tag{9}$$

(5) AUC (area under curve) is the area under the receiver operating characteristic curve. It is a useful measure for comparing different models because it is unaffected by class imbalance and is independent of the prediction threshold.

Generally, Pd and Pf will be affected by issues such as threshold setting and class imbalance and cannot effectively and accurately evaluate the prediction performance. In fact, a good classifier should both have higher Pd and lower Pf. F-measure and AUC are the trade-off measure that balances the performance between Pd and Pf. A higher F-measure means a better prediction performance. Especially, AUC is recognized by many researchers and is widely used in SDP [35]. The performance verification experiment based on the NASA datasets carried out by Jiang et al. [36] also proved that AUC is superior to other evaluation criteria in stability by comparing the variance of different evaluation indicators. The value of AUC is between 0 and 1. The larger the value of AUC, the better the performance of the prediction model.

Based on the above analysis, the overall algorithm flow is shown in Algorithm 3.

4. Experimental Verification and Performance Analysis

In this section, after the introduction of experimental datasets, three groups of experiments are designed to verify the feasibility and effectiveness of clustering-based metric matching method, sample selection-based weight setting method, and the overall proposed method. The threats to validity are given finally.

4.1. Experimental Dataset Introduction. All the experimental datasets are from NASA [37] and PROMISE, which are widely used in the field of SDP and are generally authoritative and recognized. The specific dataset information is shown in Table 2.

The number of metrics of NASA and PROMISE is 37 and 20, respectively, and the specific metric meaning is shown in Table 3 and 4, respectively. Generally, a project consists of multiple software modules, and a module is a sample in software defect prediction. The features that describe the software module mainly consist of Mccabe [38], Halstead, CK, etc. Mccabe mainly designs the metric from the structural point of view, such as 7-CYCLOMATIC_COMPLEX and 14-ESSENTIAL_COMPLEXITY in Table 3. Halstead mainly designs metrics from the aspect of code size, such as 22-HALSTEAD_LENGTH and 25-HALSTEAD_VOLUME. CK mainly designs the metrics from the object-oriented perspective, such as 1-wmc and 2-dit, in Table 4. Especially, complexity, coupling, and cohesion metrics [39, 40], i.e., the 7-CYCLOMATIC_COMPLEX, 11-DESIGN_COMPLEXITY , 14-DESIGN_COMPLEXITY, and 26-MAINTENANCE_SEVERITY in Table 3 and 4-cbo, 6-lcom, 7-ca, 8-ce, 13-moa, and 18-amc in Table 4, etc., are all important metrics related to software security. The metrics are extracted and defined from different views by different experts in different companies, covering all aspects of software quality, security and reliability, and can comprehensively represent software defect information to build more accurate prediction models. Meanwhile, the metric number and meaning of each project datasets of the two companies are quite different, which are very suitable for verifying the effectiveness of the proposed CCDP method in this study.

4.2. Clustering-Based Metric Matching Performance Analysis. The first experiment is to verify the performance of the clustering-based metric matching strategy, which is mainly divided into two parts, metric clustering performance analysis and metric matching performance analysis.

Input: source company project datasets $S \in \{x_s^i|_{i=1}^{n_s}, y_s^i|_{i=1}^{n_s}\}$ as training datasets;
Target company project datasets $T \in \{x_t^i|_{i=1}^{n_t}\}$ as test datasets;
Output: trained defect prediction model
(1) Clustering-based metric matching:
(a) Use MaxMinNormalization method to normalize S and T to get SNorm and TNorm;
(b) Extract the multigranularity metric feature vector of S and T, expressed as $S_{\text{fea vec}} \in \{\text{Fea vec}_s^j|_{j=1}^{ls}\}$ and $T_{\text{fea vec}} \in \{\text{Fea vec}_t^j|_{j=1}^{lt}\}$, where $\text{Fea vec}^j \in \Re^{25\times 1}$;
(c) Apply K-means clustering algorithm to cluster $S_{\text{fea vec}}$ and $T_{\text{fea vec}}$ into K clusters, respectively;
(d) Extract the principal component of each cluster through PCA as the representative vector $S_{\text{rep vec}} \in \{\text{Rep vec}_s^i|_{i=1}^{K}\}$ and $T_{\text{rep vec}} \in \{\text{Rep vec}_t^i|_{i=1}^{K}\}$, where $\text{Rep vec}^i \in \Re^{25\times 1}$;
(e) Perform one-to-one metric matching on $S_{\text{fea vec}}$ and $T_{\text{fea vec}}$ through metric matching;
(f) Redistribute SNorm and TNorm based on above steps, expressed as $S\text{New} \in \Re^{n_s\times K}$, $T\text{New} \in \Re^{n_t\times K}$.
(2) Sample selection-based weight setting:
(a) Use NNFilter to select N source samples that similar to each target sample based on Euclidean distance as candidate training data samples $S\text{Candidate} \in \{x_s^i|_{i=1}^{N\times n_t}, y_s^i|_{i=1}^{N\times n_t}\}$;
(b) Statistic the frequency of selected source samples in the SCandidate;
(c) Use samples frequency information in SCandidate as the basis for sample weight setting.
(3) CCDP model construction and verification:
(a) Use the weighted source samples as the training dataset and apply common machine learning methods including LR, NB, and KNN to construct the predict model;
(b) Perform experiments on multiple defect datasets and evaluate the performance of the proposed method.

