# Using Cost-cognitive Bagging Ensemble to Improve Cross-project Defects Prediction

Yong Li[1,2,3*], Ming Wen[2], Zhandong Liu[1], Haijun Zhang[1]

[1] College of Computer Science and Technology, Xinjiang Normal University, China
[2] Xinjiang Electronic Research Institute Limited by Share Ltd, China
[3] Key Laboratory of Safety-Critical Software (NUAA), Ministry of Industry and Information Technology, China
liyong@live.com, wmconet@126.com, lzd0825@163.com, zhjlp@163.com

## Abstract

Cross-project defect prediction (CPDP) is a field of study that allows predicting defects in software projects for which the availability of data is limited and produces generalizable prediction models. Due to the heterogeneity of cross projects, CPDP is particularly challenging and several methods have been employed to address this problem. Nevertheless, the class-imbalanced characteristic of the cross-project defect data also increases the learning difficulty of such a task but has not been investigated in depth. This paper proposed a novel, cost-cognitive ensemble method for CPDP, which includes four phases: bagging balanced resampling phase, base classifiers learning phase, cost value cognitive phase, and base classifiers ensemble phase. These phases create a composition of classifiers that are used for predicting defects. Results of an empirical evaluation on 10 datasets from the PROMISE repository indicated that our method achieves the best overall performance with respect to conventional methods. Moreover, our method could cognize the cost value automatically during the model training, it is shown to be more effective and practical.

**Keywords:** Cross-project defects prediction, Class imbalanced data, Bagging ensembles, Cost-cognitive learning

## 1 Introduction

Software defects prediction is one of the most common research topics in empirical software engineering. It can be utilized for helping developers focus on activities such as code inspection or testing on likely defect-prone modules, thus optimizing the usage of resources for software quality assurance [1] to reduce the risks of critical system failure [2-3]. Most software defects prediction approaches are based on software metrics by means of machine learning to build a prediction model and are evaluated in within-project prediction settings [4]. However, software engineering is inherently a competitive and protean business, changing rapidly in response to changes in markets, hardware, and software platforms. When a new project starts or an old project is completely rewritten, this is a challenge for within-project defect prediction [5]. To work around this issue, researchers have turned toward cross-project defect prediction.

Several researchers have discussed the possibility to use other finished projects' data to train models for target projects that have limited availability data. This strategy is referred to as cross-project defects prediction (CPDP). However, CPDP often yields poor performance, and its reliable prediction is still an open issue [6-7]. The reasons are mainly due to the projects' heterogeneity. A noticeable way to overcome the data distribute difference among projects is to perform data selection, and several data selection strategies have been proposed by researchers in recent years [8].

It is worth noting that works in CPDP usually have focused only on the data selection methods and ignored the other key factor affecting the performance of the CPDP model, namely, class imbalance. In fact, there is still a class imbalance problem that the defective-free modules significantly outnumber the defective modules in both the filtered training data and the target testing data. Traditional machine learning algorithms may be biased towards the majority class. Thus, they may produce poor predictive accuracy for the minority class [9]. However, there are few studies that have taken into consideration the important characteristic of the CPDP problem.

In this study, we focus on the solutions of class imbalance in CPDP and seek empirical evidence that they can achieve acceptable results. Our main contributions are summarized as follows: (1) We proposed a cost-cognitive bagging ensemble (CCBE) approach for CPDP, which addresses the class imbalance problem by using the resampling technique to build cost-cognitive classifiers and using the ensemble technique to capture better generalizable properties in each classifier. (2) We also validated the proposed method by a large number of experiments, and found that it can achieve better performance than the conventional ones, and shows better stability and flexibility in CPDP, especially for highly imbalanced datasets.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the CCBE approach proposed. Section 4 and Section 5 are devoted to the experimental setups and results analysis. Section 6 discusses the threats to validity. Finally, Section 7 concludes the paper and presents the future work.

## 2 Related Work

### 2.1 Cross-project Defects Prediction

*Corresponding Author: Yong Li; E-mail: liyong@live.com
DOI: 10.53106/160792642022072304013

CPDP is used to predict defect-prone software modules based on the data collected from other software projects. These data can either be collected from completed development projects or taken from software repositories, such as PROMISE and so on [10]. In the last decade, a lot of studies have been put to define approaches for cross-project defects prediction.

To the best of our knowledge, the earliest work on cross-project prediction is by [11]. They built a CPDP model by using another open-source project's data and logistic regression classifier, and validated that the model outperforms the random model. However, the both projects were developed by the same team, which is an assumption that usually does not hold for cross-project defect prediction. Zimmermann et al. [12] conducted a large study to determine the feasibility and challenges of cross-project defect prediction. They found that the CPDP model that was directly trained on one or a set of projects might not be generalized well for other projects. The study also pointed out that CPDP was a serious challenge, and more attention should be paid to this problem.

To overcome the data distribution difference between source and target projects, transfer learning techniques have been proposed. Transfer learning addresses these issues by transferring knowledge extracted from the source projects to the target project for building more accurate CPDP models [13]. The CPDP approaches based on transfer learning can be classified as either feature-based or instance-based approaches [14-15].

Feature-based approaches propose to transform the cross-project data such that the underlying distributions are similar, and many existing models can be reused for CPDP, such as feature compensation approach [16], TCA+ approach [17], simplified metric-set approach [18], and so on. However, these approaches usually adopt single-source data as source projects. Especially when cross-project data are distributed with large differences, they are limited in the improvement of CPDP performance.

Instance-based approaches realize CPDP through capturing appropriate source project modules that will work for the target project. A study by [19] found that with suitable training data, the success rate of cross-project defect prediction could be drastically improved to over 50%. However, this is a post-facto approach, and thus not intended for practical prediction settings. How to select training data from multi-source projects data is an empirical issue. Consequently, most studies have focused on the data filtering approaches for CPDP and proposed some strategies, such as hierarchical select-based filter [8], target project data guided filter [20], source project data guided filter [21], data characteristics-based filter [22], local clusters-based filter [23], and so on. Since instance-based approaches can capture the more suitable model data for the target project from multi-source project data, it has drawn the attention of many researchers in recent years.

All these studies suggested that CPDP is particularly challenging, and due to the heterogeneity of projects, prediction accuracy might be poor. However, the aforementioned CPDP models take no consideration of the class-imbalanced characteristic between the defect and non-defect classes of the data set.

