

# Value-cognitive boosting with a support vector machine for cross-project defect prediction

Duksan Ryu · Okjoo Choi · Jongmoon Baik

Published online: 17 December 2014  
© Springer Science+Business Media New York 2014

**Abstract** It is well-known that software defect prediction is one of the most important tasks for software quality improvement. The use of defect predictors allows test engineers to focus on defective modules. Thereby testing resources can be allocated effectively and the quality assurance costs can be reduced. For within-project defect prediction (WPDP), there should be sufficient data within a company to train any prediction model. Without such local data, cross-project defect prediction (CPDP) is feasible since it uses data collected from similar projects in other companies. Software defect datasets have the class imbalance problem increasing the difficulty for the learner to predict defects. In addition, the impact of imbalanced data on the real performance of models can be hidden by the performance measures chosen. We investigate if the class imbalance learning can be beneficial for CPDP. In our approach, the asymmetric misclassification cost and the similarity weights obtained from distributional characteristics are closely associated to guide the appropriate resampling mechanism. We performed the effect size A-statistics test to evaluate the magnitude of the improvement. For the statistical significant test, we used Wilcoxon rank-sum test. The experimental results show that our approach can provide higher prediction performance than both the existing CPDP technique and the existing class imbalance technique.

**Keywords** Boosting · Class imbalance · Cross-project defect prediction · Transfer learning

## 1 Introduction

It is very difficult to develop a defect-free software system. Therefore, most software development organizations have been trying to detect and correct defects as many as possible to

---

Communicated by: Tim Menzies

D. Ryu (✉) · O. Choi · J. Baik  
Department of Computer Science, Korea Advanced Institute of Science and Technology, 291 Daehak-ro  
(373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, Republic of Korea  
e-mail: dsryu@kaist.ac.kr

O. Choi  
e-mail: okjoo.choi@kaist.ac.kr

J. Baik  
e-mail: jbaik@kaist.ac.kr

improve the process and project performance before they release their software products. Various types of defect prediction mechanisms (or models) have been developed to predict defective modules or components in software intensive systems. Based on the defect prediction results, software developers can concentrate on defect-prone modules with schedule and cost constraints, which can lead to high project success. In general, software project measures are collected locally within a company and utilized for process and quality improvements. Especially for quality improvement, it is difficult to build prediction models using local project data in case of a pilot project or the lack of historical project data within a company. Cross-project defect prediction (CPDP) is known as a feasible alternative solution in such cases. There have been different approaches for CPDP. First, a transfer learning technique extracts common knowledge from one domain and transfers it to another, and a prediction model can be constructed using the transferred knowledge (Ma et al. 2012; Nam et al. 2013). Second, instance selection can be used to find the most appropriate training data for classifying the project. This method exploits the distributional characteristics of a dataset, e.g., median, mean, and variance. Then, the relationship among the distributional characteristics of datasets and the CPDP results can be examined. Training data are selected from other projects using the verified distributional characteristics of a dataset (He et al. 2011). Third, a data filtering technique can be used to guide the selection of the filtered training dataset because the prediction models developed from the entire dataset have low recall and high false alarm rates (Turhan et al. 2009; Peters et al. 2013).

Mostly, the number of defective instances is significantly fewer than that of non-defective instances. This imbalanced distribution can cause poor prediction performance of specific classification methods (Arisholm et al. 2010; Hall et al. 2012; GRBAC and GORAN 2013). As Hall et al. (Hall et al. 2012) asserted, the performance measures chosen can hide the impact of imbalanced data on the real performance of classifiers. It means that the probability of defect detection can be low while the overall performance is high. Under WPDP settings, there exist different methods, e.g. data sampling (Gao and Khoshgoftaar 2011), threshold moving (Zheng 2010), and ensemble methods (Wang and Yao 2013). However, none of the existing CPDP methods have taken into account on the class imbalance problem of software defect datasets. Since there is no previous CPDP approaches considering the class imbalance problem, it is uncertain whether the class imbalance learning can improve the prediction performance under CPDP settings.

In this study, we investigate the applicability of the class imbalance learning under CPDP setting with our proposed method called the value-cognitive boosting with support vector machine (VCB-SVM).

To evaluate the VCB-SVM method, the following research questions are explored.

- RQ1: Does the VCB-SVM provide higher prediction performance than the existing CPDP techniques not prepared for the class imbalance?
- RQ2: Does the VCB-SVM provide higher prediction performance than the class imbalance techniques not prepared for the cross-project learning?

The main goal of this research is to develop a more effective prediction modeling method dealing with the class imbalance issue for cross-project environments as the answers to the two research questions. The VCB-SVM is a novel boosting model designed to improve the prediction performance by sampling and modifying the datasets (Class Imbalance Learning) based on the common knowledge in different distributions (Transfer Learning). For the first question, we compare our approach with Transfer Naïve Bayes (Ma et al. 2012) to identify whether the emphasis of the minority class is useful under the CPDP setting. For the second

question, we compare our method with Boosting-SVM (Wang and Japkowicz 2009) to figure out whether the class imbalance learning methods not using any procedure to transfer knowledge can be useful for CPDP. The experimental results demonstrate that the proposed approach is promising. According to the experimental results using the effect size test and the Wilcoxon rank-sum test, the performance of our approach was better than those of other methods. The answers to these questions will help to design an effective prediction model for CPDP.

The remainder of this paper is organized as follows. In the next section, we briefly introduce software defect prediction, transfer learning, and class imbalance learning to provide the necessary background to this study. In [section 3](#), we describe related work in the area of CPDP and the class imbalance learning. In [section 4](#), we present our new approach. The experimental setup is described in [section 5](#) and the results of the experiments are given in [section 6](#). The threats to the validity of our approach are explained in [section 7](#). In the last section, we conclude this paper and discuss future research.

## 2 Background

### 2.1 Software Defect Prediction

Software testing and inspection are crucial software quality assurance activities but they are labor-intensive and time-consuming. In general, there are limited human and time resources for those activities. Software defects are distributed disproportionally. According to the Pareto principle, most software defects are found in a small number of modules. Thus, to enhance the efficacy and efficiency of software quality control, resources should be allocated cautiously to defect-prone software modules (He et al. 2011). Software defect prediction is a method for predicting defect-prone software modules. Defect prediction can help to optimize resource allocation for testing and inspection. There are two types of defect prediction, i.e. change classification and buggy file prediction (Kim et al. 2011). Change classification learns the buggy and clean change patterns from the revision history, before predicting a defect that introduces a change. Buggy file prediction uses code features, e.g., complexity metrics and process metrics, to predict buggy files. For example, complexity metrics include the lines of code and the number of attributes. Process metrics include the number of fixes and the number of lines added. In the present study, we focus on the buggy file prediction method. Various buggy file prediction models are proposed, but most are only applicable to WPDP, which requires sufficient historical data to train models. Occasionally, there are cases where historical data are not available, such as when an organization starts a pilot project or does not have enough set of training dataset. Some organizations may not be able to maintain a software repository because of the cost involved. To address this problem, we can employ a CPDP that uses data from other projects to build defect classifiers (He et al. 2011; Ma et al. 2012).

### 2.2 Transfer Learning Technique

In real-world applications, there is sometimes a limited set of data (or no data) for the domain of interest. As Pan and Yang (Pan and Yang 2010) explained, the transfer learning technique is a relatively new learning technique where knowledge obtained from another domain, which has an adequate training dataset, can be transferred to the domain of interest. Using the extracted knowledge, data from different domains and distributions can be used for training during the learning process.

CPDP can be considered a transfer learning problem because the training data and testing data used in CPDP come from different projects with different feature distributions (Ma et al. 2012). According to Pan and Yang (Pan and Yang 2010), there are different transfer learning settings. In our study, we considered that the most reasonable setting is that the source project has an adequate labeled dataset with defect information whereas the target project only has some unlabeled data without defect information.

According to Zimmermann et al. (Zimmermann et al. 2009), the identification of the distributional characteristics of data sets are crucial to the success of CPDP. Previous researches of CPDP (He et al. 2011; Ma et al. 2012; Nam et al. 2013) also employed dataset characteristics to measure the similarity between a source project and a target project. The elements of the characteristics include the mean, median, minimum, maximum, standard deviation, etc. In this study, we utilize the range between the maximum value and the minimum value of the attribute which can represent an aspect of the distributional characteristic of a target project.

### 2.3 Class Imbalance Learning Technique

Class imbalance learning indicates the learning on data sets with imbalanced class distributions, meaning that the number of defective instances is much fewer than that of non-defective instances (Tan et al. 2005). Though the frequency of occurrence is low, classifying an instance correctly from the minority class has greater value. If defective instances are not correctly identified, software quality could be significantly deteriorated. In the context of the class imbalance learning, it is important to obtain a learner providing high accuracy for the minority class without severely lowering the accuracy of the majority class (Garcia 2009). Since many of the existing learning methods do not effectively detect instances of the minority class, various methods have been developed at data and algorithm levels. As a data-level method, sampling is a widely used method to deal with the class imbalance problem (Tan et al. 2005). Under-sampling the majority class can be used to provide a balanced distribution. As the opposite of under-sampling, over-sampling is utilized to duplicate the minority instances.

As an algorithm-level method, cost-sensitive learning assigns distinct costs to the training instances. Zheng (Zheng 2010) suggested a cost-sensitive method using a neural network for software defect prediction. Boosting techniques have been used with success for handling the class imbalance problem (Wang and Yao 2013).

## 3 Related Work

### 3.1 Cross-Project Defect Prediction

Many defect prediction approaches using machine learning techniques have been proposed (Kim et al. 2008; Elish and Elish 2008; Shatnawi and Li 2008; Mende and Koschke 2009; Singh et al. 2009; Arisholm et al. 2010; Menzies et al. 2010; Lee et al. 2011; D'Ambros et al. 2011; Dejaeger 2013). In the within-project settings, several studies have used AUC (the area under the receiver operating characteristic curve) as a performance measure (Shatnawi and Li 2008; Mende and Koschke 2009; Singh et al. 2009; Arisholm et al. 2010; Menzies et al. 2010; D'Ambros et al. 2011; Dejaeger 2013). Recently, researchers have become interested in the issue of predicting defects in cross-project environments.

Using 12 applications, Zimmermann et al. (Zimmermann et al. 2009) performed 622 CPDPs, but only 21 predictions worked successfully. This indicates that CPDP will not be

successful in most cases without the careful selection of training data. The characteristics of the data and the process have important roles in successful CPDP. Decision trees for the precision, recall, and accuracy are used to discover the interactions among the characteristics that affect cross-project predictions. CPDP is considered challenging and more researchers should be concerned about this problem.

