Software Qual J (2017) 25:235–272
DOI 10.1007/s11219-015-9287-1

A transfer cost-sensitive boosting approach for cross-project defect prediction

Duksan Ryu[1] · **Jong-In Jang**[1] · **Jongmoon Baik**[1]

Published online: 1 September 2015

Abstract Software defect prediction has been regarded as one of the crucial tasks to improve software quality by effectively allocating valuable resources to fault-prone modules. It is necessary to have a sufficient set of historical data for building a predictor. Without a set of sufficient historical data within a company, cross-project defect prediction (CPDP) can be employed where data from other companies are used to build predictors. In such cases, a transfer learning technique, which extracts common knowledge from source projects and transfers it to a target project, can be used to enhance the prediction performance. There exists the class imbalance problem, which causes difficulties for the learner to predict defects. The main impacts of imbalanced data under cross-project settings have not been investigated in depth. We propose a transfer cost-sensitive boosting method that considers both knowledge transfer and class imbalance for CPDP when given a small amount of labeled target data. The proposed approach performs boosting that assigns weights to the training instances with consideration of both distributional characteristics and the class imbalance. Through comparative experiments with the transfer learning and the class imbalance learning techniques, we show that the proposed model provides significantly higher defect detection accuracy while retaining better overall performance. As a result, a combination of transfer learning and class imbalance learning is highly effective for improving the prediction performance under cross-project settings. The proposed approach will help to design an effective prediction model for CPDP. The improved defect prediction performance could help to direct software quality assurance activities and reduce costs. Consequently, the quality of software can be managed effectively.

✉ Duksan Ryu
dsryu@kaist.ac.kr

Jong-In Jang
forestar0719@kaist.ac.kr

Jongmoon Baik
jbaik@kaist.ac.kr

[1] School of Computing, Korea Advanced Institute of Science and Technology, 291 Daehak-ro (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, Republic of Korea

Keywords Boosting · Class imbalance · Cost-sensitive learning · Cross-project defect prediction · Software defect prediction · Transfer learning

1 Introduction

Software defects may inflict system failures and result in significant financial and human losses in the end. Such defects can be detected and removed by different levels of testing before the release of software. However, software testing activities are labor intensive and time consuming. It is very crucial to effectively allocate valuable resources for making a project successful. In this context, accurate software defect prediction is an important task to improve software quality by allocating valuable resources to fault-prone modules. A sufficient set of historical data is required when building a defect prediction model to attain a high prediction performance. Without a sufficient set of data within a company, cross-project defect prediction (CPDP), where data from other companies are used to build predictors, can be employed. Due to the different distributions between the source project and target project data, defect predictors learned from cross-project (CP) data show much lower prediction performance than the ones learned from within-project (WP) data (He et al. 2011; Ma et al. 2012; Nam et al. 2013; Ryu et al. 2014; Zimmermann et al. 2009). Thus, until a sufficient set of WP data is collected, how to use both sufficient CP data and insufficient WP data for building a predictor could be a key issue. In such cases, a transfer learning technique, which extracts common knowledge from the source project and transfers it to the target project, can be used to enhance the prediction performance.

In general, there is a considerably smaller number of defective examples than non-defective examples on software defect data sets. This class imbalance problem causes difficulties for the learner to predict defects. To overcome the class imbalance problem, researchers have proposed different approaches, such as cost-sensitive learning that treats the different misclassifications differently, data sampling that includes over-/under-sampling, and ensemble methods, including boosting and bagging techniques.

None of the existing CPDP approaches aggressively utilizes a small amount of WP data with considering the impact of imbalanced data. In this paper, we investigate how the cost-sensitive learning and a small amount of WP data can be utilized for CPDP. To this end, we explore the following research questions:

- RQ1: Is the combination of the transfer learning and the cost-sensitive learning more effective than each of them for CPDP?
- RQ2: Is a small amount of WP data not sufficient to build a WPDP model effective for CPDP?

The main objective of this research is to develop an effective prediction model for CPDP given a small set of labeled target data with consideration of the class imbalance. We propose a transfer cost-sensitive boosting (TCSBoost) method to overcome the problems mentioned above. The proposed approach performs boosting that assigns the correct/incorrect classification costs to the data weight vectors differently, based on the distributional characteristics and the class imbalance.

To evaluate the prediction performances, TCSBoost approach is compared with the transfer learning and class imbalance learning techniques. The performance results are analyzed based on the research of D'Ambros et al. (2011 and Menzies et al. (2010),

assessing the variability of the predictors across multiple runs. The performance between the two distributions is evaluated by using Wilcoxon's rank-sum test (Wilcoxon 1945) at a 1 % significance level. A-statistics effect size test (Vargha and Delaney 2000) is also performed to evaluate the magnitude of the improvement. The experimental results show that TCSBoost provides significantly higher defect prediction power than those of the models we compared it with.

As a result, a combination of transfer learning and class imbalance learning is highly effective for improving the prediction performance under CP settings when given a small set of labeled target data. TCSBoost will help to design an effective prediction model for CPDP. The improved defect prediction performance could help to direct software quality assurance activities and reduce development costs. Thus, the quality of software can be managed effectively.