## 2.2 Class Imbalance Learning

In classification prediction, class imbalance means that the number of samples in one class is far less than that in other classes. The class imbalance problem also appears in software defects prediction. In most cases, the software defect data contain much fewer defective modules than the defect-free modules, and this ratio is lower in extreme cases. In general, the minority defective class is usually called positive class, while the majority defect-free class is called negative class correspondingly [24-25].

Class imbalance is an important factor that affects machine learning performance, especially for the positive class identification. In the presence of imbalanced class data, the traditional machine learning method biases easily to the negative class. As a consequence, the classification accuracy of the negative class overwhelms that of the positive class. A serious class imbalance may even lead to the classifier completely losing its classification capability, classifying data samples all into the negative class [26]. However, the goal of software defects prediction is to find more defect-prone software modules, that is, positive samples.

Several approaches have been developed trying to address class imbalance problems for defect prediction. These previous approaches can be categorized into two groups: data level techniques and algorithm level techniques, which depend on how they deal with class imbalance [27-28]. Data level techniques add a sampling step where the data is re-balanced in order to decrease the effect of the imbalanced class distribution. Algorithm level techniques incorporate different misclassification cost to take into account the significance of positive examples in the learning phase.

Since the under-sampling technique may drop some important information, and oversampling may lead to over-fitting of the model, it is generally believed that the sampling methods that alter the original defects class distribution may result in poor generalization of the model [29]. Therefore, the cost-sensitive learning is considered to be more suitable for software defects prediction, but the determination of cost factor is a difficult problem [24, 30].

Considering that there exists a class imbalance in both filtered data and target data in CPDP, combining class imbalance learning to build models is helpful to improve the performance of CPDP. However, how to deal with the imbalance of data and get a better performance model under the premise of using filtered data defect information has not been investigated in depth so far. So, we study the issue of if and how the approach can benefit CPDP with the aim of finding better solutions.

# 3 Problem Solution

## 3.1 The Proposed Approach Overview

In the cross-project defect prediction, an important factor affecting the performance of the model is the defects distribution heterogeneity between the source project and the target project, i.e., data shift [31]. Due to the data shift phenomenon in software defect data, if we only select a single source project data, it may not be enough to reflect the defect patterns of the target project [32]. Therefore, the focus of this paper is to filter the multi-source project data to obtain the CPDP model data, and then build the model.

The task of CPDP using a data filter can be defined as follows. We have $K$ source projects $S_s = \{ S_{s(1)}, S_{s(2)}, S_{s(3)}, \ldots, S_{s(k)} \}$ and

a target project $S_t$. Each source project contains many modules, and each module has two parts: a set of metrics $x$ and a label $y$ that corresponds to the defect information. For unlabeled modules in the target project $S_t$, the goals of data filter are to select suitable training data $D_{train}$ from the source projects $S_s$ and build a model for the target project $S_t$ with unknown labels.

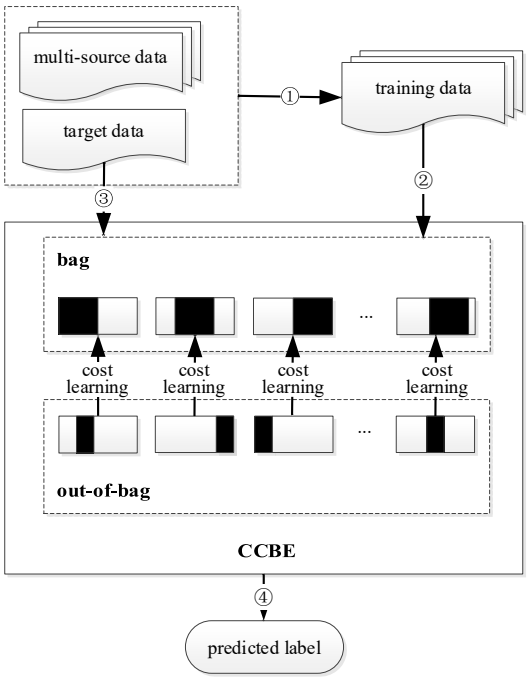



**Figure 1.** The CCBE approach framework.

Figure 1 presents the CPDP overall architecture using the proposed CCBE algorithm, which includes three steps:

(1) The first step is filtering data for the model building with the goal of reducing the impact of data shift between the source projects and the target project.
(2) Then, the CCBE algorithm is used to train the prediction model, which aims to relatively fully learn the defect patterns that are suitable for the target project in the filtered multi-source projects.
(3) Finally, the trained integrated model is used to predict whether the target module is defective or non-defective and the output is its defect label.

## 3.2 Data Preprocessing

It should be noted that the focus of this paper is the class imbalance problem in CPDP. Considering that the proposed CCBE algorithm is independent of the data filtering algorithm, although different data filtering algorithms have been proposed in previous studies, we chose the most commonly used TGF strategy to filter multi-source data in the experiment [20, 33].

The TGF provides a filtering strategy for the model training data based on the *K*-nearest neighbor algorithm. For each module in the target project, TGF strategy selects $k$ (The authors recommend $k$ =10) modules of the available data closest to the target module as the training data. Turhan et al. [20] found that CPDP using the TGF strategy can compete with WPDP in some cases. The pseudo code of TGF strategy is shown in Algorithms 1.

**Algorithm 1**. The TGF algorithm

**Input:**

Multi-source projects data: $S_s = \{S_{s(1)}, S_{s(2)}, \cdots, S_{s(k)}\}$ ;

Target project data: $S_t$ ;

**Output:**

Training data for target project, $D_{train}$ ;

$D_{train} \leftarrow \varnothing$ ;

$S_{mix} \leftarrow S_{s(1)} \cup S_{s(2)} \cdots \cup S_{s(k)}$ ;

**for** all modules $M_{t(i)} \in S_t$ **do**

  $D_{temp} \leftarrow$ search KNN modules from $S_{mix}$ ;

  $D_{train} \leftarrow D_{train} \cup D_{temp}$ ;

**end for**

**return** $D_{train}$ ;

## 3.3 The CCBE Algorithm

The goal of CCBE algorithm is to more fully obtain the defect patterns suitable for the target data from the filtered model data without paying attention to the data source. In the following section, we describe the key technology and pseudo code of the algorithm.

### 3.3.1 Cost Sensitive Software Defect Prediction

The prediction results of the CPDP model for the target software module may lead to two types of errors:

- The type I error is to classify the defective module as defective-free, which may lead to some defects that cannot be repaired, resulting in software failure and software unreliability.
- The type II error is to classify the non-defective module as defective-prone, which may lead to useless testing and waste of development resources.