He et al. (He et al. 2011) examined defect prediction with a focus on selecting training data in a cross-project setting. They performed three experiments using 34 datasets from 10 open source projects. They showed that the prediction results were connected with the distributional attributes of datasets, which is useful for training data selection. Indicators of the distributional characteristics of a dataset include the median, mean, range, variance, standard deviation, skewness, and kurtosis. They used the precision, recall, and F-measure to demonstrate the performance of the proposed instance selection method.

Turhan et al. (Turhan et al. 2009) applied the principle of analogy-based learning (i.e., nearest neighbor (NN) filtering) to cross-project data. They analyzed 10 projects, i.e., seven NASA projects from seven different companies and three projects from a Turkish software company. They used the static code attributes to construct defect predictors and a defect classifier learned from within-project data was superior to those learned from cross-project data.

Nam et al. (Nam et al. 2013) applied transfer learning approaches to CPDP. This approach applied transfer component analysis (TCA) to defect prediction. They also proposed a new approach called TCA+, which selected suitable normalization options for TCA. The precision, recall, and F-measure were used for the performance evaluation.

Premraj et al. (Premraj and Herzig 2011) compared the network metrics and code metrics for three open source Java projects in a cross-project setting. They found that network metrics were not superior to code metrics in the context of cross-project prediction. A series of cutoffs for each metric was used to calculate a ROC curve. Next, the importance of the metric is determined using the AUC.

Ma et al. (Ma et al. 2012) proposed an algorithm called transfer naive Bayes (TNB), which used information from all of the suitable attributes in the training data. Based on the estimated distribution of the test data, this method transferred cross-company data information to the weights of the training data. The defect prediction model was constructed using these weighted data.

Previous works studied various software defect prediction approaches under cross-project settings. However, the class imbalance issue of software defect datasets was not taken into consideration. Hall et al. (Hall et al. 2012) carried out a systematic literature review on software defect prediction. In their work, they indicated that data imbalance with regard to specific classification methods may be connected to poor performance. In addition, they suggested more studies be aware of the need to deal with data imbalance. Particularly, they assert that the performance measures chosen can hide the impact of imbalanced data on the real performance of classifiers.

### 3.2 Class Imbalance Learning

Grbac and Goran (GRBAC and GORAN 2013) studied the performance stability of machine learning techniques while differentiating levels of imbalance for software defect datasets. The results showed a high level of imbalance could make the prediction performance unstable.

Gao and Khoshgoftaar (Gao and Khoshgoftaar 2011) proposed an approach to deal with high dimensionality and class imbalance of software defect data. To this end, feature selection

and data sampling were utilized together. This method deals with class imbalance by modifying the training data.

Ren et al. (Ren et al. 2014) presented kernel based prediction models to deal with the class imbalance issue. The NASA and SOFTLAB datasets are used for experiments.

Wang and Yao (Wang and Yao 2013) propose a boosting algorithm, which adjusted its parameter automatically during training. Imbalance learning techniques, e.g., resampling methods, threshold moving, and ensemble methods were investigated. The G-mean and AUC were used to evaluate the performance.

Zheng (Zheng 2010) suggested a boosting approach where a neural network was used for software defect prediction. A threshold-moving method was proposed to construct a cost-sensitive boosting algorithm. To predict more buggy modules correctly, this method moved the prediction threshold toward the clean modules. The normalized expected cost of misclassification (NECM) was utilized as a performance measure. The misclassification costs and the prior probabilities of the two classes were combined in the NECM.

For imbalanced datasets, a classifier can be built under the influence of skewed vector spaces. To solve the skewed vector spaces problem, Wang and Japkowicz (Wang and Japkowicz 2009) proposed a support vector machine with the asymmetric misclassification cost. Their proposed approach named Boosting-SVM combined the modification of the data distribution with the modification of the classifier. They attempted to solve the class imbalance problem by exploiting the property of soft margins. Any bias introduced by applying soft margins was mitigated with a boosting method. The geometric mean (G-mean) was used as a performance measure for medical datasets.

In our study, we adopted the Boosting-SVM and resampling techniques. Boosting-SVM is employed since the asymmetric misclassification cost of the training instances can be seamlessly related to the similarity weight of the training instances based on the test instances. Resampling methods guided by the asymmetric misclassification cost and the similarity weight are used to balance the distributions. Details of the proposed algorithm will be described in the next section.

#### 4 Value-Cognitive Boosting with Support Vector Machine (VCB-SVM) Approach

We propose a value-cognitive boosting with SVM approach, which is a prediction modeling method that exploits the similarity weight drawn from distributional characteristics and the asymmetric misclassification cost. It is called VCB-SVM since similarity weight values of the training instances are used in a mechanism to balance the imbalanced distributions.

The overall defect prediction process based on the value-cognitive boosting approach is shown in Fig. 1. First, to prepare the dataset for training, it is necessary to assign labels to the training and test datasets before computing the similarity weight. A Java class, i.e., an instance of data, is marked as buggy if there is one or more bugs in the Java class. Otherwise, it is marked as clean. Second, the similarity weights of the training data are computed based on the test data. In this step, without labels, only the input features of the training set and the test set are used for the computation. Third, we build a prediction model using the boosting method with SVM. In this study, we adopt Boosting-SVM (Wang and Japkowicz 2009) and resampling techniques to address the class imbalance problem. Fourth, classifications are predicted using a boosted ensemble classifier. Finally, a performance evaluation of prediction results is performed.

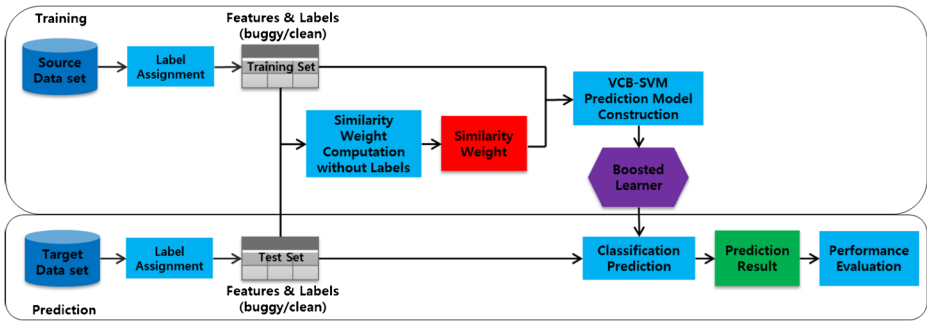


Fig. 1 Overall defect prediction process with the VCB-SVM approach

### 4.1 Label Assignment

Software metrics data cannot be used directly as the inputs for training. During this step, buggy or clean labeling is conducted using previous defect information. If bugs exist, they are marked as buggy.

### 4.2 Similarity Weight Computation without Labels

To measure the similarity between a source project and a target project, the dataset characteristic vector (DCV) was proposed in a previous study (Nam et al. 2013). The elements of the DCV include the mean, median, minimum, maximum, and standard deviation of the attribute. The attribute indicates the input feature not including labels. Ma et al. (Ma et al. 2012) computed the similarity by counting the number of similar attributes. The range between the maximum value and the minimum value of the attribute is used to check the similarity. Then, a data gravitation method is used to compute the weights of training instances.

In our approach, we compute the number of similar attributes in a similar manner to a previous method (Ma et al. 2012), i.e., we compute the similarity weight by dividing it by the total number of attribute.

Given a sequence  $x_i = \{a_{i1}, a_{i2}, \dots, a_{ik}\}$ ,  $a_{ij}$  is the  $j$ th attribute of  $x_i$ . The maximum value and minimum value of  $j$ th attribute in the test set are calculated:

$$\max_j = \max\{a_{1j}, a_{2j}, \dots, a_{mj}\}, \min_j = \min\{a_{1j}, a_{2j}, \dots, a_{mj}\}$$

where  $j=1, 2, \dots, k$ ,  $k$  is the number of attributes and  $m$  is the number of test data. The following two vectors have the maximum and the minimum value of the attribute on the test set.  $\text{Max} = \{\max_1, \max_2, \dots, \max_k\}$ ,  $\text{Min} = \{\min_1, \min_2, \dots, \min_k\}$ . Next, we compute the similarity weight of each training instance using the following formula:

$$S_i = \sum_{j=1}^k h(a_{ij})/k$$

where  $h(a_{ij}) = \{ 1 \text{ if } \min_j \leq a_{ij} \leq \max_j, 0 \text{ otherwise} \}$ ,  $a_{ij}$  is the  $j$ th attribute of instance  $x_i$ .

### 4.3 VCB-SVM Prediction Model Construction

#### 4.3.1 VCB-SVM

We propose a VCB-SVM considering data imbalance for CPDP. Algorithm 1 shows the details of our boosting algorithm. In this algorithm, transfer learning and class imbalance learning techniques are closely related together. The instances of a validation set are selected based on their similarity weights. This means that the distribution of the validation set will become similar to the distribution of the test set.

**Algorithm 1.** VCB-SVM algorithm

**Input parameters**

**X:** input sequence

**SW:** similarity weight

**M:** the maximum number of iterations

**$\lambda$ :** parameter of the penalty scale for each iteration ( $0 < \lambda \leq 1$ )

**Local variables**

**$X_{\text{train}}$ :** training set of input sequence

**$X_{\text{validation}}$ :** validation set of input sequence

**N:** the number of training examples in  $X_{\text{train}}$

**$t_n$ :** binary target variable of  $X_{\text{train}}$  where  $t_n \in \{-1, 1\}$  and  $n = 1, \dots, N$ .

**$\rho$ :** a value of AUC or H-measure

**T:** selected running iterations

**Function calls:**

**w:** the weighting coefficient

**Resampling(X, SW, w):** resample X based on SW and w

**h:** base learner

**SVMTrain(X):** train a base learner h using SVM

**SVMClassify(X, h):** classify X by the learner h

**I:** an indicator function where  $I(\text{false}) = 0$ ,  $I(\text{true}) = 1$

**V:** ensemble learner

**F(X, V):** obtain the AUC or H-measure from V using X

1. Select instances for validation

**$X_{\text{validation}}$ :** instances of X with high SW

**$X_{\text{train}} = \{X - X_{\text{validation}}\}$**

2. Initialize

**$w_n = 1$  for  $n = 1, \dots, N$ .**

**$\rho_{\text{best}} = 0$**

**T = 1**

3. For  $m = 1, 2, \dots, M$ :

**$X_{\text{train}} \leftarrow \text{Resampling}(X_{\text{train}}, SW_n, t_n, w_n)$  where  $m > 1$**

**$X_{\text{train}}(x) \leftarrow X_{\text{train}}(x)$  using weights  $w_n$**

**$h_m \leftarrow \text{SVMTrain}(X_{\text{train}})$**

$$\epsilon_m = \frac{\sum_{n=1}^N w_n I(\text{SVMClassify}(X_{\text{train}}, h_m) \neq t_n)}{\sum_{n=1}^N w_n}$$

$$\alpha_m = \lambda \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

$$w_n \leftarrow w_n \exp\{\alpha_m (\text{SVMClassify}(X_{\text{train}}, h_m) \neq t_n)\}$$



$$\rho_m = F(X_{\text{validation}}, V_m), \text{ where } V_m = \text{sign} \left[ \sum_{j=1}^m \alpha_j h_j \right]$$

If  $\rho_m \geq \rho_{\text{best}}$   
 Then  $T = m$  and  $\rho_{\text{best}} = \rho_m$

Else return  $V_T$

$X$  is given as the sequence of a dataset to produce a final ensemble classifier.  $X$  is divided into two datasets: a training set and a validation set for the algorithm.  $N$  is the number of training examples in a training set. The initial weights of the training instances is set as 1.  $M$  is the maximum number of iterations.  $\rho$  is the value of the AUC or H-measure. The AUC and H-measure were selected because they are effective performance measures for the class imbalance problem.  $T$  is the running iteration. Based on the termination criteria in the main loop,  $T$  becomes optimal for a validation set.  $V$  is the final ensemble learner built by the algorithm, which is used to predict the classification of the test dataset.