Algorithm 3: Metric matching and sample weight setting based CCDP algorithm.

Table 2: Defect datasets.

Company	Dataset	Feature num	Samples num	Defective samples num	Nondefective samples num	Defect rate (%)	Imbalance ratio
NASA	CM1	37	327	42	285	12.8	6
	MW1	37	253	27	226	10.7	8
	PC1	37	705	61	644	8.7	10
	PC3	37	1077	134	943	12.4	7
	PC4	37	1287	177	1110	13.8	6
PROMISE	Ant-1.7	20	745	166	579	22.3	3
	Camel-1.6	20	965	188	777	19.5	4
	Ivy-2.0	20	352	40	312	11.4	7
	Jedit-4.3	20	492	11	481	2.2	43
	Synapse-1.2	20	256	86	170	33.6	1
	Velocity-1.6	20	229	78	151	34.1	1
	Xalan-2.4	20	723	110	613	15.2	5

4.2.1. Metric Clustering Performance Analysis. Generally, there are metric inconsistencies and redundancy problems in the cross-company defect prediction, so after extracting multigranularity metric feature vectors, K-means clustering, and PCA methods are applied to cluster the metrics and extract the representative metric to address above problems. As mentioned earlier, the parameter setting (i.e., clustering number) is very important, too large or too small may affect the accuracy of final prediction results. Therefore, the selection of the number of clusters will be discussed here. Five source projects in NASA are selected as training dataset to construct the prediction model to predict the defects of the target project Ant-1.7 in PROMISE. Each source project will conduct 6 experiments, and the number of clusters varies from 4 to 9, a total of 30 experiments. The experimental results are shown in Table 5.

In Table 5, K represents the clustering number and ratio is the imbalance rate of predicted labels in target projects, that is, the percentage of samples predicted to be nondefective divided by the percentage of samples predicted to be defective. Pd, Pf, F1, and AUC are used to evaluate the prediction performance. Among them, F1 and AUC are considered mainly due to their accuracy and robustness of performance evaluation. It can be observed from the table that

(1) The proposed method performs well in the heterogeneous cross-project defect prediction. The evaluation indicators such as Pd, Pf, F1, and AUC can achieve satisfactory results to a certain extent even with different K values. The reason is that the metric clustering can alleviate the metric redundancy while retaining the metric information maximally, thereby

Table 3: The metrics in NASA datasets.

ID	Feature
1	LOC_BLANK
2	BRANCH_COUNT
3	CALL_PAIRS
4	LOC_CODE_AND_COMMENT
5	LOC_COMMENTS
6	CONDITION_COUNT
7	CYCLOMATIC_COMPLEXITY
8	CYCLOMATIC_DENSITY
9	DECISION_COUNT
10	DECISION_DENSITY
11	DESIGN_COMPLEXITY
12	DESIGN_DENSITY
13	EDGE_COUNT
14	ESSENTIAL_COMPLEXITY
15	ESSENTIAL_DENSITY
16	LOC_EXECUTABLE
17	PARAMETER_COUNT
18	HALSTEAD_CONTENT
19	HALSTEAD_DIFFICULTY
20	HALSTEAD_EFFORT
21	HALSTEAD_ERROR_EST
22	HALSTEAD_LENGTH
23	HALSTEAD_LEVEL
24	HALSTEAD_PROG_TIME
25	HALSTEAD_VOLUME
26	MAINTENANCE_SEVERITY
27	MODIFIED_CONDITION_COUNT
28	MULTIPLE_CONDITION_COUNT
29	NODE_COUNT
30	NORMALIZED_CYLOMATIC_COMPLEXITY
31	NUM_OPERANDS
32	NUM_OPERATORS
33	NUM_UNIQUE_OPERANDS
34	NUM_UNIQUE_OPERATORS
35	NUMBER_OF_LINES
36	PERCENT_COMMENTS
37	LOC_TOTAL

Table 4: The metrics in PROMISE datasets.

ID	Feature	Description
1	wmc	Weighted methods per class
2	dit	Depth of inheritance tree
3	noc	Number of children
4	cbo	Coupling between object classes
5	rfc	Response for a class
6	lcom	Lack of cohesion in methods
7	ca	Afferent couplings
8	ce	Efferent couplings
9	npm	Number of public methods
10	Lcom3	Lack of cohesion in methods, different from LCOM
11	loc	Lines of code
12	dam	Data access metric
13	moa	Measure of aggregation
14	mfa	Measure of functional abstraction
15	cam	Cohesion among methods of class
16	ic	Inheritance coupling
17	cbm	Coupling between methods
18	amc	Average method complexity
19	max_cc	Maximum McCabe's cyclomatic complexity
20	avg_cc	Average McCabe's cyclomatic complexity

building a more accurate prediction model to further improve the prediction performance.