The cost of type I error and type II error can be expressed as $Cost(1,0)$ and $Cost(0,1)$ respectively, where the defect-prone module is usually marked as "1", while the defect-free module is marked as "0". According to the practical experience of software engineering, it is obvious that when the model is mispredicted, $Cost\ (1,0) > Cost\ (0,1)$, that is, when $Cost\ (1,0) = \alpha$ and $Cost\ (0,1) = 1$, $\alpha > 1$. When the model is correctly predicted, $Cost\ (1,1) = Cost\ (0,0) = 0$. Therefore, considering the cost of different error predictions, cost-sensitive software defect prediction may be more effective in practice.

For the CPDP model, the prediction cost function can be defined as $L$, whose value is the sum of predicted costs of different classes. Therefore, the function $L(x)$ can be formally expressed as:

$$L(x) = L(x,1) + L(x,0)$$

$$L(x,1) = Prob(x,0) \times C(0,1) + Prob(x,1) \times C(1,1)$$

$$L(x,0) = Prob(x,1) \times C(1,0) + Prob(x,0) \times C(0,0)$$

The function $L(x)$ represents the expected cost of defect prediction for module $x$, that is, minimizing $L(x,i)$ is equivalent to selecting the optimal defect prediction label, where $i \in \{0,1\}$, and $Prob(x,c=i)$ represents the probability of predicting module $x$ as class $i$.

Due to the practical requirements of software defect prediction, the cost factor $\alpha > 1$. If the cost value is too high,

the prediction rate and the false alarm rate will decrease synchronously. If the cost value is too low, the cost of different prediction errors is not obvious in the prediction model. Therefore, the most appropriate cost value should be determined according to the practical need.

### 3.3.2 The Cost-cognitive Process by Bootstrap Resampling

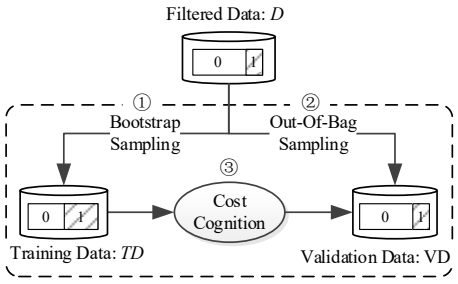



**Figure 2.** The Cost-cognitive process

Figure 2 gives the cost-cognitive process using bootstrap resampling for the CCBE algorithm. In the figure, *D* represents the filtered data, in which the number of defective modules is recorded as *p*, and the number of non-defective modules is recorded as *n*. Due to the class imbalance of software defect data, generally $p < n$. The process of cost learning is described as follows:

(1) Class-balance model data sampling. Firstly, the software modules marked as "1" and "0" in *D* are sampled *p* times respectively, and then the sampled data are combined into a class balanced training dataset *TD*.

(2) Class-imbalance verification data sampling. After the defective modules in *D* are sampled *p* times, about 36.8% of them are not selected into *TD*, and these modules are recorded as *VDP*. Then, *m* modules are sampled again from the defect-free modules that have not been sampled and recorded as *VDN*, where $m=|VDP|\times(n/p)$. Finally, combined *VDP* and *VDN* as verification set *VD*.

(3) Cost-cognitive using the verification data. First, the balanced data *TD* is used to train the model, and then the imbalanced verification data *VD* is used to obtain the optimal cost value based on a certain model performance metric.

In the process of the above method, the intersection of the training set and the verification set is empty, which can effectively avoid the overfitting of the model and improve the generalization performance of the prediction model.

### 3.3.3 The CCBE Algorithm Pseudo Code

We use the bagging method to implement training and cost learning for the base classifier in the CCBE algorithm and use the voting method to achieve prediction. In ensemble learning, the accuracy and difference of base classifiers are important factors affecting the performance of ensemble model [34]. Because the rules generated by the decision tree algorithm are easy to understand and the classification speed is fast, we choose the *C4.5* algorithm to build the base classifier in the CCBE algorithm. Algorithm 2 shows the CCBE algorithm pseudo code.

**Algorithm 2**. The CCBE algorithm

**Input:**

Filtered data: $D=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$

Base-classifier algorithm: $L$

Number of base-classifiers: $T$

**Output:**

Ensemble classifier: $H(x)=sign(\sum_{t=1}^{T}h_t(x))$,

where $h_t$ is the base classifier, $h_t \in [1,0]$.

$P \leftarrow$ Get the defective modules from $D$, $P \subseteq D$

$N \leftarrow$ Get the non-defective modules from $D$, $N \subseteq D$

Compute the imbalance rate: $ir \leftarrow |P|/|N|$

**for** $t=1$ to $T$ **do**

  Bootstrap sample the defective module subset $P_t$ from $P$ with $|P_t|=|P|$

  Bootstrap sample the non-defective module subset $N_t$ from $N$ with $|N_t|=|P|$

  Combine $P_t$ and $N_t$ as the base-classifier training data: $TD_t \leftarrow \{P_t, N_t\}$

  Get the out-of-bag defective modules: $P_{oob(t)} \leftarrow P - P_t$

  Bootstrap sample the non-defective module subset $N_{oob(t)}$ from $N - N_t$ with $|N_{oob(t)}| \leftarrow |P_{oob(t)}|/ir$

  Combine $P_{oob(t)}$ and $N_{oob(t)}$ as the model validation data: $VD_t \leftarrow \{P_{oob(t)}, N_{oob(t)}\}$

  **for** $cost=1$ **to** $C$ do

    $h_t \leftarrow L(TD_t, cost)$

    Evaluate $h_t$ on $VD_t$, and record the cost value which achieves the highest performance

  **end for**

  $h_t \leftarrow L(TD_t, \alpha)$

**end for**

# 4 Empirical Study

## 4.1 Datasets

For our experiments, we used 10 open-source projects for validation. These datasets are part of the public PROMISE repository. The original datasets were collected by Jureczko and Madeyski with the Ckjm and BugInfo tools [35], in which each instance represents a class of the release and consists of two parts: 20 independent static code attributes and the dependent attribute labeled "defects" indicating the defect count information.

For the work, we refer to each class as a module instance, and the defects attribute is discretized into a Boolean value where "0" indicates no defects for the class and "1" indicates otherwise. Since most of the software defect prediction research literature uses the same setting, we follow them. Detailed information on the experimental dataset is listed in Table 1.