In the first step, the validation set and training set are divided based on their similarity weight. We used a stratified sampling method for all training and validation sets. In this way, the ratio of negative to positive examples would be same for each of them. Figure 2 shows the validation set selection process. (1) Input set is randomly split in 50:50. (2) The half of the input set is ordered based on the similarity weights in descending order and the validation sets are selected based on the high similarity weights. Then, 2/5 of them are selected for the validation sets. As a result, 1/5 of the total input set are used as the validation set. This ratio of the validation set is commonly used (as a pareto principle). The remaining instances are selected as the training set. The validation set is very important for the algorithm because it determines the termination of the main loop. Thus, to increase the effectiveness of the prediction of the test dataset, the validation dataset should be similar to the test dataset in terms of its feature distributions.

In the second step, the initial weights for the SVM are set. This weight for each instance  $x_i$  affects the soft margin parameter  $C_i$  assigned to each instance  $x_i$ . If  $C_i$  increases, the classification of the SVM for  $x_i$  becomes strict.

In the third step, if the value of the AUC or H-measure cannot be increased for the validation set  $X_{\text{validation}}$ , the boosting algorithm is applied to the training dataset  $X_{\text{train}}$ . Resampling technique commonly used to balance the class distribution is employed based on the similarity weight and the weighting coefficient. In the case of  $m=1$ , all the weighting coefficient equals to one. Such a case is ruled out. Algorithm 2 shows the resampling algorithm. In this algorithm, we firstly divide the training instances into the instances with the similar distribution ( $X_S$ ) and the instances with the different distribution ( $X_D$ ) based on the similarity weight. The similarity weight of each training instance indicates how much each training instance is similar to the test dataset via the distributional characteristics. If SW equals to one, all the attributes are similar to the test dataset. Then, each group is subdivided into the minority and the majority classes. Finally, we resample instances of each group based on the

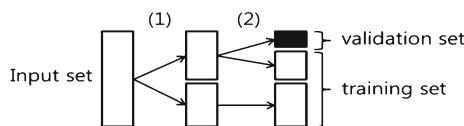


Fig. 2 Validation set selection

similarity weight and the weighting coefficient. Basically, the instances of the minority class are over-sampled and the instances of the majority class are under-sampled. Higher weighting coefficient of an instance indicates that it is misclassified continually. Lower weighting coefficient of an instance means that it is correctly classified. We focus on the misclassified instances of the similar distribution while disregarding the misclassified instances of the different distribution. To this end, we applied the following sampling policy.

- The minority class with the high weighting coefficient of the similar distribution is over-sampled. Thus, the misclassified instances will be focused.
- The majority class with the low weighting coefficient of the similar distribution is under-sampled. Thus, the correctly classified instances will be disregarded.
- The minority class with the low weighting coefficient of the different distribution is over-sampled. Thus, the correctly classified instances will be focused.
- The majority class with the high weighting coefficient of the different distribution is under-sampled. Thus, the misclassified instances will be disregarded.

The number of samples for each over-/under-sampling is typically designed as the number of the minority class (Garcia 2009). Since we divided the training dataset into the similar and the different datasets, 1/2 of the total number of the minority class is resampled for each over-/under-sampling case.  $h(x)$ , the base classifier, uses the weighting coefficients  $w_n$  for training. In successive iterations, the weighting coefficients  $w_n$  are increased for misclassified instances in the dataset and decreased for correctly classified instances in the dataset. Thus, subsequent classifiers focus on the instances misclassified by previous classifiers. If instances are misclassified continually by subsequent classifiers, they will receive increasingly higher weight. The quantities  $\varepsilon_m$  indicate the weighted measures of the error rates for each base classifier. The weighting coefficient  $\alpha_j$  gives higher weight to more accurate classifiers when computing  $V_m = \text{sign}[\sum_{j=1}^m \alpha_j h_j]$ . The final ensemble classifier is produced by a weighted majority vote based on the AUC or H-measure value.

**Algorithm 2.** Resampling algorithm

Input parameters

**X:** input sequence

**SW:** similarity weight

**t:** binary target variable of X

**w:** the weighting coefficient

Output: resampled instances of X

1. Divide into two groups based on the similarity weight
  - $X_S \leftarrow X$  where  $SW = 1$
  - $X_D \leftarrow X$  where  $SW \neq 1$
2. Divide into the minority class and the majority class
  - $X_{Smin} \leftarrow X_S$  where  $t = 1$
  - $X_{Smaj} \leftarrow X_S$  where  $t = -1$
  - $X_{Dmin} \leftarrow X_D$  where  $t = 1$
  - $X_{Dmaj} \leftarrow X_D$  where  $t = -1$
3. Resample instances of each group based on the similarity weight and the weighting coefficient
  - Over-sample  $X_i$  with high  $w$  where  $X_i \in X_{Smin}$
  - Under-sample  $X_i$  with low  $w$  where  $X_i \in X_{Smaj}$
  - Over-sample  $X_i$  with low  $w$  where  $X_i \in X_{Dmin}$
  - Under-sample  $X_i$  with high  $w$  where  $X_i \in X_{Dmaj}$

#### 4.4 Classification Prediction

Using the final ensemble classifier, classification prediction is the phase where the classifier predicts whether the unseen data in a target domain is defective. In within-project defect prediction (WPDP), the training and test sets contain instances from the same project. It means that they exist in the same feature space and the same feature distribution. In contrast to WPDP, the training and the test sets come from different projects in CPDP. The instances in the training and test sets should have the same features. In other words, the data points exist in the same feature space for cross-project settings. However, there are different feature distributions in the source and target domains.

#### 4.5 Performance Evaluation

To measure the learner built on software defect data sets having the imbalanced nature, the performance on the buggy class and the overall performance are typically both evaluated. To measure the performance on the buggy class, the probability of detection (PD) and the probability of a false alarm (PF) are commonly utilized. The evaluation of the overall performance aims at measuring how well the learner can balance the performance between the buggy and the clean classes. AUC (Bradley 1997) and H-measure (Hand 2009) can be employed for this purpose.

Table 1 shows the confusion matrix for the performance evaluation. True positive (TP) is the number of buggy modules predicted as buggy. False positive (FP) is the number of clean modules predicted as buggy. False negative (FN) is the number of buggy modules predicted as clean. True negative (TN) is the number of clean modules predicted as clean. PD indicates the proportion of appropriate instances retrieved, which is also known as the recall, and it was computed by:  $TP/(TP + FN)$ . PF is also called the false positive rate, and it was calculated by:  $FP/(FP + TN)$ . In contrast to PD, the performance of PF is better when its value is lower.

AUC indicates the probability that a learner will rank a randomly chosen example of the positive class higher than a randomly chosen example of the negative class. As an alternative to the AUC, H-measure can be used for evaluating classification performance (Hand 2009). A better learner should produce a higher AUC and H-measure. In our study, PD, PF, AUC, and H-measure are used to evaluate the performance.

### 5 Experimental Setup

With the two research questions mentioned earlier, we carried out experiments. To check if the VCB-SVM provides higher prediction performance than the CPDP techniques not prepared for the class imbalance (RQ1), performance estimators effective for the class imbalance problems should be selected. AUC, H-measure, G-mean are usually used for that purpose

**Table 1** Confusion matrix

		Predicted class	
		Buggy	Clean
Actual class	Buggy	TP (true positive)	FN (false negative)
	Clean	FP (false positive)	TN (true negative)

since they are related to the values of both PD and PF. The H-measure and G-mean have not been used in any previous study of CPDP. In terms of AUC, we compared our approach with Transfer Naïve Bayes (TNB) (Ma et al. 2012). The NASA and SOFTLAB datasets used by Ma et al. are employed to compare the performance of our approach in cross-project environments. To check if the VCB-SVM provide higher prediction performance than the class imbalance techniques not prepared for the cross-project learning (RQ2), we compared our approach with Boosting-SVM (Wang and Japkowicz 2009).

There is a trade-off between the overall performance (e.g., AUC and H-measure) and the defect detection rate (PD) (Wang and Yao 2013). Consequently, it is important to obtain a learner providing high accuracy for the minority class without severely lowering the accuracy of the majority class. Menzies et al. (Menzies et al. 2007) also indicated that PD measure is practically useful.

To answer the RQ1 and RQ2, the following hypotheses are formalized.

- If two classifiers produce the overall performance (e.g., AUC and H-measure) equivalently, a classifier producing higher PD values is better than the other.
- If two classifiers produce the PD performance equivalently, a classifier producing higher overall performance values is better than the other.

## 5.1 Data Collection

The NASA datasets were extracted from seven NASA subsystems. The SOFTLAB datasets were three systems related to an embedded controller from a Turkish software company (SOFTLAB). Two datasets are obtained from PROMISE repository (Shepperd 2011; Menzies et al. 2012). The characteristics of each dataset are described in Tables 2 and 3, respectively.