(2) For the same source project, different K values show different prediction performance, and in some cases, the difference is quite large. For example, when CM1 is used as the source project, the value of F1 falls in the area between 0.462 and 0.58, and AUC is in the range of 0.654 and 0.746, correspondingly that means the performance difference is about 10%. For an inappropriate clustering parameter, the performance may be very poor in some source projects, i.e., when MW1 is used as the source project and $K = 6$, all of the evaluation indicators are lower than the average level. As the number of clusters affects the clustering performance, when it is not set properly, the extracted principal component feature vector cannot effectively represent the metrics, thereby reducing the prediction accuracy.

(3) The imbalance ratio can be used as a reference for selecting effective clustering parameters. For example, when K is 6 for MW1, the prediction performance is poor and the ratio is also an abnormal value, so the imbalance ratio can be used to exclude an inappropriate cluster number. For a specific target project, the above 30 experiments are all aimed at this target project. Therefore, the imbalance ratio of the prediction results of 30 models is calculated for each target project, the related results are shown in Figure 5. The visualization results are the box plots corresponding to 7 target projects in PROMISE.

Here, we use the imbalance ratio of the predicted result as an indicator and then filter outliers by calculating the quartiles of the boxplot and finally choose an appropriate clustering number for the following metric clustering. As can be seen from Table 6, the prediction result class imbalance ratio box plot of Ant-1.7 has $Q1 = 2.027$, $Q3 = 2.555$. Here, the ratio between $Q1$ and $Q3$ is taken as candidate parameters to filter out inappropriate value of K. For the results of MW1, the ratio is 0.815 when $K = 6$ and the predicted result is obviously not within the normal range of ratio, thus the abnormal value is not involved to calculate the overall result. This strategy excludes the inappropriate parameters, such as $K = 4$, 5, 6, 8, 9, and the remaining parameter $K = 7$ is set to the number of clusters. When the ratio of a target project is not within the range to be selected, the K closest to the range will be selected to perform metric clustering. Generally, after excluding inappropriate K values, the median of remaining K values is chosen as the number of clusters to obtain the final prediction result.

4.2.2. Metric Matching Performance Analysis. To verify the effectiveness of metric matching method, we use the original defect dataset in NASA without and with metric matching as training dataset for comparison experiments, and build a LR model as defect prediction model. Here, Euclidean distance

Table 5: Performance results of different cluster numbers for target project Ant-1.7

Source	K	Ratio	Pd	Pf	F1	AUC	K	Ratio	Pd	Pf	F1	AUC
CM1	4	2.477	0.584	0.202	0.511	0.691	7	2.543	0.578	0.197	0.511	0.691
	5	2.278	0.687	0.196	0.58	0.746	8	3.022	0.488	0.18	0.462	0.654
	6	2.307	0.675	0.196	0.573	0.74	9	3.482	0.476	0.151	0.476	0.663
MW1	4	2.701	0.639	0.164	0.578	0.737	7	2.153	0.639	0.225	0.527	0.707
	5	2.56	0.633	0.18	0.56	0.726	8	2.012	0.645	0.242	0.518	0.701
	6	**0.815**	**0.307**	**0.621**	**0.177**	**0.343**	9	2.012	0.62	0.249	0.499	0.686
PC1	4	2.207	0.669	0.209	0.558	0.73	7	1.522	0.458	0.379	0.33	0.539
	5	1.918	0.681	0.246	0.537	0.718	8	2.153	0.536	0.254	0.443	0.641
	6	1.725	0.681	0.277	0.515	0.702	9	2.113	0.578	0.247	0.474	0.665
PC3	4	2.221	0.669	0.208	0.559	0.731	7	2.153	0.452	0.279	0.373	0.587
	5	2.413	0.645	0.192	0.557	0.726	8	2.074	0.705	0.216	0.574	0.744
	6	2.526	0.669	0.173	0.589	0.748	9	1.988	0.723	0.223	0.578	0.75
PC4	4	2.979	0.596	0.152	0.561	0.722	7	3.376	0.398	0.18	0.393	0.609
	5	2.012	0.289	0.344	0.232	0.472	8	2.351	0.663	0.194	0.567	0.734
	6	2.895	0.434	0.206	0.403	0.614	9	2.796	0.536	0.185	0.492	0.676