**Table 1.** Defect dataset characteristics

| Project | Modules# | Defect-prone# | Defect-prone% |
|---|---|---|---|
| ant13 | 125 | 20 | 16% |
| arc | 234 | 27 | 12% |
| ivy11 | 111 | 63 | 57% |
| jedit32 | 272 | 90 | 33% |
| log4j10 | 135 | 34 | 25% |
| lucene20 | 195 | 91 | 47% |
| poi15 | 237 | 141 | 59% |
| synapse10 | 157 | 16 | 10% |
| velocity14 | 196 | 147 | 75% |
| xercesinit | 162 | 77 | 48% |

## 4.2 Performance Evaluation

In the experiment, the number of modules whose defect label marked as $i$ and predicted as $j$ is expressed as $n_{ij}$. Therefore, the prediction results of the target software module can be divided into four situations: True Positive (TP)=$n_{11}$, True Negative (TN)=$n_{00}$, False Positive (FP)=$n_{01}$, and False Negative (FN)=$n_{10}$.

Considering the various practice requirements, several performance measures are usually adopted to evaluate different aspects of predictors. In the existing studies, the commonly used single model metrics are PD/PF, and the comprehensive evaluation model metrics are AUC/BAL. These performance metrics are shown in Formulas 1, 2, 3, and 4 respectively.

$$PD = \frac{TP}{TP + FN} \tag{1}$$

$$PF = \frac{FP}{FP + TN} \tag{2}$$

$$AUC = \frac{1 + TP - FP}{2} \tag{3}$$

$$BAL = 1 - \frac{\sqrt{(1 - PD)^2 + PF^2}}{\sqrt{2}} \tag{4}$$

PD (Probability of detection) measures how many of the defective modules are found. PF (Probability of false alarm) measures how many of the modules that triggered the detector actually did not contain defective concept. Menzies et al. [36] claimed that a high-PD predictor is still useful in practice, even if the other measures may not be good enough.

The ROC curve illustrates the trade-off between PD and PF, which serves as the performance of a classifier across all possible decision thresholds. AUC estimates the area under the ROC curve, which is a good measure for the overall PD/PF performance. The higher the AUC is, the better the performance will be. BAL is another commonly used comprehensive measure to indicate the tradeoff between PD and PF, which is defined as the normalized Euclidean distance from the desired point (1,0) to observed (PD, PF) in a ROC curve. A larger BAL value indicates that the performance is closer to the ideal case.

In some studies, the Recall/Precision/F-value metrics that are commonly used in machine learning research are also used to evaluate the software defect prediction model. Due to the imbalanced characteristics of software defect data, we avoided the precision measure, and add the F-value with AUC and BAL to evaluate the CPDP comprehensive performance.

## 4.3 Experimental Design

The experiment consists of three investigations. The first investigation finds out which measure is better to be the criterion for choosing the cost of the CCBE approach, while the second investigation explores the relationship between the cost factor and the imbalance rates in our approach. The third investigation explores whether our proposed approach has advantages compared with conventional ones. More specifically, we try to find empirical evidence to answer the following questions.

### 4.3.1 RQ1: Which Measure Is Better to Be the Criterion for Choosing the Cost of CCBE Approach?

This investigation intended to find out which measure is suitable for cost learning. As a matter of fact, it is hard to determine the cost value when applying the cost-sensitive learning method to defect prediction, since the best cost parameter is always problem- and algorithm- dependent. Generally, there is competition among the single evaluation metrics of the model. For example, when the cost value is too high, PD and PF may decrease simultaneously, and vice versa. These trade-offs make it difficult to compare the performances of several prediction models by using only either PD or PF.

Therefore, we usually choose the value that makes the CPDP model have the best comprehensive performance as the optimal cost value, which is also conducive to the experimental comparison and selection of models. For this purpose, we used AUC, BAL, and F-values as the criterion for determining the optimal cost value of class imbalance learning methods in the CCBE algorithm.

### 4.3.2 RQ2: What is the Relationship between the Cost Factor and the Imbalance Rates in Our Proposed Method?

This investigation was to explore whether or not the proposed method can effectively deal with the class-imbalanced problem in the context of cross-project defect prediction. In fact, for different application scenarios, there are different requirements on the cost value. For example, in the security critical software, the prediction model is required to have a higher cost factor. When the software testing resources are limited, the cost factor cannot be too high. Therefore, in this section, PD/PF/AUC-based optimal learning cost factor are used to calculate PD, PF, and AUC, and to compare their changes with different imbalance rates. Because the BAL and F-values are positively correlated with AUC, only the AUC with the most stable results is selected.

From Table 1, we know that the imbalance rate of the datasets that are used in the experiments vary from 10% to 75%. This provides us with a chance to explore the relationship between the cost factor and the imbalance rates, and further investigate the effectiveness of our proposed method on dealing with the class-imbalanced problem.

#### 4.3.3 RQ3: How Does the CCBE Method Perform Compared to the Conventional Ones?

This investigation was used to explore whether our proposed method is effective or not. This was achieved by comparing our proposed CCBE method with seven conventional current state-of-the-art approaches. We investigated four single classifier algorithms and three ensemble algorithms; these are Naive Bayes (NB), Random Forest (RF), Decision Tree (DT) and SVM because of their simplicity, effectiveness, and popularity in the literature [4]. In addition, three representative ensemble algorithms are also considered in our study, namely AdaBoost (AB), EasyEnsemble (EE) [37] and Local Approach (LA) [23]. AB is the traditional ensemble algorithm, EE is the latest algorithm to deal with imbalanced data, and LA is a state-of-the-art approach that uses local prediction to mitigate the heterogeneity of cross projects.

Usually, there are trade-offs between PD and PF, which makes it difficult to compare the performances of several prediction models by using only either the PD or PF. The higher the AUC, BAL, and F-value are, the better the performance is. We compute AUC, BAL, and F-value to evaluate the performance of these 8 approaches on the 10 datasets from the PROMISE repository. For each dataset, we run CCBE and the baseline approaches 10 times and the average value is used as the final experimental result.

# 5 Results and Analysis

This section presents the experiment result. The results are structured according to the research questions presented in above section.

## 5.1 Optimal Cost Factor Learning Approach for the CCBE Algorithm

To understand which measure is better to be the criterion for choosing the cost-value of the CCBE algorithm for CPDP, we produce bar graphs in Figure 3, in which the average performance values of AUC, BAL, and F-value on the ten datasets were displayed by the three sub-plot respectively.