The official PROMISE repository (Menzies et al. 2012) has been up-to-dated continuously. Several of the latest datasets have different numbers of instances when comparing with datasets used in a previous study (Ma et al. 2012). Therefore, we used previous versions of the dataset maintained by the PROMISE repository, which had the same number of instances of datasets used in the previous study. However, the PROMISE repository site no longer maintains the kc1 dataset. We found a different version of the kc1 dataset with different numbers of instances (Shepperd 2011). We used this dataset because we could not find the kc1 dataset described in the previous study (Ma et al. 2012). In Table 2, the values in parentheses are from the

**Table 2** Projects in the NASA dataset (PROMISE)

Project	Instances	% Buggy	Description
pc1	1109	6.94	Flight software
kc1	1183 (1212)	24.85 (26.00)	Storage management
kc2	522	20.49	Storage management
kc3	458	9.38	Storage management
cm1	498	9.83	Spacecraft instrument
mw1	403	7.69	A zero gravity experiment
mc2	161	32.30	Video guidance system

**Table 3** Projects in the SOFTLAB dataset

Project	Instances	% Buggy	Description
ar3	63	12.7	Embedded controller
ar4	107	18.69	Embedded controller
ar5	36	22.22	Embedded controller

kc1 dataset used in a previous study. The NASA and SOFTLAB datasets have different features and thus only the common features were used in our experiments. All features in each of the NASA and SOFTLAB datasets are listed in the [Appendix](#). The common features of the NASA and SOFTLAB datasets are shown in [Table 4](#).

## 5.2 NASA and SOFTLAB Datasets

Two types of experiments were performed in our study, which were the same to the previous study of Ma et al. First, all of the NASA datasets were used as training dataset. Each SOFTLAB dataset was used as a test dataset. Second, each NASA dataset was selected as a test dataset and the remaining NASA datasets were used as a training dataset.

### 5.2.1 NASA to SOFTLAB

All of the NASA datasets were used as training dataset and each SOFTLAB dataset was used as a test dataset. We performed the test with 30 iterations commonly used to analyze and compare randomized algorithms (Arcuri and Briand 2011). For each iteration, the training set was randomized. The similarity weights of the NASA data instances were computed based on each SOFTLAB dataset. Z-score normalization was applied to the training and test datasets. A boosted classifier was constructed using Algorithm 1. Classifications were predicted using the test dataset and the AUC values were computed for the evaluation.

### 5.2.2 NASA to NASA

In this setting, only the NASA datasets were used for training and test datasets. Each NASA dataset was chosen as a test dataset and the other NASA datasets were utilized as a training dataset. The test was iteratively performed 30 times. For each iteration, the training set was randomized before building the classification model.

**Table 4** Shared features of the NASA and SOFTLAB datasets

Type	Features
McCabe	Cyclomatic complexity, design complexity
LOC	Code and comment loc, comment loc, executable loc, total loc
Halstead	Halstead difficulty, Halstead effort, Halstead error, Halstead length, Halstead time, Halstead volume, unique operands, unique operators, total operands, total operators
Other	Branch count

### 5.3 Learning Algorithms

SVM was employed as a base learner for the boosting approach and we used LIBSVM (Chang and Lin 2013). We only used the simple linear kernel option to execute LIBSVM. In this way, we can minimize the effect of kernel function and focus on the effect of the proposed algorithm. Based on the published guidelines for support vector classification (Gray et al. 2009; Hsu et al. 2010), z-score normalization was applied to all of the training and test datasets. We compared our proposed method with other learning algorithms, i.e., Naïve Bayes (NB), logistic regression (LR), PART, J48, random forest (RF), IBk, and multi-layer perceptrons (MLP) from the WEKA machine learning toolkit (Hall et al. 2009) as well as Boosting-SVM (Wang and Japkowicz 2009) and TNB (Ma et al. 2012). As a classification model, NB applies Bayes' theorem and assumes strong independence. LR is a classification model, i.e., it makes predictions when the dependent variable has two classes. PART generates a decision list. A rule is generated after constructing a partial C4.5 decision tree. J48 builds pruned or unpruned C4.5 decision trees. RF builds a forest of random trees. As an ensemble learner, decision trees are used iteratively to form a strong learner. IBk is a k-nearest neighbor classifier, which makes predictions based on the closest training instances. MLP is an artificial neural network model, which maps input datasets into relevant output datasets. Similar to SVM, we applied z-score normalization to all of training and test datasets before running those algorithms.

### 5.4 Parameter Configurations

To calculate the AUC, H-measure, PD and PF as the final performance measures, we used AUC as the validation criterion of the VCB-SVM and Boosting-SVM. AUC is used since it reflects the overall performance effectively. AUC and G-mean are frequently used to evaluate the overall performance in the imbalance context since they are related to the values of both PD and PF (Wang and Yao 2013). In the VCB-SVM algorithm, lambda ( $\lambda$ ) is an empirical user-defined parameter for the penalty magnitude during each iteration. We set  $\lambda$  as 1.0 to simplify the usage of our algorithm and Boosting-SVM. M is the maximum number of iterations in the algorithm. It is set as 30. If it is too large, over-fitting problem can occur. If it is too small, the best performance of boosting cannot be attained.

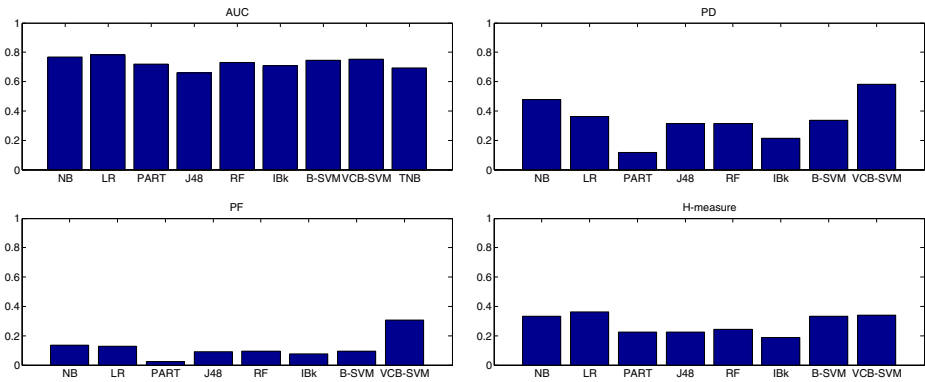
Since the performance of the classification model is sensitive to the parameters of a model (Arcuri and Fraser 2011; Song et al. 2013), we performed the parameter tuning for each learner of the WEKA. We used the CVParameterSelection to determine parameter values for the learners. It is a meta-classifier in the WEKA toolkit. The details of parameter values are described in the Appendix.

## 6 Experimental Results

### 6.1 Comparisons by Performance Measures

In this section, we show the performance results of Boosting-SVM (B-SVM) from Wang and Japkowicz (Wang and Japkowicz 2009), TNB from Ma et al. (Ma et al. 2012), selected binary classifiers from WEKA, and VCB-SVM. The performance results of MLP and SVM are omitted since their results were meaninglessly low.

At first, we show the average performance values of AUC, PD, PF and H-measure over the ten data sets of each method. Then, we summarize the results of carrying out the effect size A-



**Fig. 3** Bar plots of average AUC, PD, PF, and H-measure over the ten data sets of the classifiers

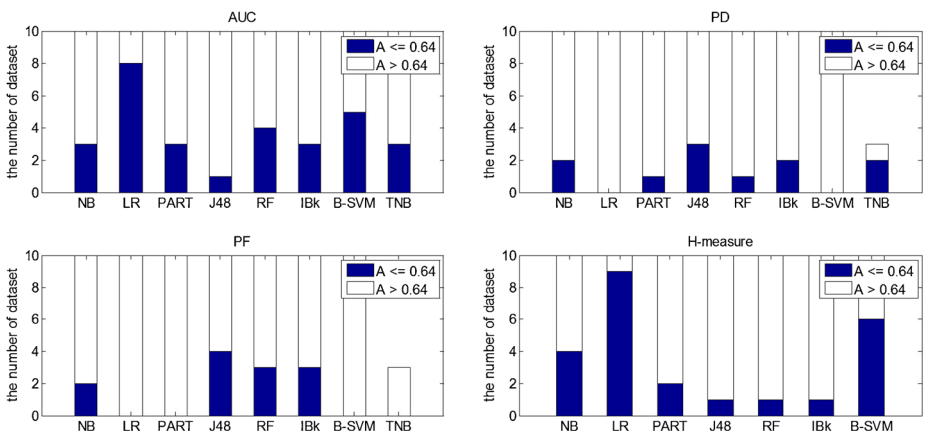
statistics (Vargha and Delaney 2000) to evaluate the magnitude of the improvement. The details of the effect size test over each data set are shown in the Appendix. We use Wilcoxon rank-sum test (Wilcoxon 1945) at a confidence level of 95 % for the statistical significance test. This non-parametric test is recommended for comparison of two classifiers (Arcuri and Briand 2011).

In Fig. 3, the overall performance of each prediction model is shown. In case of AUC and H-measure, all the classifiers show the similar performance. However, in case of PD and PF, VCB-SVM shows different results compared to the other models. Even though VCB-SVM shows the worst performance in case of PF, it shows the best performance in case of PD.

We performed the effect size A-statistics test which is recommended for evaluating randomized algorithms in software engineering (Arcuri and Briand 2011). In the A-statistics, the probability of algorithm X producing higher M values compared to another algorithm Y is computed where M is a performance measure.  $A=0.6$  means that X can produce higher results 60 % of the time.

Based on the guidelines (Vargha and Delaney 2000), if  $A > 0.64$ , X is better than Y having a medium size difference. If  $A \leq 0.64$ , X is not better than Y.

Figure 4 shows the summary of the effect size A-statistics test over ten data sets. By using A-statistics test, VCB-SVM and each classifier is compared over each data set. The number of



**Fig. 4** Summary of the effect size A-statistics test over ten data sets

each case, (i.e.,  $A > 0.64$  and  $A \leq 0.64$ ) is counted over ten data sets. In terms of AUC, PD, and H-measure,  $A > 0.64$  indicates that VCB-SVM is better than another model. In terms of PF,  $A > 0.64$  means that VCB-SVM is worse than another model. The summary shows that VCB-SVM shows worse performance in case of PF. However, in terms of AUC, PD, and H-measure, VCB-SVM shows better performance in more data sets except for LR. Even though LR outperforms VCB-SVM in terms of AUC, PF, and H-measure, it is worse than VCB-SVM in case of PD over all data sets. It indicates that LR is not practically effective for finding defects compared to VCB-SVM.

Tables 5, 6, 7 and 8 show the mean of AUC, PD, PF, and H-measure values respectively. In Tables, NASA to SOFTLAB indicates that NASA data were used as the training dataset and SOFTLAB data were used as the test dataset. NASA to NASA indicates that the NASA data were used as both the training and test datasets. The line w/t/l indicates that the results of comparing VCB-SVM with another algorithm based on Wilcoxon rank-sum test. w/t/l indicates the number of data sets VCB-SVM wins/ties/loses, compared to the classifier at the corresponding column. Values in boldface indicates the best performers over each case.