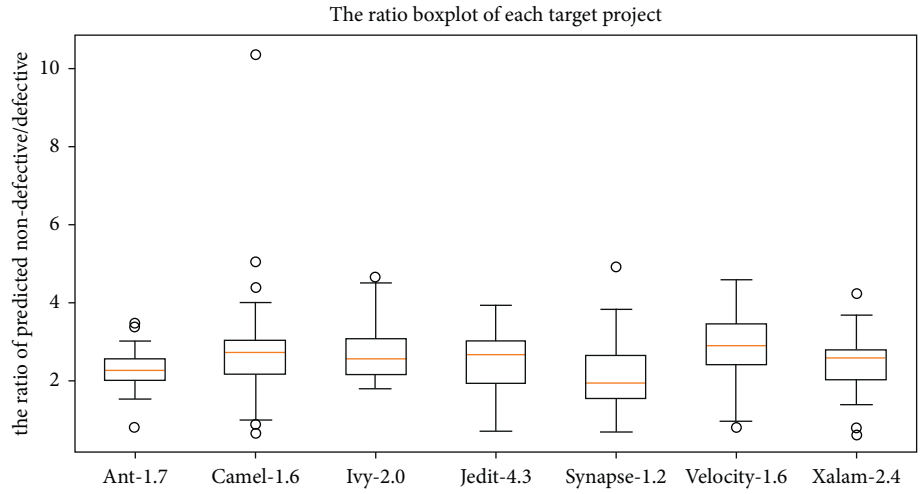

Figure 5: The imbalance ratio boxplots for each target project.

Table 6: Performance results of different K values for target project Ant-1.7

Source	Target	K	Ratio	Pd	Pf	F1	AUC
MW1	Ant-1.7	4	2.701	0.639	0.164	0.578	0.737
		5	2.56	0.633	0.18	0.56	0.726
		6	0.815	0.307	0.621	0.177	0.343
		7	**2.153**	0.639	0.225	0.527	0.707
		8	2.012	0.645	0.242	0.518	0.701
		9	2.012	0.62	0.249	0.499	0.686
	Median		2.153	0.639	0.225	0.527	0.707

is used as a measure to support metric matching. The experimental results are shown in Table 7, AUC is selected as the evaluation indicator here. The larger the AUC value, the better the prediction model, and the bold value in the table indicates better performance. In the table, the AUC-Original refers to the AUC values obtained when the source project dataset without metric matching is used as training dataset. AUC-MM refers to the AUC values obtained when the source project dataset with metric matching is used as training dataset for prediction model construction. It can be seen from the table that, in most cases, the prediction model constructed based on the dataset after metric matching shows better prediction performance, higher prediction stability, and accuracy. Under the same machine learning model, the final average AUC of the source project dataset after metric matching in 35 groups of experiments is 31.05% = (0.65–0.496)/0.496 higher than the source project dataset without metric matching. The reason is that metric matching not only unifies the metrics and eliminates the metric redundancy problem, and transforms the heterogeneous

Table 7: Metric matching performance analysis results.

No.	Source	Target	AUC-original	AUC-MM
1	CM1	Ant-1.7	0.569	**0.746**
2	MW1	Ant-1.7	0.431	**0.707**
3	PC1	Ant-1.7	0.427	**0.641**
4	PC3	Ant-1.7	0.687	**0.748**
5	PC4	Ant-1.7	0.424	**0.734**
6	CM1	Camel-1.6	**0.611**	0.549
7	MW1	Camel-1.6	0.457	**0.536**
8	PC1	Camel-1.6	0.419	**0.55**
9	PC3	Camel-1.6	**0.574**	0.549
10	PC4	Camel-1.6	0.472	**0.589**
11	CM1	Ivy-2.0	0.569	**0.761**
12	MW1	Ivy-2.0	0.382	**0.71**
13	PC1	Ivy-2.0	0.457	**0.603**
14	PC3	Ivy-2.0	0.681	**0.756**
15	PC4	Ivy-2.0	0.467	**0.729**
16	CM1	Jedit-4.3	0.555	**0.689**
17	MW1	Jedit-4.3	0.405	**0.693**
18	PC1	Jedit-4.3	0.333	**0.674**
19	PC3	Jedit-4.3	0.656	**0.678**
20	PC4	Jedit-4.3	0.454	**0.7**
21	CM1	Synapse-1.2	0.556	**0.635**
22	MW1	Synapse-1.2	0.398	**0.676**
23	PC1	Synapse-1.2	0.45	**0.656**
24	PC3	Synapse-1.2	**0.69**	0.665
25	PC4	Synapse-1.2	0.45	**0.65**
26	CM1	Velocity-1.6	0.495	**0.68**
27	MW1	Velocity-1.6	0.371	**0.593**
28	PC1	Velocity-1.6	0.362	**0.599**
29	PC3	Velocity-1.6	**0.613**	0.596
30	PC4	Velocity-1.6	0.496	**0.579**
31	CM1	Xalan-2.4	0.583	**0.602**
32	MW1	Xalan-2.4	0.42	**0.611**
33	PC1	Xalan-2.4	0.394	**0.6**
34	PC3	Xalan-2.4	0.623	**0.663**
35	PC4	Xalan-2.4	0.437	**0.607**
	Average		0.496	**0.65**

prediction problem into a more common homogeneous prediction problem, but also improves the data distribution similarity between the source and target project. Therefore, metric matching is effective and can improve the performance of the defect prediction model to a certain extent.