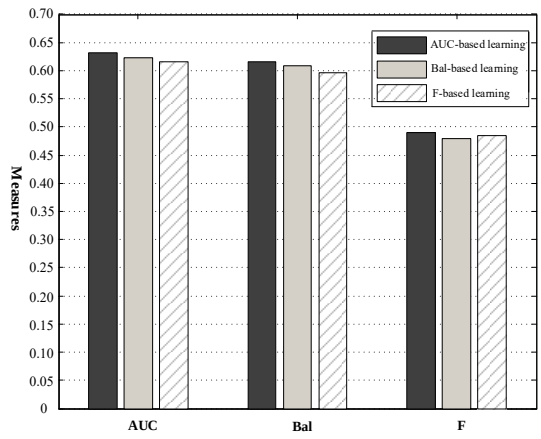



**Figure 3.** Comparison of different methods in terms of prediction performance

This result illustrates that AUC is the best choice than others for learning the cost factor parameters. In most cases, the models with cost-factor learning by the AUC-based method could obtain better comprehensive performance than the model trained based on other methods. There is evidence supporting that AUC is a more stable metric than the others [38]. Hence, different settings do not change AUC significantly, and using AUC would be more appropriate for the choice of training parameters.

Through the analysis of the above results, we can draw the following conclusions for RQ1:

(1) The AUC and BAL measures of the three cost-value learning methods are better than the F-value measure as a whole. The possible explanation is that AUC and BAL are comprehensive performance measures calculated based on PD and PF. The poor performance of F-value may be due to the low accuracy or precision. Some studies have shown that the two measures are relatively poor in the evaluation of imbalanced data [39].

(2) When the AUC, BAL, and F-value are used for model evaluation, the performance of AUC-based cost factor cognition method is better than the other two methods. Therefore, in practice, it is suggested to use the method of optimizing AUC measure to realize the cost-factor learning.

## 5.2 The Relationship between the Cost Factor and the Imbalance Rates

In this section, for convenience, PD/PF/AUC-based cost cognition methods are represented as M1, M2, and M3, respectively.

Figure 4(a) shows the relationship between the cost factor by M1/M2/M3 methods and the imbalance rate. It can be seen from the figure that the cost learning methods based on M1 and M2 obtain the maximum and minimum cost factor respectively. Moreover, M3 balances the prediction rate and false alarm rate, and its cost factor value ranges between M1 and M2. In addition, from the perspective of the relationship between the data imbalance rate and the cost value, it can be seen intuitively that the cost factor value obtained by M1/M2/M3 has no direct relationship with the data imbalance rate, that is, the cost factor value is relatively stable.

Figure 4(b) and Figure 4(c) are PD and PF values of the CPDP models respectively realized by M1/M2/M3 based on the CCBE algorithm. From this figure, it can be seen that the PD and PF values of the three models are also not affected by the change of class imbalance rate. In particular, 80% of the data sets achieved a prediction rate higher than 70%. In many research literature, 70% prediction rate is taken as a benchmark comparison value [40].

Figure 4(d) shows the AUC values of different cost perception models. From this figure, it can be seen intuitively that AUC, as a trade-off indicator of PD and PF, has relatively small differences based on M1/M2/M3. But in general, the comprehensive performance of the model based on M3 is relatively optimal, and the performance of the model is relatively stable with the change of class data imbalance rate, which also shows that the CCBE algorithm has certain advantages in dealing with class imbalance data for cross-project software.

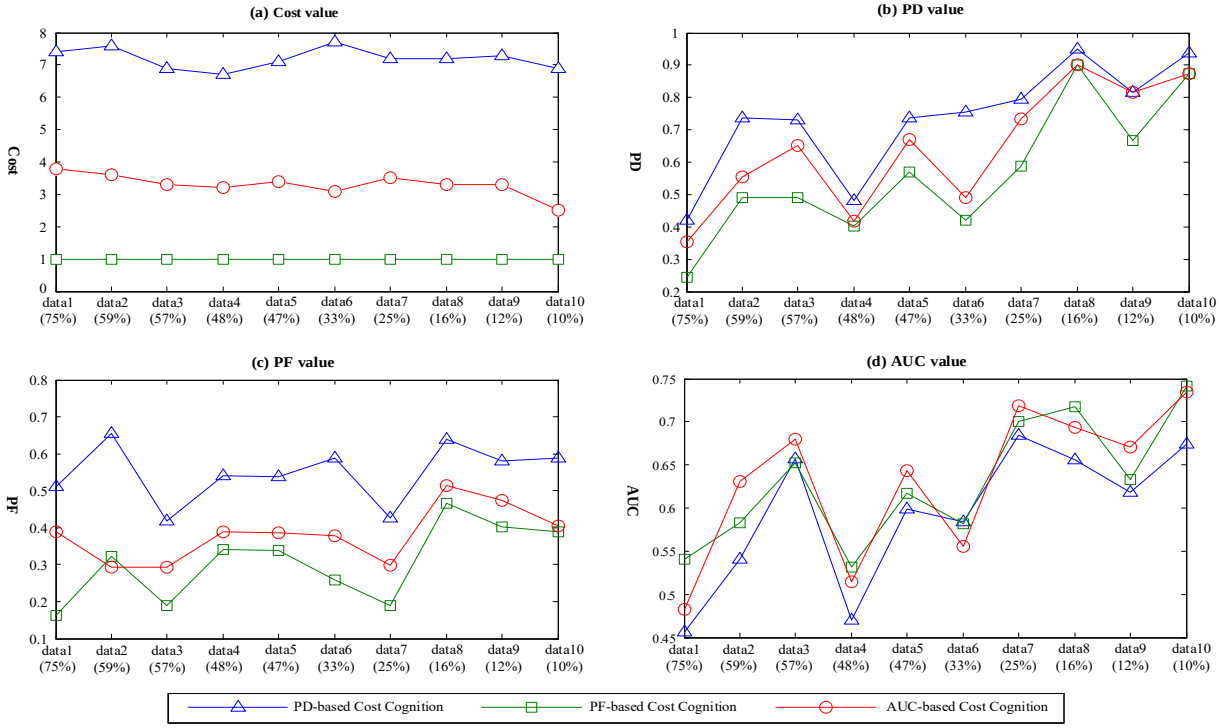



**Figure 4.** The relationship between the cost factor and the imbalance rates

Through the analysis of the above results, we can draw the following conclusions for RQ2:

(1) The proposed CCBE algorithm is relatively stable in the class imbalance data, especially in the extreme case of class imbalance. In addition, the CPDP model and cost factor can be obtained from the class balanced model data and class imbalanced cost factor validation data based on under sampling.

(2) In software engineering practice, the CCBE algorithm can be selected according to the test resources and the security requirements. From the perspective of practical validity, the prediction rate of the proposed method is higher than 60% (artificial defects prediction rate [41]), which also shows that the algorithm has better flexibility and effectiveness.