In Table 5, AUC is used as a performance measure. VCB-SVM outperforms other models except for LR. VCB-SVM wins 4 data sets, ties 4 data sets, and loses 2 data sets, compared to B-SVM. VCB-SVM wins 7 data sets, ties 2 data sets, and loses 1 data set, compared with TNB. Table 6 shows the classification results in terms of PD. In this case, VCB-SVM outperforms all the other models. VCB-SVM is better than NB over 8 data sets while it is worse than NB over 2 data sets. When comparing with B-SVM, VCB-SVM shows better performance over all data sets. Table 7 shows the PF values of each classifier on the datasets. VCB-SVM shows worse performance than all the other models. In Table 8, the classification results based on H-measure are shown. LR shows the best performance. VCB-SVM outperforms PART, J48, RF, and IBk. It shows similar performance to NB and B-SVM.

The results can be analyzed based on the two research questions.

RQ1: Does the VCB-SVM provide higher prediction performance than the CPDP techniques not prepared for the class imbalance?

To answer this question, we compare VCB-SVM with TNB. In Tables 5, 6 and 7, TNB outperforms VCB-SVM in terms of PF. The defect detection performance of VCB-SVM and

**Table 5** Comparison of the cross-project predictions when the AUC was used as the performance measure

Source ⇒ Target	NB	LR	PART	J48	RF	IBk	B-SVM	TNB	VCB-SVM
NASA to SOFTLAB									
NASA ⇒ ar3	0.814	<b>0.841</b>	0.682	0.769	0.745	0.700	0.831	0.714	0.837
NASA ⇒ ar4	0.827	<b>0.839</b>	0.805	0.706	0.782	0.816	0.835	0.692	0.838
NASA ⇒ ar5	0.900	0.938	0.770	0.886	0.826	0.810	0.929	0.827	<b>0.942</b>
NASA to NASA									
NASA ⇒ kc1	0.636	<b>0.690</b>	0.623	0.586	0.627	0.631	0.620	0.624	0.616
NASA ⇒ mc2	0.674	0.581	0.656	0.533	<b>0.686</b>	0.620	0.527	0.625	0.578
NASA ⇒ kc3	0.776	<b>0.817</b>	0.748	0.668	0.771	0.657	0.816	0.738	0.812
NASA ⇒ mw1	0.768	<b>0.799</b>	0.720	0.690	0.690	0.726	0.801	0.678	0.797
NASA ⇒ kc2	0.810	<b>0.846</b>	0.790	0.634	0.765	0.788	0.811	0.779	0.805
NASA ⇒ pc1	<b>0.717</b>	0.692	0.634	0.609	0.618	0.627	0.495	0.580	0.541
NASA ⇒ cm1	0.767	<b>0.793</b>	0.757	0.504	0.787	0.695	0.774	0.659	0.763
w/t/l	7/1/2	2/3/5	7/1/2	9/0/1	6/2/2	7/1/2	4/4/2	7/2/1	



**Table 6** Comparison of the cross-project predictions when the PD was used as the performance measure

Source $\Rightarrow$ Target	NB	LR	PART	J48	RF	IBk	B-SVM	TNB	VCB-SVM
NASA to SOFTLAB									
NASA $\Rightarrow$ ar3	0.750	0.750	0.000	0.625	0.625	0.500	0.745	0.875	<b>0.887</b>
NASA $\Rightarrow$ ar4	0.600	0.800	0.050	0.350	0.550	0.350	0.751	0.830	<b>0.853</b>
NASA $\Rightarrow$ ar5	0.875	0.875	0.000	0.625	0.500	0.375	0.812	<b>1.000</b>	0.966
NASA to NASA									
NASA $\Rightarrow$ kc1	0.109	0.126	0.099	0.197	0.218	0.231	0.125		<b>0.266</b>
NASA $\Rightarrow$ mc2	<b>0.365</b>	0.019	0.154	0.288	0.038	0.019	0.003		0.173
NASA $\Rightarrow$ kc3	0.372	0.000	0.116	0.140	0.093	0.047	0.008		<b>0.492</b>
NASA $\Rightarrow$ mw1	0.323	0.161	0.065	0.355	0.194	0.065	0.070		<b>0.713</b>
NASA $\Rightarrow$ kc2	0.346	0.252	0.093	0.280	0.280	0.196	0.257		<b>0.445</b>
NASA $\Rightarrow$ pc1	<b>0.662</b>	0.065	0.195	0.273	0.169	0.208	0.145		0.350
NASA $\Rightarrow$ cm1	0.367	0.592	0.408	0.020	0.469	0.163	0.455		<b>0.660</b>
w/t/l	8/0/2	9/1/0	9/1/0	7/2/1	9/1/0	8/2/0	10/0/0	1/2/0	

TNB are statistically same. However, the overall performance (AUC) of VCB-SVM is better. VCB-SVM wins 7 data sets, ties 2 data sets, and loses 1 data set.

RQ2: Does the VCB-SVM provide higher prediction performance than the class imbalance techniques not prepared for the cross-project learning?

To answer this question, we compare VCB-SVM with B-SVM. B-SVM outperforms VCB-SVM over all data sets in terms of PF. The overall performance (AUC and H-measure) of VCB-SVM and B-SVM are not statistically different. However, in terms of the defect detection performance, VCB-SVM outperforms B-SVM over all data sets. In software defect prediction, the most important issue is to detect the defects correctly. It means that the defect detection performance (PD) is considered far more important than the false alarm rate (PF).

NB showed a good performance in both the overall performance (AUC and H-measure) and the defect detection performance (PD). In cases of LR, PART, J48, RF, and IBk, the overall

**Table 7** Comparison of the cross-project predictions when the PF was used as the performance measure

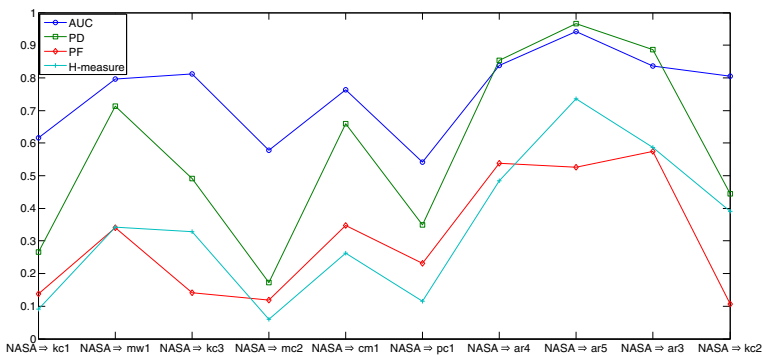
Source $\Rightarrow$ Target	NB	LR	PART	J48	RF	IBk	B-SVM	TNB	VCB-SVM
NASA to SOFTLAB									
NASA $\Rightarrow$ ar3	0.309	0.436	<b>0.018</b>	0.236	0.182	0.218	0.317	0.447	0.575
NASA $\Rightarrow$ ar4	0.126	0.287	<b>0.011</b>	0.172	0.092	0.069	0.204	0.446	0.538
NASA $\Rightarrow$ ar5	0.179	0.250	<b>0.000</b>	0.071	0.107	0.107	0.146	0.346	0.526
NASA to NASA									
NASA $\Rightarrow$ kc1	0.034	0.038	<b>0.029</b>	0.101	0.118	0.102	0.039		0.137
NASA $\Rightarrow$ mc2	0.147	<b>0.009</b>	0.028	0.128	0.037	<b>0.009</b>	<b>0.009</b>		0.119
NASA $\Rightarrow$ kc3	0.077	<b>0.000</b>	0.007	0.014	0.010	0.010	0.001		0.142
NASA $\Rightarrow$ mw1	0.059	0.016	0.016	0.046	0.081	0.013	<b>0.009</b>		0.340
NASA $\Rightarrow$ kc2	0.024	0.027	<b>0.014</b>	0.060	0.072	0.055	0.030		0.106
NASA $\Rightarrow$ pc1	0.270	<b>0.009</b>	0.029	0.079	0.073	0.084	0.046		0.231
NASA $\Rightarrow$ cm1	0.134	0.205	0.089	<b>0.013</b>	0.169	0.080	0.146		0.347
w/t/l	0/1/9	0/0/10	0/0/10	0/4/6	0/3/7	0/3/7	0/0/10	0/0/3	

**Table 8** Comparison of the cross-project predictions when the H-measure was used as the performance measure

Source $\Rightarrow$ Target	NB	LR	PART	J48	RF	IBk	B-SVM	VCB-SVM
NASA to SOFTLAB								
NASA $\Rightarrow$ ar3	0.495	<b>0.605</b>	0.168	0.426	0.398	0.178	0.543	0.543
NASA $\Rightarrow$ ar4	0.434	<b>0.497</b>	0.321	0.294	0.378	0.390	0.494	0.491
NASA $\Rightarrow$ ar5	0.603	<b>0.728</b>	0.360	0.608	0.482	0.341	0.687	0.726
NASA to NASA								
NASA $\Rightarrow$ kc1	0.126	<b>0.140</b>	0.076	0.059	0.076	0.073	0.095	0.091
NASA $\Rightarrow$ mc2	<b>0.161</b>	0.071	0.151	0.049	0.147	0.074	0.037	0.060
NASA $\Rightarrow$ kc3	0.311	<b>0.329</b>	0.243	0.197	0.243	0.161	<b>0.329</b>	<b>0.329</b>
NASA $\Rightarrow$ mw1	0.285	0.369	0.202	0.243	0.130	0.167	<b>0.381</b>	0.342
NASA $\Rightarrow$ kc2	<b>0.438</b>	0.432	0.348	0.261	0.245	0.270	0.397	0.391
NASA $\Rightarrow$ pc1	<b>0.202</b>	0.155	0.145	0.123	0.085	0.071	0.088	0.115
NASA $\Rightarrow$ cm1	0.254	<b>0.288</b>	0.230	0.005	0.253	0.145	0.272	0.263
w/t/l	6/0/4	2/0/8	8/0/2	9/1/0	9/0/1	9/0/1	3/6/1	

performances are competitive while PD is not. Boosting-SVM produced good overall performance, but its PD is low. On the contrary, TNB showed high PD values but low overall performance. Although VCB-SVM showed relatively high PF values, its overall performance and PD values are better than those of other classifiers. The classifiers which do not explicitly consider both the difference of feature distributions and the class imbalance nature showed either poor overall performance or poor PD performance. However, NB showed good performance as comparable to the VCB-SVM. It would be another research subject why NB showed good performance for CPDP. In terms of VCB-SVM, it is necessary to lower false alarms while retaining high probability of defect detection.