4.3. Sample Weight Setting Performance Analysis. The second experiment is to verify the impact of data distribution adjustments based on sample weight setting on performance improvement. After performing metric matching, the datasets with and without sample weight setting were used to train the LR model for performance comparison, respectively. The experimental results are shown in Table 8. Here, AUC-MM refers to the AUC value obtained by training the prediction model using the source project dataset with only metric matching. AUC-MMWS refers to the AUC value obtained by training the prediction model after sample weight setting. Similarly, the bold in the table indicates a relatively high AUC value, which has better prediction performance. It can be seen that, in most cases, the prediction model constructed after setting the sample weights performs better. Even if the former (AUC-MM) performs better, the performance of the two is not of much different, i.e., for the 4th, 10th, 19th, 22th, 26th experiment, and so on. Under the same machine learning model, the source project dataset after sample weight setting in the 35 experiments is 5.54% = (0.686–0.65)/0.65 higher than the source project dataset with only metric matching. Therefore, the prediction performance can be optimized by improving the data distribution similarity between the source and target projects by sample weight setting strategy.

4.4. Overall Prediction Performance Verification. The last experiment is to verify the overall prediction performance of the proposed method (abbreviated as MMWS) in this study. Comparative experiments are made with the mainstream CCDP algorithms (FMT [29], HDP [28], and RM [29]) and the proposed method to demonstrate its applicability and effectiveness. Five datasets of NASA are used as the training dataset in the CCDP to build the defect prediction model, and seven datasets of PROMISE are used as the test dataset, a total of 35 groups of experiments. Three commonly used

Table 8: Sample weight setting performance analysis results.

No.	Source	Target	AUC-MM	AUC-MMWS
1	CM1	Ant-1.7	**0.746**	0.69
2	MW1	Ant-1.7	0.707	**0.74**
3	PC1	Ant-1.7	0.641	**0.708**
4	PC3	Ant-1.7	**0.748**	0.743
5	PC4	Ant-1.7	**0.734**	0.715
6	CM1	Camel-1.6	0.549	**0.558**
7	MW1	Camel-1.6	0.536	**0.629**
8	PC1	Camel-1.6	0.55	**0.617**
9	PC3	Camel-1.6	0.549	**0.622**
10	PC4	Camel-1.6	**0.589**	0.544
11	CM1	Ivy-2.0	**0.761**	0.708
12	MW1	Ivy-2.0	0.71	**0.772**
13	PC1	Ivy-2.0	0.603	**0.769**
14	PC3	Ivy-2.0	**0.756**	0.723
15	PC4	Ivy-2.0	**0.729**	0.687
16	CM1	Jedit-4.3	0.689	**0.711**
17	MW1	Jedit-4.3	0.693	**0.707**
18	PC1	Jedit-4.3	0.674	**0.743**
19	PC3	Jedit-4.3	**0.678**	0.67
20	PC4	Jedit-4.3	0.7	**0.703**
21	CM1	Synapse-1.2	0.635	**0.648**
22	MW1	Synapse-1.2	**0.676**	0.667
23	PC1	Synapse-1.2	0.656	**0.685**
24	PC3	Synapse-1.2	**0.665**	0.632
25	PC4	Synapse-1.2	0.65	**0.711**
26	CM1	Velocity-1.6	**0.68**	0.663
27	MW1	Velocity-1.6	0.593	**0.68**
28	PC1	Velocity-1.6	0.599	**0.679**
29	PC3	Velocity-1.6	0.596	**0.686**
30	PC4	Velocity-1.6	0.579	**0.67**
31	CM1	Xalan-2.4	0.602	**0.684**
32	MW1	Xalan-2.4	0.611	**0.726**
33	PC1	Xalan-2.4	0.6	**0.717**
34	PC3	Xalan-2.4	0.663	**0.716**
35	PC4	Xalan-2.4	0.607	**0.703**
	Average		0.65	**0.686**

machine learning models LR, NB, and KNN are used as the prediction models, and the experimental results are shown in Table 9. The standard deviations and average values of AUC for different experiments are shown in the last two rows. Compared with the three mainstream defect prediction algorithms, the standard deviation value of MMWS for all the target projects is the lowest, which shows better stability in defect prediction. In addition, the experimental results show that the prediction performance of MMWS is also the best. As can be seen from the last row, although the prediction results of the proposed method are not always the best when using different machine learning models. However, its best performance on the LR model is 0.686, which is 1.9% (0.686–0.667), 4.4% (0.686–0.642), and 3.9% (0.686–0.647) higher than the average best AUC values of all the comparison algorithms. The reason is that the other mainstream methods only extract part of the metric or samples to adjust the data distribution of the source and target projects. For the proposed method (MMWS), on the one hand, metric matching strategy can characterize the metric information maximally while solving the problem of metric inconsistency and redundancy, so as to transform the heterogeneous prediction problem into a homogeneous problem. On the other hand, the weight setting strategy further improves the data distribution similarity between the source and target project, which facilitates the construction of a more general and accurate prediction model. The experimental results of the first two groups also verify the feasibility and effectiveness of these two strategies. Overall, the proposed method shows better performance in the CCDP, with higher prediction accuracy and stronger prediction stability.