## 5.3 Comparison Results of Our Proposed Approach with the Conventional Ones

This section presents the results of our proposed approach vs. the conventional approaches CPDP experiments in order to address our third question. For each dataset, we build seven predictive models following the algorithm settings described in the previous section. It is important to note that we only use the AUC-based cost learning scheme for building the CCBE predictor. We focus on comparing the overall prediction performance (i.e., AUC, BAL, and F-value) between the CCBE and other ones. The results are summarized with mean performance values and average rank (AR) values in Table 2, Table 3, and Table 4 for AUC, BAL, and F-value, respectively. In all tables, the best approach is denoted in boldface.

In order to demonstrate the experimental results more directly, the results are also visualized by using a boxplot in Figure 5. It can be seen from Figure 5 that the proposed CCBE algorithm is better than other methods in general, and the experimental results indirectly show that AUC/BAL/F-value is effective as a comprehensive index, and they have the same performance in describing the comprehensive performance of the prediction model.

Through the analysis of the above results, we can draw the following conclusions for RQ3:

(1) The comprehensive performance of the conventional model is poor, which also shows that the traditional algorithm is poor in dealing with class imbalanced data. As the most widely used model algorithm in defect prediction, NB is second only to the CCBE and EE algorithm in imbalance data prediction.

(2) The CCBE algorithm and EE algorithm are special algorithms for dealing with class imbalanced data, and their prediction performance is relatively excellent, especially the CCBE algorithm which has better flexibility in software defect prediction and stability.

**Table 2.** The AUC values of the models built with eight learning algorithms

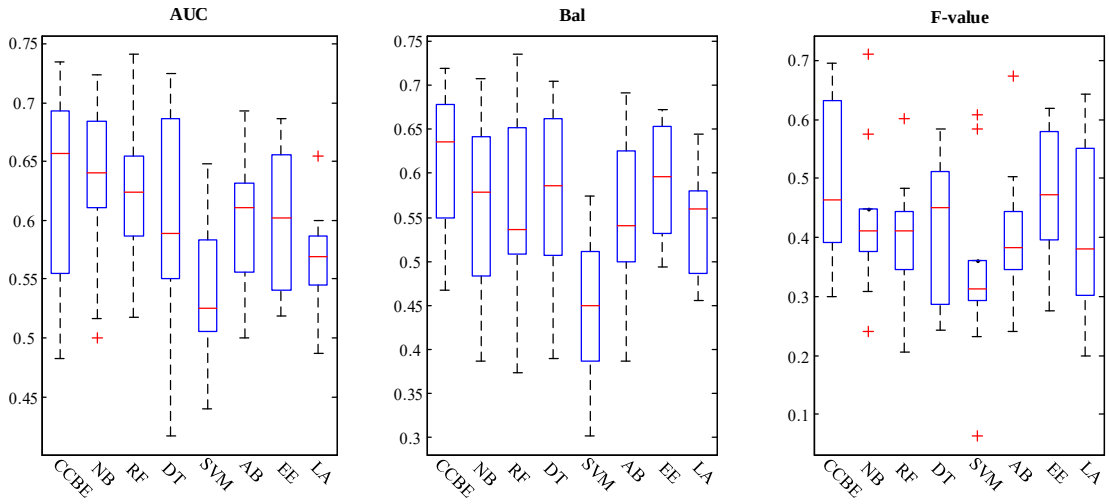
| | CCBE | NB | RF | DT | SVM | AB | EE | LA |
|---|---|---|---|---|---|---|---|---|
| ant13 | 0.693 | 0.660 | 0.655 | 0.720 | 0.569 | 0.631 | 0.686 | 0.573 |
| arc | 0.671 | 0.671 | 0.682 | 0.580 | 0.648 | 0.693 | 0.673 | 0.655 |
| ivy11 | 0.680 | 0.614 | 0.612 | 0.550 | 0.509 | 0.586 | 0.616 | 0.585 |
| jedit32 | 0.555 | 0.620 | 0.612 | 0.624 | 0.510 | 0.526 | 0.519 | 0.493 |
| log4j10 | 0.719 | 0.710 | 0.637 | 0.725 | 0.583 | 0.632 | 0.651 | 0.587 |
| lucene20 | 0.643 | 0.611 | 0.587 | 0.597 | 0.541 | 0.603 | 0.587 | 0.600 |
| poi15 | 0.631 | 0.684 | 0.653 | 0.560 | 0.506 | 0.618 | 0.572 | 0.565 |
| synapse10 | 0.735 | 0.724 | 0.741 | 0.686 | 0.617 | 0.691 | 0.656 | 0.545 |
| velocity14 | 0.483 | 0.500 | 0.537 | 0.534 | 0.476 | 0.500 | 0.541 | 0.545 |
| xercesinit | 0.514 | 0.516 | 0.518 | 0.417 | 0.440 | 0.556 | 0.539 | 0.487 |
| **AVG** | **0.632** | 0.631 | 0.623 | 0.599 | 0.540 | 0.604 | 0.604 | 0.563 |
| **AR** | **3.9** | 4.0 | 4.3 | 5.9 | 9.5 | 5.1 | 5.0 | 7.4 |

**Table 3.** The BAL values of the models built with eight learning algorithms

| | CCBE | NB | RF | DT | SVM | AB | EE | LA |
|---|---|---|---|---|---|---|---|---|
| ant13 | 0.630 | 0.624 | 0.652 | 0.692 | 0.564 | 0.625 | 0.666 | 0.566 |
| arc | 0.641 | 0.620 | 0.678 | 0.579 | 0.574 | 0.692 | 0.672 | 0.645 |
| ivy11 | 0.678 | 0.483 | 0.531 | 0.507 | 0.387 | 0.506 | 0.605 | 0.580 |
| jedit32 | 0.550 | 0.537 | 0.541 | 0.610 | 0.462 | 0.484 | 0.517 | 0.486 |
| log4j10 | 0.719 | 0.642 | 0.520 | 0.705 | 0.438 | 0.520 | 0.639 | 0.562 |
| lucene20 | 0.642 | 0.513 | 0.508 | 0.594 | 0.427 | 0.562 | 0.586 | 0.591 |
| poi15 | 0.623 | 0.683 | 0.613 | 0.535 | 0.494 | 0.616 | 0.572 | 0.558 |
| synapse10 | 0.701 | 0.707 | 0.735 | 0.662 | 0.511 | 0.685 | 0.654 | 0.538 |
| velocity14 | 0.467 | 0.386 | 0.374 | 0.484 | 0.315 | 0.386 | 0.493 | 0.474 |
| xercesinit | 0.504 | 0.433 | 0.453 | 0.389 | 0.302 | 0.500 | 0.531 | 0.455 |
| **AVG** | **0.615** | 0.563 | 0.561 | 0.576 | 0.447 | 0.558 | 0.593 | 0.545 |
| **AR** | **3.1** | 5.8 | 5.3 | 4.9 | 10.0 | 5.9 | 4.1 | 6.0 |