We tested how the performance of VCB-SVM can be influenced by the difference in the distribution between the training and the test data. The KL-divergence (Kullback and Leibler 1951) can be used to measure the difference between two probability distributions. We calculated the KL-divergence for each classification case. Figure 5 shows that the data sets have been sorted by KL-divergence in ascending order from left to right. The performance of VCB-SVM based on the distribution

**Fig. 5** Performance of VCB-SVM sorted by KL-divergence in ascending order

distance appears irregular. We would try to find the relation between the distribution distance and the performance of VCB-SVM as a future work.

## 7 Threats to Validity

Our experimental results might be affected by some threats to validity.

### 7.1 Construct Validity

To address the distributional difference between the training set and the test set, the similarity weight values are calculated by using the minimum and the maximum values of each attribute. They may not be sufficient information to obtain the similarity between source and target projects. In addition to the two characteristics, there are a variety of characteristics including the median, mean, and variance. They might better reflect the distributional characteristics and thus produce better performance in different cross-project learning cases.

We used SVM as a base classifier in our boosting approach. We employed a simple linear kernel. This indicates that our experiments are valid only when using a linear kernel. Using other kernels might lead to better or worse prediction performances.

The VCB-SVM algorithm has several configurations, i.e., the maximum number of iteration ( $M$ ), the penalty magnitude during each iteration ( $\lambda$ ), the ratio of validation set, and the number of samples for each over-/under-sampling. Our experimental results are based on single choice of values for  $M$  and  $\lambda$ , the ratio of validation set, and the number of samples for each over-/under-sampling. Different conclusions may be reached when using other values.

We performed parameter tuning for the WEKA learners used in our experiments. The parameter values are obtained based on the training set with labels. However, under the CPDP setting, the test set has different distributional characteristics from the training set. Since the parameters optimal for the training set might not be optimal for the test set, the results of the WEKA learners might be worse than those of the WEKA learners without parameter tuning.

### 7.2 Internal Validity

We divided the training instances into the instances with the similar distribution and the instances with the different distribution based on the similarity weight. Then, we focused on the misclassified instances of the similar distribution while disregarding the misclassified instances of the different distribution. We assumed all the misclassified instances of the similar distribution will have positive effects in defect prediction when they are classified correctly. In addition, we assumed all the misclassified instances of the different distribution will have negative effects in defect prediction when they are classified correctly. Such assumptions may not properly reflect the characteristics of software defect datasets. Some of the misclassified instances may have either positive or negative effect in both cases.

### 7.3 External Validity

We validated the proposed method using open source and commercial software project data with different characteristics. However, various types of software projects have their own properties, so the results might not be generalizable to other closed projects.

## 7.4 Statistical Conclusion Validity

To evaluate the magnitude of the improvement, we performed the nonparametric effect size A-statistics test. Its use in software engineering is very rare, but recommended by Arcuri and Briand (Arcuri and Briand 2011).

We carried out Wilcoxon rank-sum test to determine if there is statistically significant difference between the two classifiers. The test was applied at a 5 % significance level. Unlike the commonly used t-test, since Wilcoxon rank-sum test does not make assumption on the data distribution, it is generally recommended (Arcuri and Briand 2011).

## 8 Conclusion and Future Work

Cross-project defect prediction (CPDP) plays an important role in improving software quality in case of projects without sufficient historical data. For the success of CPDP, dataset characteristics are employed to measure the similarity between a source project and a target project.

Software defect datasets have the class imbalanced nature. Previous researches (Arisholm et al. 2010; Hall et al. 2012; GRBAC and GORAN 2013) assert that the imbalanced distribution can cause poor prediction performance of specific models. In addition, the performance measures chosen can hide the impact of imbalanced data on the real performance of classifiers. To address the class imbalance problem, data sampling, threshold moving, and ensemble methods are commonly used.

Until now there is no previous CPDP approaches considering the class imbalance problem. In this study, we investigate the applicability of the class imbalance learning for CPDP with our proposed method called the value-cognitive boosting with support vector machine (VCB-SVM). The asymmetric misclassification cost designed by the Boosting-SVM (Wang and Japkowicz 2009) and the similarity weights obtained from the distributional characteristics are seamlessly related to guide the appropriate resampling mechanism.

Through the effect size A-statistics test and Wilcoxon rank-sum test, we show our proposed method can better identify defects correctly compared to the existing CPDP methods not prepared for the class imbalance and the class imbalance method not prepared for the cross-project learning. Our proposed approach will help to design an effective prediction model for CPDP. The improved defect prediction performance with our method could help to allocate limited human and time resources more effectively to software quality assurance activities. Thus, the cost of software quality assurance control could be reduced using our proposed approach.

The proposed approach could be enhanced in several areas. First, we could use other classifiers instead of SVM as a base learner for the boosting method. Second, we may consider other transfer learning techniques, which might be more optimal for VCB-SVM. Third, we will further investigate whether other sophisticated class imbalance learning techniques instead of re-sampling methods are better related with the distributional difference between a source project and a target project. Fourth, we may apply our approach to more datasets to validate our findings.

**Acknowledgments** This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2013R1A1A2006985).

## Appendix

Table 9 shows all features in each of the NASA and SOFTLAB datasets obtained from PROMISE repository (Shepperd 2011; Menzies et al. 2012).

Table 10 shows the parameter values used for tuning them. For SVM, Radial Basis Function (RBF) kernel is employed since it is widely used.

Table 11 shows the selected parameter values for each learner according to each dataset.

Figures 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 shows the result of the effect size A-statistics test between VCB-SVM and each classifier in terms of AUC, PD, PF, and H-measure over each test set. Figure 6 shows the effect size test result of AUC, PD, PF and H-measure when ar3 dataset is used as a test set. VCB-SVM is worse than other models in terms of PF. However, it is better than other models in terms of PD. The overall performance (AUC and H-measure) of VCB-SVM is better than those of other models except for LR. Similar results can be found over the other data sets.

**Table 9** Features of the NASA and SOFTLAB datasets

Metrics	NASA							SOFTLAB		
	pc1	kc1	kc2	kc3	cm1	mw1	mc2	ar3	ar4	ar5
McCabe										
Cyclomatic_Complexity	•	•	•	•	•	•	•	•	•	•
Cyclomatic_Density				•		•	•	•	•	•
Decision_Density				•		•	•	•	•	•
Design_Complexity	•	•	•	•	•	•	•	•	•	•
Design_Density				•		•	•	•	•	•
Essential_Complexity	•	•	•	•	•	•	•			
Essential_Density				•		•	•			
Global_Data_Complexity				•			•			
Global_Data_Density				•			•			
Norm_Cyclomatic_Compl				•		•	•	•	•	•
Maintenance_Severity				•		•	•			
LOC										
LOC_Total	•	•	•	•	•	•	•	•	•	•
LOC_Blank				•		•	•	•	•	•
LOC_Executable	•	•	•	•	•	•	•	•	•	•
LOC_Comments	•	•	•	•	•	•	•	•	•	•
LOC_Code_and_Comment	•	•	•	•	•	•	•	•	•	•
Number_of_Lines				•		•	•			
Percent_Comments				•		•	•			
Halstead										
Num_Operators	•	•	•	•	•	•	•	•	•	•
Num_Operands	•	•	•	•	•	•	•	•	•	•
Num_Uniq_Operators	•	•	•	•	•	•	•	•	•	•
Num_Uniq_Operands	•	•	•	•	•	•	•	•	•	•
Length	•	•	•	•	•	•	•	•	•	•
Difficulty	•	•	•	•	•	•	•	•	•	•

**Table 9** (continued)

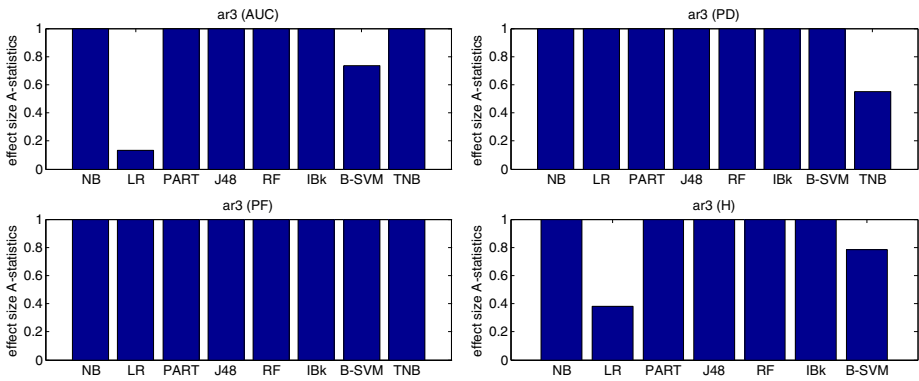
Metrics	NASA							SOFTLAB		
	pc1	kc1	kc2	kc3	cm1	mw1	mc2	ar3	ar4	ar5
Level				•		•	•	•	•	•
Volume	•	•	•	•	•	•	•	•	•	•
Programming_Effort	•	•	•	•	•	•	•	•	•	•
Programming_Time	•	•	•	•	•	•	•	•	•	•
Error_Estimate	•	•	•	•	•	•	•	•	•	•
Content				•		•	•	•	•	•
Other										
Branch_Count	•	•	•	•	•	•	•	•	•	•
Call_Pairs				•		•	•	•	•	•
Condition_Count				•		•	•	•	•	•
Decision_Count				•		•	•	•	•	•
Edge_Count				•		•	•			
Node_Count				•		•	•			
Parameter_Count				•		•	•	•	•	•
Multiple_Condition_Count				•		•	•	•	•	•
Modified_Condition_Count				•		•	•			