4.5. Threats to Validity. Experimental results prove that the method proposed in this study performs better in the cross-company defect prediction, but there are still some factors and potential threats that affect the method validity:

First, it is difficult to obtain large-scale project datasets with defect labels, here only some datasets in NASA and PROMISE are used for comparison experiments. More cross-company software defect datasets should be used in the future to further verify the availability and stability of the cross-company defect model.

Table 9: Performance verification results of mainstream methods.

Source	Target	KNN model				NB model				LR model			
		FMT	HDP	RM	MMWS	FMT	HDP	RM	MMWS	FMT	HDP	RM	MMWS
CM1	Ant-1.7	**0.713**	0.654	0.664	0.555	**0.756**	0.646	0.688	0.663	0.703	**0.826**	0.660	0.690
MW1	Ant-1.7	0.728	0.722	**0.756**	0.701	0.812	**0.816**	0.685	0.630	0.710	**0.768**	0.641	0.740
PC1	Ant-1.7	**0.743**	0.675	0.717	0.618	**0.778**	0.582	0.598	0.612	**0.772**	0.651	0.494	0.708
PC3	Ant-1.7	**0.745**	0.708	0.704	0.676	**0.751**	0.732	0.604	0.702	0.715	**0.743**	0.392	0.743
PC4	Ant-1.7	0.616	0.560	**0.670**	0.659	**0.701**	0.576	0.553	0.584	0.634	0.579	0.506	**0.715**
CM1	Camel-1.6	**0.606**	0.588	0.547	0.589	**0.611**	0.585	0.588	0.538	**0.636**	0.568	0.573	0.558
MW1	Camel-1.6	0.584	0.546	**0.594**	0.582	0.592	0.597	0.596	**0.600**	0.624	0.610	0.555	**0.629**
PC1	Camel-1.6	0.529	0.542	**0.576**	0.567	**0.596**	0.489	0.542	0.583	0.572	0.535	0.498	**0.617**
PC3	Camel-1.6	0.580	0.571	0.584	**0.586**	**0.606**	0.587	0.547	0.573	0.615	0.594	0.455	**0.622**
PC4	Camel-1.6	0.552	0.467	**0.569**	0.567	0.519	0.422	0.534	**0.571**	0.498	0.414	0.517	**0.544**
CM1	Ivy-2.0	**0.691**	0.649	0.665	0.641	**0.792**	0.706	0.694	0.643	0.578	**0.738**	0.661	0.708
MW1	Ivy-2.0	0.749	0.607	**0.764**	0.706	**0.807**	0.700	0.695	0.688	0.496	0.717	0.619	**0.772**
PC1	Ivy-2.0	0.690	0.556	**0.719**	0.641	0.682	0.518	0.602	**0.723**	0.506	0.514	0.517	**0.769**
PC3	Ivy-2.0	0.684	0.658	**0.715**	0.682	0.603	0.669	0.649	**0.740**	**0.765**	0.732	0.405	0.723
PC4	Ivy-2.0	0.625	0.605	**0.655**	0.612	0.636	0.540	0.595	**0.669**	0.637	0.534	0.516	**0.687**
CM1	Jedit-4.3	0.594	0.538	0.628	**0.717**	**0.667**	0.639	0.561	0.545	0.569	0.578	0.552	**0.711**
MW1	Jedit-4.3	0.612	**0.719**	0.626	0.613	0.611	**0.626**	0.543	0.606	0.614	0.616	0.537	**0.707**
PC1	Jedit-4.3	0.594	0.547	0.601	**0.602**	0.563	0.537	0.567	**0.661**	0.592	0.460	0.524	**0.743**
PC3	Jedit-4.3	0.598	0.591	0.580	**0.647**	0.585	0.603	0.514	**0.636**	0.593	0.637	0.477	**0.670**
PC4	Jedit-4.3	0.520	0.545	0.540	**0.579**	0.514	0.551	0.521	**0.555**	0.522	0.555	0.511	**0.703**
CM1	Synapse-1.2	**0.646**	**0.646**	0.605	0.586	**0.666**	0.653	0.638	0.619	**0.740**	0.719	0.655	0.648
MW1	Synapse-1.2	0.655	0.655	**0.693**	0.603	0.715	**0.729**	0.651	0.633	0.634	**0.728**	0.583	0.667
PC1	Synapse-1.2	0.656	0.660	**0.661**	0.628	0.582	**0.646**	0.571	0.639	0.446	**0.703**	0.494	0.685
PC3	Synapse-1.2	0.666	**0.680**	0.649	0.658	0.580	**0.691**	0.600	0.575	0.573	**0.700**	0.410	0.632
PC4	Synapse-1.2	**0.633**	—	0.630	0.617	**0.674**	—	0.563	0.618	0.591	—	0.542	**0.711**
CM1	Velocity-1.6	0.647	0.605	0.600	**0.657**	**0.714**	0.621	0.624	0.569	**0.703**	0.627	0.611	0.663
MW1	Velocity-1.6	0.643	0.623	**0.669**	0.629	**0.736**	0.734	0.669	0.626	0.722	**0.729**	0.566	0.680
PC1	Velocity-1.6	0.577	**0.650**	0.642	0.614	**0.748**	0.576	0.579	0.570	**0.767**	0.643	0.495	0.679
PC3	Velocity-1.6	0.663	**0.721**	0.634	0.657	0.699	**0.719**	0.599	0.656	0.729	**0.736**	0.431	0.686
PC4	Velocity-1.6	0.562	0.546	0.584	**0.628**	**0.615**	**0.615**	0.551	0.545	0.585	0.604	0.515	**0.670**
CM1	Xalan-2.4	0.662	0.621	0.657	**0.687**	0.666	0.520	**0.671**	0.648	0.608	0.665	0.635	**0.684**
MW1	Xalan-2.4	0.683	0.639	**0.726**	0.644	**0.741**	0.723	0.677	0.634	0.623	0.721	0.607	**0.726**
PC1	Xalan-2.4	0.675	0.627	**0.687**	0.651	**0.717**	0.601	0.600	0.682	0.713	0.625	0.513	**0.717**
PC3	Xalan-2.4	0.670	0.671	**0.679**	0.590	**0.667**	0.667	0.605	0.591	0.637	0.679	0.392	**0.716**
PC4	Xalan-2.4	0.593	0.563	**0.648**	0.507	**0.651**	0.584	0.560	0.566	0.562	0.589	0.525	**0.703**
Std		0.084	0.093	0.076	**0.051**	0.060	0.063	0.057	**0.046**	0.080	0.081	0.053	**0.052**
Average		0.640	0.616	**0.647**	0.626	**0.667**	0.624	0.601	0.620	0.628	0.642	0.531	**0.686**