**Table 4.** The F values of the models built with eight learning algorithms

| | CCBE | NB | RF | DT | SVM | AB | EE | LA |
|---|---|---|---|---|---|---|---|---|
| ant13 | 0.391 | 0.408 | 0.373 | 0.425 | 0.294 | 0.350 | 0.395 | 0.302 |
| arc | 0.299 | 0.377 | 0.343 | 0.242 | 0.361 | 0.350 | 0.317 | 0.294 |
| ivy11 | 0.695 | 0.415 | 0.484 | 0.454 | 0.231 | 0.444 | 0.600 | 0.584 |
| jedit32 | 0.433 | 0.448 | 0.443 | 0.503 | 0.315 | 0.345 | 0.404 | 0.362 |
| log4j10 | 0.562 | 0.576 | 0.440 | 0.583 | 0.311 | 0.431 | 0.481 | 0.395 |
| lucene20 | 0.635 | 0.446 | 0.430 | 0.562 | 0.298 | 0.503 | 0.580 | 0.550 |
| poi15 | 0.632 | 0.710 | 0.602 | 0.511 | 0.608 | 0.674 | 0.619 | 0.643 |
| synapse10 | 0.322 | 0.392 | 0.393 | 0.286 | 0.313 | 0.300 | 0.275 | 0.198 |
| velocity14 | 0.477 | 0.240 | 0.205 | 0.449 | 0.064 | 0.240 | 0.464 | 0.418 |
| xercesinit | 0.451 | 0.309 | 0.345 | 0.279 | 0.584 | 0.417 | 0.486 | 0.365 |
| **AVG** | **0.490** | 0.432 | 0.406 | 0.429 | 0.338 | 0.405 | 0.462 | 0.411 |
| **AR** | **3.8** | 4.4 | 6.1 | 5.5 | 7.6 | 6.3 | 4.6 | 6.8 |




(a) AUC (b) Bal (c) F-Value

**Figure 5.** Comparison of different methods

# 6 Threats To Validity

In this study, we obtained several significant results to answer the three research questions proposed in Section 4.3. However, potential threats to the validity of our work still remain.

Threats to construct validity concern the accuracy of independent variable and dependent variable to the described concept. These threats are primarily related to the static code metrics we used. In this study, all the data sets selected in the experiment are obtained from the PROMISE repository and have been widely used in related research. Therefore, it can be considered that the construction validity threat of the experimental study meets the requirements.

Threats to internal validity concern any confounding factor that could influence the results, and they are mainly related to various experimental settings in this study. There are two main manifestations: one is the choice of model evaluation measure, we only select the most commonly used PD/PF/AUC/BAL/F-value metrics to achieve cost-cognition in the experiment. In fact, from the requirements of software engineering practice, these approaches are not enough to meet the needs; Second, the integrity of benchmark methods in experimental comparison. Therefore, in future studies, we will continue to explore cost-cognition methods for different practical needs and select more benchmark algorithms for the comparison validation.

Threats to external validity concern the generalization of the results obtained. The experimental datasets selected in this experiment are open-source software projects based on Java language. Furthermore, in order to study the relationship between class imbalance rates and cost values, we randomly selected 10 medium-sized projects for the experiment. The experimental results may not be extended to other types of software systems, but this is a common problem in empirical research. In future research, we will try to mitigate this external effectiveness threat by selecting more types and sizes of software project data.

# 7 Conclusion

Cross-project defects prediction plays an important role in improving software quality in case of projects without sufficient historical data. This paper proposes a novel method for cross-project defects prediction, namely the cost-cognitive bagging ensemble (CCBE) classifier, which handles the class-imbalance problem in cross-project defects data.

We compared the proposed method with the conventional methods on the 10 PROMISE datasets, which shows that our proposed method significantly outperforms others. Our method could cognize the cost value automatically during the model training, and without the need to predefine the cost parameters, it is shown to be more effective and practical.

Though results are promising, many problems still need to be solved in this context. Future work will be focused primarily on three aspects: Firstly, we may consider data filters other than TGF to further verify the performance of the CCBE algorithm in cross-project defect prediction. Secondly, we will further explore other cost factor learning methods to meet the needs of different software engineering practices. Thirdly, we may apply our approach to more datasets to validate the findings.

# Acknowledgment

This work is supported by the Xinjiang Tianshan Youth Project of China (2020Q019), the National Natural Science Foundation of China (61562087), and the Doctoral Scientific Research Foundation of Xinjiang Normal University (XJNUBS1905).

# References

[1] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, Z. Wang, Large-Scale Empirical Studies on Effort-aware Security Vulnerability Prediction Methods, *IEEE Transactions on Reliability*, Vol. 69, No. 1, pp. 70-87, March, 2020.

[2] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, M. F. Siok, Recent Catastrophic Accidents: Investigating How Software was Responsible, *Proceedings of the Fourth IEEE International Conference on Secure Software*

*Integration and Reliability Improvement*, Singapore, Singapore, 2010, pp. 14-22.

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

[4] M. Shepperd, D. Bowes, T. Hall, Researcher bias: The Use of Machine Learning in Software Defect Prediction, *IEEE Transactions on Software Engineering*, Vol. 40, No. 6, pp. 603-616, June, 2014.

[5] B. A. Kitchenham, E. Mendes, G. H. Travassos, Cross versus Within-Company Cost Estimation Studies: A Systematic Review, *IEEE Transactions on Software Engineering*, Vol. 33, No. 5, pp. 316-329, May, 2007.

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

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

[8] Y. Li, Z. Huang, Y. Wang, B. Fang, Evaluating Data Filter on Cross-project Defect Prediction: Comparison and Improvements, *IEEE Access*, Vol. 5, pp. 25646-25656, November, 2017.

[9] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, A Review on Ensembles for the Class Imbalance Problem: Bagging, Boosting, and Hybrid-based Approaches, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 4, pp. 463-484, July, 2012.

[10] D. Rodriguez, I. Herraiz, R. Harrison, On Software Engineering Repositories and Their open Problems, *2012 First International Workshop on Realizing AI Synergies in Software Engineering*, Zurich, Switzerland, 2012, pp. 52-56.

[11] L. C. Briand, W. L. Melo, 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, July, 2002.