**Table 10** Parameter values

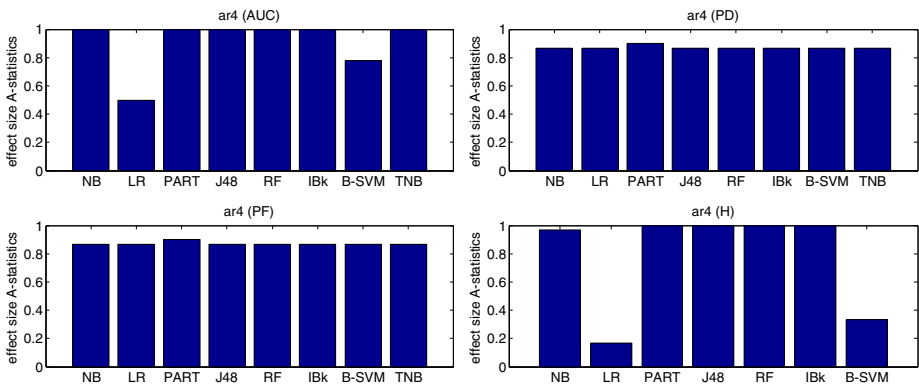
Learner	Parameters
NB	No parameter
LR	R(ridge value) = {1.0E-10, 1.0E-9, 1.0E-8, 1.0E-7, 1.0E-6}
PART	C(confidence) = {0.1, 0.2, 0.3, 0.4, 0.5}
J48	C(confidence) = {0.1, 0.2, 0.3, 0.4, 0.5}
RF	I(# trees) = {5, 10, 15, 20, 25}
Ibk	K(# neighbors) = {1, 3, 5, 7, 9}
MLP	L(learning rate) = {0.1, 0.2, 0.3, 0.4, 0.5} M(momentum) = {0.1, 0.2, 0.3, 0.4, 0.5} N(# epochs) = {100, 500, 900} H(# hidden nodes) = {1, 3, 5, 7, 9}
SVM	C(cost) = {2 <sup>-5</sup> , 2 <sup>-3</sup> , 2 <sup>-1</sup> , 2 <sup>1</sup> , 2 <sup>3</sup> } G(gamma) = {2 <sup>-5</sup> , 2 <sup>-3</sup> , 2 <sup>-1</sup> , 2 <sup>1</sup> , 2 <sup>3</sup> }

**Table 11** Parameter values for NASA dataset

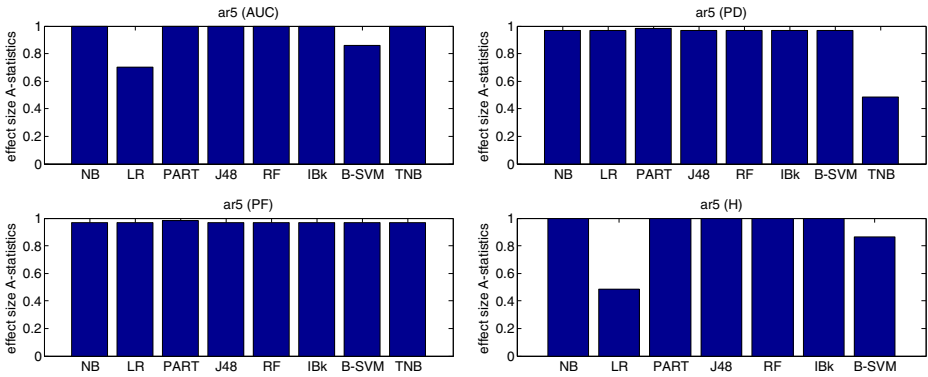
	NB	LR	PART	J48	RF	Ibk	MLP				SVM	
							R	C	C	I	K	L
All	NA	1.0E-10	0.1	0.1	20	9	0.5	0.2	900	5	2	2
w/o cm1	NA	1.0E-10	0.2	0.1	25	9	0.1	0.1	900	7	2	2
w/o kc1	NA	1.0E-10	0.2	0.1	10	7	0.5	0.3	500	5	2	2
w/o kc2	NA	1.0E-10	0.1	0.1	20	9	0.5	0.1	900	7	2	2
w/o kc3	NA	1.0E-10	0.1	0.1	20	9	0.2	0.2	100	7	2	2
w/o mc2	NA	1.0E-10	0.1	0.1	20	9	0.3	0.1	500	9	2	2
w/o mw1	NA	1.0E-10	0.1	0.1	20	9	0.1	0.3	900	3	2	2
w/o pc1	NA	1.0E-10	0.2	0.1	25	9	0.3	0.1	500	9	2	2



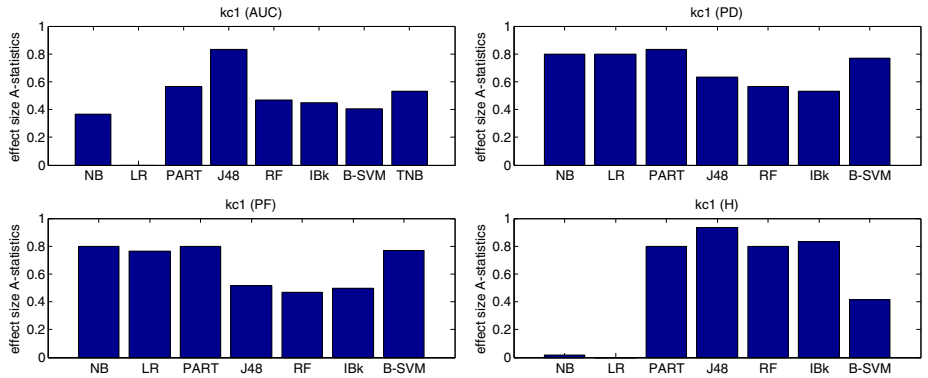
**Fig. 6** Effect size test result of AUC, PD, PF and H-measure when ar3 dataset is used as a test set



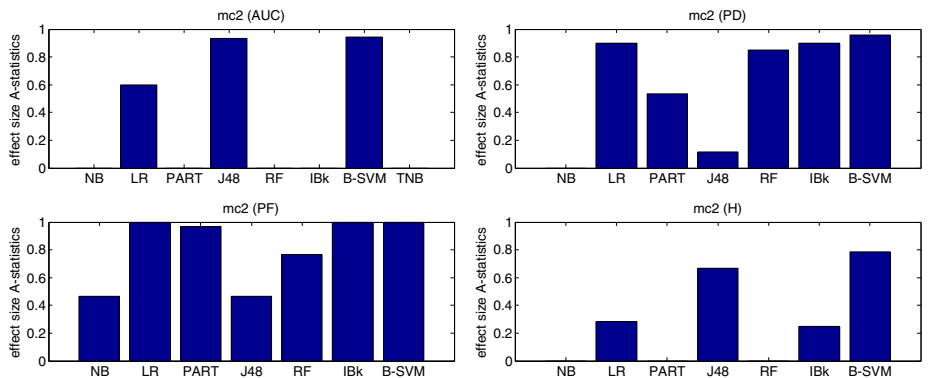
**Fig. 7** Effect size test result of AUC, PD, PF and H-measure when ar4 dataset is used as a test set



**Fig. 8** Effect size test result of AUC, PD, PF and H-measure when ar5 dataset is used as a test set



**Fig. 9** Effect size test result of AUC, PD, PF and H-measure when kc1 dataset is used as a test set



**Fig. 10** Effect size test result of AUC, PD, PF and H-measure when mc2 dataset is used as a test set



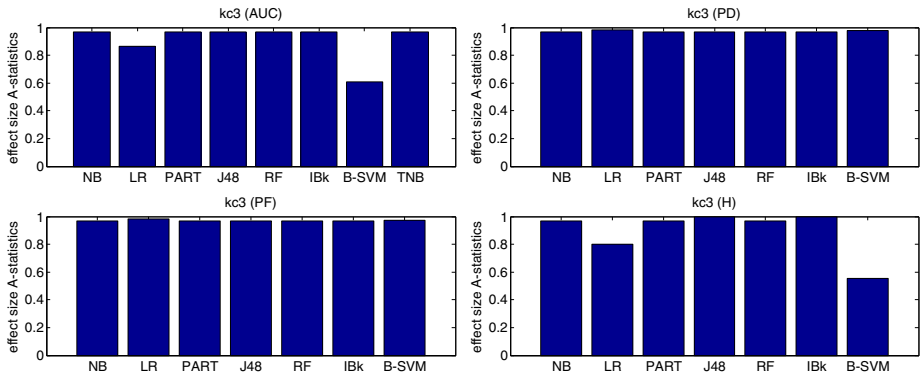


Fig. 11 Effect size test result of AUC, PD, PF and H-measure when kc3 dataset is used as a test set

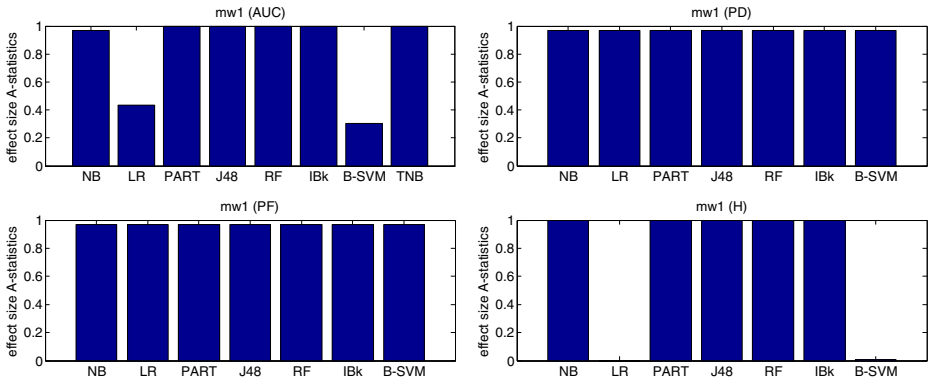


Fig. 12 Effect size test result of AUC, PD, PF and H-measure when mw1 dataset is used as a test set