Second, although the proposed method maintains the AUC value above 0.62 on different models, different machine learning models will have a certain impact on the prediction performance. Hence, more machine learning methods will be applied to choose a more suitable prediction model construction method. Moreover, it is necessary to explore different metric matching and data distribution transformation strategies to further improve the versatility and accuracy of the prediction model.

Finally, in many cases, there are fewer defective samples than nondefective samples in defect dataset. This phenomenon will lead to class imbalance problem and may affect the performance of the prediction model, so the next step will try to improve the overall prediction performance from the perspective of how to solve the class imbalance problem.

5. Conclusions and Future Work

In this study, a new CCDP method based on metric matching and sample weight setting is proposed to further improve the defect prediction performance, thereby improving the software security and reliability. The main contributions are as follows:

(1) A clustering-based metric matching algorithm is proposed first. The multigranularity metric feature vector is extracted to unify metric dimension. Moreover, metric clustering is applied to eliminate the metric redundancy problem, and the representative metric is extracted to facilitate the subsequent one-to-one metric matching. This method not only unifies the metrics and eliminates the impact of redundant metrics but also has no restrictions on the scale of the source and target projects. Furthermore, this method approximately converted the heterogeneity problem in CCDP into a homogeneous issue, which has certain reference value for solving heterogeneous situations in other fields.

(2) A sample selection-based weight setting algorithm is proposed to reduce the differences in data

distribution of different projects. Based on the metric matching results, the selection frequency information of source samples is obtained through the metric similarity measure as an influence factor to increase the weight of source samples that are more similar to target samples. This can further improve the data distribution similarity between the source and target projects, thereby improving the prediction accuracy.

(3) Based on the above key technologies, extensive experiments are conducted to demonstrate the feasibility and effectiveness of the proposed strategies and overall method. Experimental results prove that the proposed method has superior prediction performance over other mainstream CCDP methods.

However, there are still some open problems, such as only part of NASA and PROMISE datasets, are used for performance verification. In the future, more defect datasets will be collected to verify effectiveness of this method. Moreover, the metric clustering operation does not consider the impact of irrelevant metrics in the project when constructing the prediction model, which may affect the defect prediction accuracy to a certain extent. Those issues will also be solved in the future work to further improve the security and reliability of large-scale software.

Data Availability

Data will be available in the following link: https://www.researchgate.net/search/publication?q=NASA%20MDP.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors appreciate the support from the Zhejiang Provincial Natural Science Foundation of China (LY20F020015 and LY21F020015), the National Science Foundation of China (61702517, 61972121, 61902345, and 61772525), the Defense Industrial Technology Development Program (no. JCKY2019415C001), the Open Project Program of the State Key Lab of CAD&CG (Grant no. 2109), and Zhejiang University.