[12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process, *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, Amsterdam, The Netherlands, 2009, pp. 91-100.

[13] K. Weiss, T. M. Khoshgoftaar, D. Wang, A Survey of Transfer Learning, *Journal of Big Data*, Vol. 3, No. 1, pp. 1-40, May, 2016.

[14] Y. Li, Z. Q. Huang, Y. Wang, B. W. Fang, Survey on Data Driven Software Defects Prediction, *Acta Electronica Sinica*, Vol. 45, No. 4, pp. 982-988, April, 2017.

[15] Q. Zou, L. Lu, S. Qiu, X. Gu, Z. Cai, Correlation Feature and Instance Weights Transfer Learning for Cross Project Software Defect Prediction, *Institution of Engineering and Technology Software*, Vol. 15, No. 1, pp. 55-74, February, 2021.

[16] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a Fault Prediction Model to Allow inter Language Reuse, *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, Leipzig, Germany, 2008, pp. 19-24.

[17] J. Nam, S. J. Pan, S. Kim, Transfer Defect Learning, *2013 35th International Conference on Software Engineering*, San Francisco, CA, USA, 2013, pp. 382-391.

[18] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An Empirical Study on Software Defect Prediction with a Simplified Metric Set, *Information and Software Technology*, Vol. 59, No. C, pp. 170-190, March, 2015.

[19] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An Investigation on the Feasibility of Cross-project Defect Prediction, *Automated Software Engineering*, Vol. 19, No. 2, pp. 167-199, June, 2012.

[20] B. Turhan, T. Menzies, A. B. Bener, 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, October 2009.

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

[22] S. Herbold, Training Data Selection for Cross-project Defect Prediction, *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, Baltimore, Maryland, USA, 2013, pp. 1-10.

[23] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, T. Zimmermann, Local versus Global Lessons for Defect Prediction and Effort Estimation, *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, pp. 822-834, June, 2013.

[24] Q. Song, Y. Guo, M. Shepperd, A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction, *IEEE Transactions on Software Engineering*, Vol. 45, No. 12, pp. 1253-1269, December, 2019.

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

[26] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, N. Seliya, A Survey on Addressing High-class Imbalance in Big Data, *Journal of Big Data*, Vol. 5, No. 1, pp. 1-30, November, 2018.

[27] S. Wang, L. L. Minku, Y. Xin, A Systematic Study of Online Class Imbalance Learning with Concept Drift, *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 10, pp. 4802-4821, October, 2018.

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

[29] K. Bennin, J. Keung, A. Monden, On the Relative Value of Data Resampling Approaches for Software Defect Prediction, *Empirical Software Engineering*, Vol. 24, No. 2, pp. 602-636, April, 2019.

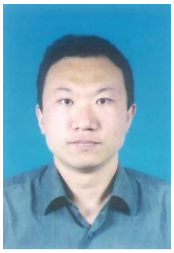
[30] C. Tantithamthavorn, S. Mcintosh, A. E. Hassan, K. Matsumoto, The Impact of Automated Parameter Optimization on Defect Prediction Models, *IEEE Transactions on Software Engineering*, Vol. 45, No. 7, pp. 683-711, July, 2019.
[31] B. Turhan, On the Dataset Shift Problem in Software Engineering Prediction Models, *Empirical Software Engineering*, Vol. 17, No. 1-2, pp. 62-74, February, 2012.
[32] X. Xia, D. Lo, S. J. Pan, N. Nagappan, X. Wang, HYDRA: Massively Compositional Model for Cross-project Defect Prediction, *IEEE Transactions on Software Engineering*, Vol. 42, No. 10, pp. 977-998, October, 2016.
[33] Z. Yuan, X. Chen, Z. Cui, Y. Mu, ALTRA: Cross-project Software Defect Prediction via Active Learning and Tradaboost, *IEEE Access*, Vol. 8, pp. 30037-30049, February, 2020.
[34] S. B. Wang, Y. Li, W. B. Mi, Y. Liu, Software Defect Prediction Incremental Model using Ensemble Learning, *International Journal of Performability Engineering*, Vol. 16, No. 11, pp. 1771-1780, November, 2020.
[35] M. Jureczko, L. Madeyski, Towards Identifying Software Project Clusters with Regard to Defect Prediction, *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, Timişoara, Romania, 2010, pp. 1-10.
[36] T. Menzies, A. Dekhtyar, J. Distefano, J. Greenwald, Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'", *IEEE Transactions on Software Engineering*, Vol. 33, No. 9, pp. 637-640, September, 2007.
[37] X. Y. Liu, J. Wu, Z. H. Zhou, Exploratory Under sampling for Class-Imbalance Learning, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 39, No. 2, pp. 539-550, April, 2009.
[38] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, A. Bener, Defect Prediction from Static Code Features: Current Results, Limitations, new Approaches, *Automated Software Engineering*, Vol. 17, No. 4, pp. 375-407, December, 2010.
[39] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, Further thoughts on Precision, *15th Annual Conference on Evaluation and Assessment in Software Engineering*, Durham, United Kingdom, 2011, pp. 129-133.
[40] A. Monden, T. Hayashi, S. Shinoda, K. Shirai, J. Yoshida, M. Barker, K. Matsumoto, Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing, *IEEE Transactions on Software Engineering*, Vol. 39, No. 10, pp. 1345-1357, October, 2013.
[41] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, M. Zelkowitz, What We Have Learned about Fighting Defects, *Proceedings Eighth IEEE Symposium on Software Metrics*, Ottawa, ON, Canada, 2002, pp. 1-10.

## Biographies

**Yong Li** received the Ph.D. degree in Computer Science from Nanjing University of Aeronautics and Astronautics in 2018. He is currently an Associate Professor of Xinjiang Normal University and a Postdoctoral Fellow of Xinjiang Electronic Research Institute. His research interests include Machine Learning and Intelligent Software Engineering.

**Ming Wen** graduated from Xi'an University of Technology in 1988 with a major in automatic control. Now he is the director and researcher of software development and testing center of Xinjiang Electronic Research Institute. His research interests include Software Engineering and Artificial Intelligence.

**Zhandong Liu** received the Ph.D. degree in Computer Science from University of Science and Technology of China in 2018. He is currently an Associate Professor with the College of Computer Science and Technology of Xinjiang Normal University. His research interests include Pattern Recognition and Computer Algorithm.



**Haijun Zhang** received the Ph.D. degree in Computer Science from University of Science and Technology of China in 2011. He is currently a Professor with the College of Computer Science and Technology of Xinjiang Normal University. His research interests include Software Engineering and Data Mining.