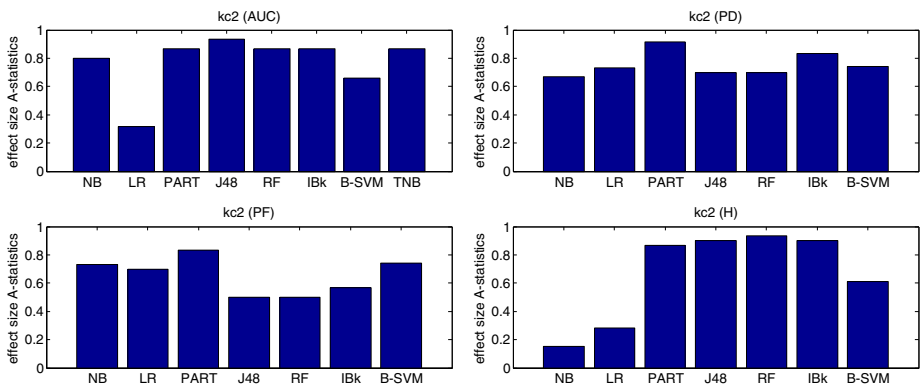
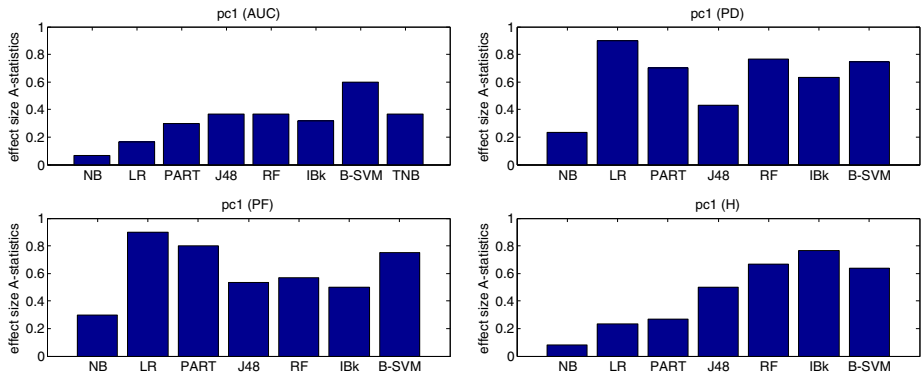
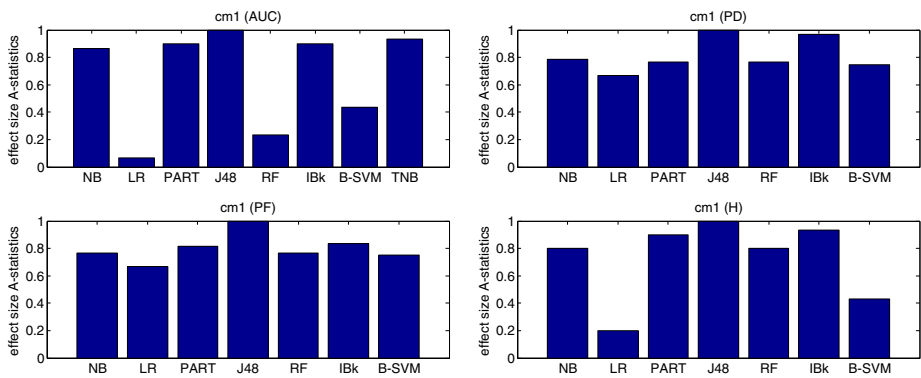


Fig. 13 Effect size test result of AUC, PD, PF and H-measure when kc2 dataset is used as a test set



**Fig. 14** Effect size test result of AUC, PD, PF and H-measure when pc1 dataset is used as a test set



**Fig. 15** Effect size test result of AUC, PD, PF and H-measure when cm1 dataset is used as a test set

## References

- Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. 2011 33rd Int Conf Softw Eng 1–10. doi: [10.1145/1985793.1985795](https://doi.org/10.1145/1985793.1985795)
- Arcuri A, Fraser G (2011) On parameter tuning in search based software engineering. Search Based Softw Eng 33–47
- Arisholm E, Briand LC, Johannessen EB (2010) A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J Syst Softw 83:2–17. doi:[10.1016/j.jss.2009.06.055](https://doi.org/10.1016/j.jss.2009.06.055)
- Bradley AP (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recogn 30:1145–1159. doi:[10.1016/S0031-3203\(96\)00142-2](https://doi.org/10.1016/S0031-3203(96)00142-2)
- Chang C, Lin C (2013) LIBSVM: a library for support vector machines. 1–39
- D’Ambros M, Lanza M, Robbes R (2011) Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empir Softw Eng 17:531–577. doi:[10.1007/s10664-011-9173-9](https://doi.org/10.1007/s10664-011-9173-9)
- Dejaeger K (2013) Toward comprehensible software fault prediction models using bayesian network classifiers. Softw Eng IEEE Trans 39:237–257
- Elish KO, Elish MO (2008) Predicting defect-prone software modules using support vector machines. J Syst Softw 81:649–660. doi:[10.1016/j.jss.2007.07.040](https://doi.org/10.1016/j.jss.2007.07.040)
- Gao K, Khoshgoftar T (2011) Software Defect Prediction for High-Dimensional and Class-Imbalanced Data. SEKE
- Garcia EA (2009) Learning from imbalanced data. IEEE Trans Knowl Data Eng 21:1263–1284. doi:[10.1109/TKDE.2008.239](https://doi.org/10.1109/TKDE.2008.239)

- Gray D, Bowes D, Davey N, et al. (2009) Using the support vector machine as a classification method for software defect prediction with static code metrics. *Eng Appl Neural Networks* 223–234
- Grbac T, Goran M (2013) Stability of software defect prediction in relation to levels of data imbalance. *SQAMIA*
- Hall M, Frank E, Holmes G (2009) The WEKA data mining software: an update. *ACM SIGKDD Explor News* 11:10–18.
- Hall T, Beecham S, Bowes D et al (2012) A systematic literature review on fault prediction performance in software engineering. *IEEE Trans Softw Eng* 38:1276–1304. doi:10.1109/TSE.2011.103
- Hand DJ (2009) Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Mach Learn* 77:103–123. doi:10.1007/s10994-009-5119-5
- He Z, Shu F, Yang Y, et al. (2011) An investigation on the feasibility of cross-project defect prediction. *Autom Softw Eng* 167–199
- Hsu C, Chang C, Lin C (2010) A practical guide to support vector classification. 1:1–16
- Kim S, Whitehead E, Zhang Y (2008) Classifying software changes: clean or buggy? *Softw Eng IEEE Trans* 34:181–196
- Kim S, Zhang H, Wu R, Gong L (2011) Dealing with noise in defect prediction. *Proceeding 33rd Int Conf Softw Eng - ICSE '11* 481. doi: 10.1145/1985793.1985859
- Kullback S, Leibler R (1951) On information and sufficiency. *Ann Math Stat*
- Lee T, Nam J, Han D, et al. (2011) Micro interaction metrics for defect prediction. *Proc 19th ACM SIGSOFT Symp 13th Eur Conf Found Softw Eng - SIGSOFT/FSE '11* 311. doi: 10.1145/2025113.2025156
- Ma Y, Luo G, Zeng X, Chen A (2012) Transfer learning for cross-company software defect prediction. *Inf Softw Technol* 54:248–256. doi:10.1016/j.infsof.2011.09.007
- Mende T, Koschke R (2009) Revisiting the evaluation of defect prediction models. *Proc 5th Int Conf Predict Model Softw Eng - PROMISE '09* 1. doi: 10.1145/1540438.1540448
- Menzies T, Dekhtyar A, Distefano J, Greenwald J (2007) Problems with precision: a response to “comments on ‘data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng* 33:637–640. doi:10.1109/TSE.2007.70721
- Menzies T, Milton Z, Turhan B et al (2010) Defect prediction from static code features: current results, limitations, new approaches. *Autom Softw Eng* 17:375–407. doi:10.1007/s10515-010-0069-5
- Menzies T, Caglayan B, He Z, et al. (2012) The PROMISE repository of empirical software engineering data. <http://promisedata.googlecode.com>
- Nam J, Pan SJ, Kim S (2013) Transfer defect learning. *2013 35th Int Conf Softw Eng* 382–391. doi: 10.1109/ICSE.2013.6606584
- Pan SJ, Yang Q (2010) A survey on transfer learning. *IEEE Trans Knowl Data Eng* 22:1345–1359. doi:10.1109/TKDE.2009.191
- Peters F, Menzies T, Gong L, Zhang H (2013) Balancing privacy and utility in cross-company defect prediction. *IEEE Trans Softw Eng* 39:1054–1068. doi:10.1109/TSE.2013.6
- Premraj R, Herzig K (2011) Network versus code metrics to predict defects: a replication study. *Int Symp Empir Softw Eng Meas* 2011:215–224. doi:10.1109/ESEM.2011.30
- Ren J, Qin K, Ma Y, Luo G (2014) On software defect prediction using machine learning. *J Appl Math* 2014:1–8. doi:10.1155/2014/785435
- Shatnawi R, Li W (2008) The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *J Syst Softw* 81:1868–1882. doi:10.1016/j.jss.2007.12.794
- Shepperd M (2011) NASA MDP software defect data sets. <http://nasa-softwaredefectdatasets.wikispaces.com/>
- Singh Y, Kaur A, Malhotra R (2009) Empirical validation of object-oriented metrics for predicting fault proneness models. *Softw Qual J* 18:3–35. doi:10.1007/s11219-009-9079-6
- Song L, Minku LL, Yao X (2013) The impact of parameter tuning on software effort estimation using learning machines. *Proc 9th Int Conf Predict Model Softw Eng - PROMISE '13* 1–10. doi: 10.1145/2499393.2499394
- Tan P-N, Steinbach M, Kumar V (2005) Introduction to data mining. *J Sch Psychol* 19:51–56. doi:10.1016/0022-4405(81)90007-8
- Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. *Empir Softw Eng* 14:540–578. doi:10.1007/s10664-008-9103-7
- Vargha A, Delaney HD (2000) A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *J Educ Behav Stat* 25:101–132. doi:10.3102/10769986025002101
- Wang BX, Japkowicz N (2009) Boosting support vector machines for imbalanced data sets. *Knowl Inf Syst* 25: 1–20. doi:10.1007/s10115-009-0198-y
- Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. *IEEE Trans Reliab* 62:434–443. doi:10.1109/TR.2013.2259203
- Wilcoxon F (1945) Individual comparisons by ranking methods. *Biom Bull* 1:80–83
- Zheng J (2010) Cost-sensitive boosting neural networks for software defect prediction. *Expert Syst Appl* 37: 4537–4543. doi:10.1016/j.eswa.2009.12.056
- Zimmermann T, Nagappan N, Gall H, et al. (2009) Cross-project defect prediction. *Proc 7th Jt Meet Eur Softw Eng Conf ACM SIGSOFT Symp Found Softw Eng* 91. doi: 10.1145/1595696.1595713



**Duksan Ryu** is a Ph.D. student at the department of computer science, KAIST. Ryu earned a bachelor's degree in computer science from Hanyang University and a Master's dual degree in software engineering from KAIST and Carnegie Mellon University. His research areas are software defect prediction and software reliability engineering



**Okjoo Choi** was a research assistant professor at the department of computer science, KAIST from 2009 to 2014. Choi earned a bachelor's degree, a master's, and Ph.D. in computer science from Sookmyung Women's University, Seoul, Korea. Before she joined KAIST, she had worked at ERP consulting services, Oracle Korea Ltd. as a consulting technical manager from 1996 to Aug, 2009 and LG Electronics Production Engineering Research Center an assistant research engineer from 1990 to 1996.



**Jongmoon Baik** received his M.S. degree and Ph.D. degree in computer science from University of Southern California in 1996 and 2000 respectively. He received his B.S. degree in computer science and statistics from Chosun University in 1993. He worked as a principal research scientist at Software and Systems Engineering Research Laboratory, Motorola Labs, where he was responsible for leading many software quality improvement initiatives. Currently, he is an associate professor in the Computer Science Department at Korea Advanced Institute of Science and Technology (KAIST). His research activity and interest are focused on software six sigma, software reliability & safety, and software process improvement.