References

[1] N. J. Pizzi and J. Nick, "A fuzzy classifier approach to estimating software quality," *Information Sciences*, vol. 241, pp. 1–11, 2013.

[2] L. He, Q. B. Song, and J. Y. Shen, "Boosting-based k-NN learning for software defect prediction," *Moshi Shibie yu Rengong Zhineng/Pattern Recognition and Artificial Intelligence*, vol. 25, pp. 792–802, 2012.

[3] Z. Xu, S. Li, J. Liu et al., "LDFR: learning deep feature representation for software defect prediction," *Journal of Systems and Software*, vol. 158, 2019.

[4] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "BPDET: an effective software bug prediction model using deep representation and ensemble learning techniques," *Expert Systems with Applications*, vol. 144, Article ID 113085, 2019.

[5] L. Zhao, Z. Shang, L. Zhao, T. Zhang, and Y. Y. Tang, "Software defect prediction via cost-sensitive siamese parallel fully-connected neural networks," *Neurocomputing*, vol. 352, pp. 64–74, 2019.

[6] M. Tim, M. Zach, T. Burhak, C. Bojan, J. Yue, and B. Ayse, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, pp. 375–407, 2010.

[7] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[8] X. Yang, S. Zhou, and M. Cao, "An approach to alleviate the sparsity problem of hybrid collaborative filtering based recommendations: the product-attribute perspective from user reviews," *Mobile Networks and Applications*, vol. 25, pp. 376–390, 2019.

[9] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, 2017.

[10] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the Joint Meeting of the European Software Engineering Conference & the Acm Sigsoft Symposium on the Foundations of Software Engineering*, Amsterdam, Netherlands, August 2009.

[11] F. Rahman et al., "How, and why, process metrics are better," in *Proceedings of the 35th International Conference on Software Engineering*, pp. 432–441, ICSE), Francisco, CA, USA, May 2013.

[12] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[13] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 35th International Conference Software Engineering (ICSE)*, May 2013.

[14] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zheng, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, 2018.

[15] C. Jinyin, K. Hu, Y. Yang, Y. Liu, and Q. Xuan, "Collective transfer learning for defect rediction," *Neurocomputing*, vol. 416, pp. 103–116, 2020.

[16] Z. Sun, J. Li, H. Sun, and L. He, "CFPS: collaborative filtering based source projects selection for cross-project defect prediction," *Applied Soft Computing*, vol. 99, Article ID 106940, 2020.

[17] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Systems with Applications*, vol. 117, Article ID 114637, 2021.

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

[19] S. L. Liu, X. Chen, and W. S. Liu, "FECAR : A feature selection framework for software defect prediction," in *Proceedings of the Annual Computer Software and Applications Conference*, pp. 426–435, Vasteras, Sweden, July 2014.

[20] Q Wang, J. Zhu, and B. Yu, *Feature Selection and Clustering in Software Quality Prediction*, Ease, East Sussex, United Kingdom, 2007.

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

[22] X. Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, 2016.

[23] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[24] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proceedings of the Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on IEEE*, San Francisco, CA, USA, May 2013.

[25] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[26] X. Jing, W. Fei, D. Xiwei, Q. Fumin, and X. Baowen, "ACM Press the 2015 10th Joint Meeting - Bergamo, Italy (2015.08.30-2015.09.04)] Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015 - Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," *Joint Meeting*, pp. 496–507, 2015.

[27] M. Ying, Z. Shunzhi, C. Yumin, and L. Jingjing, "Kernel CCA based transfer learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. 8, pp. 1903–1906, 2017.

[28] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, 2017.

[29] Q. Yu, S. Jiang, and Y. Zhang, "A feature matching and transfer approach for cross-company defect prediction," *Journal of Systems and Software*, Article ID S0164121217301346, 2017.

[30] D. Whitlark and G. H. Dunteman, "Principal components analysis," *Journal of Marketing Research*, vol. 27, no. 2, p. 243, 1990.

[31] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, pp. 67–77, 2015.

[32] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, 2002.

[33] D. Lewis, *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval*, Springer, New York, NY, USA, 1998.

[34] W. Afzal, "Using faults-slip-through metric as a predictor of fault-proneness," in *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, Sydney, Australia, December 2011.

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

[36] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in *Proceedings of the 2009. ISSRE '09. 20th International Symposium on Software Reliability Engineering*, Bengaluru-Mysuru, India, November 2009.

[37] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[38] T. J. Mccabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308–320, 2006.

[39] I. Chowdhury and M. Zulkernine, *Using Complexity, Coupling, and Cohesion Metrics as Early Indicators of Vulnerabilities*, Queen's University, Doctoral dissertation, Kingston, Canada.

[40] S. Moshtari, A. Sami, and M. Azimi, "Using complexity metrics to improve software security," *Computer Fraud & Security*, vol. 2013, no. 5, pp. 8–17, 2013.