The remainder of this paper is organized as follows. In Sect. 2, we describe related work. In Sect. 3, our proposed classification model is discussed. The experimental setup is described in Sect. 4. The results of the experiments are described in Sect. 5. We explain the threats to validity of our approach in Sect. 6. In the last section, we conclude the paper and discuss future work.

2 Related work

2.1 Software defect prediction

Software defect prediction (SDP) is a mechanism to predict defective software modules. Resource required for testing can be optimized by the virtue of SDP. To find more effective SDP models, many approaches have been proposed (Arisholm et al. 2010; D'Ambros et al. 2011; Dejaeger 2013; Elish and Elish 2008; Hall et al. 2012; Singh et al. 2009). Most approaches focused on within-project defect prediction (WPDP) models that are only applicable in cases having a sufficient set of historical data. Recently, more attention has been shown to the question of if CPDP is applicable to a project when a development organization has no or insufficient local data.

Zimmermann et al. (2009) indicated that CPDP is important for projects with little or insufficient data to construct predictors. They performed 622 CPDPs, and only 3.4 % were successful. The characteristics of the data and the process are crucial elements to lead the success of CPDP. CPDP has been considered a challenging issue that more researchers should investigate.

Turhan et al. (2009) proposed a CPDP mechanism using nearest-neighbor filtering. They pointed out that a defect classifier learned from WP data was superior to those learned from CP data. Based on the findings, they recommended a two-phase approach. In phase one, companies are recommended to use CPDP and start gathering WP data. Once enough WP data are gathered, companies should stop using CP data and employ predictors learned from WP data as phase two. They noted that WP data are very important to improving prediction performance.

Turhan et al. (2013) carried out the evaluation of the effects of mixed projects. They simulated two cases of when there are sufficient historical data of a project and when the WP data are insufficient. They examined whether mixing CP data with the WP data improves the prediction performance of both cases. The evaluation was done with 73 versions of 41 projects and Naïve Bayes classifier. The authors concluded that the mixed

project data of insufficient WP data and CP data significantly improved the performance of the defect predictor.

In our research, we investigate whether WP data are useful for CPDP although they are insufficient for WPDP. Unlike the research of Turhan et al. (2013), in which only a Naïve Bayes classifier with simple nearest-neighbor filter was used, we propose a novel CPDP model that combines transfer learning and cost-sensitive learning techniques. We analyzed the change in prediction performance of six different models when 5, 10, and 25 % of total WP data are added.

He et al. (2011) proposed a CPDP method based on the instance selection. By carrying out three experiments using 34 data sets, they asserted that the distributional attributes of data sets, e.g., the median, mean, and variance, are closely related to the prediction results.

Ma et al. (2012) presented a Transfer Naïve Bayes (TNB). This method transferred CP information to the weights of the training data, which are used to build a prediction model.

Nam et al. (2013) applied transfer component analysis (TCA) for CPDP to improve prediction performance. They showed that suitable normalization options for TCA can enhance the performance for CPDP.

Jureczko and Madeyski (2010) investigated the clustering of software projects from the perspective of defect prediction. They performed clustering on 92 versions of 38 proprietary, open-source, and academic software projects based on the characteristics they have. The authors analyzed the result, and the existence of two out of six clusters was statistically proven. This research gives insight into reusing defect predictors among the projects in a same cluster.

Ryu et al. (2014) proposed a value-cognitive boosting with a support vector machine (VCB-SVM). The authors pointed out the class imbalance problem in the software defect prediction and carried out an approach that can deal with the problem in the context of CPDP. VCB-SVM derives the similarity weights and asymmetric misclassification cost from the distribution characteristics of the training and target data set, and utilizes them to aid the effective resampling mechanism for CPDP.

Chen et al. (2015) proposed a double transfer boosting (DTB). DTB utilizes a mixed data set of target and source project. The algorithm is mainly constructed upon two levels of data transfer. The first part is to change the entire distribution of CP data by reweighting the instances based on a data gravitation method. The remaining part conducts the transfer boosting learning to refine the CP data by eliminating the negative data. The authors carried out the evaluation of the proposed approach with 15 selected publicly available data sets, concluding that the DTB algorithm makes an effective defect prediction model in the CPDP setting.

Previous researches (He et al. 2011; Ma et al. 2012; Nam et al. 2013; Ryu et al. 2014; Zimmermann et al. 2009) emphasized the identification of the distributional characteristics of data sets for the success of CPDP. The distributional characteristics, e.g., mean, median, minimum, and maximum, are typically used to compute the similarity between a source project and a target project. Likewise, in this study, the range between the minimum and the maximum values of the attribute, which is one of the distributional characteristics, is used to compute the similarity weight.

CPDP is a technique that is useful in the case of no or insufficient WP data. If sufficient WP data are accumulated, WPDP can be employed.

Until now, there are no existing CPDP methods considering class imbalance that use WP data aggressively. Although WP data are insufficient for WPDP, it may improve the prediction performance for CPDP.

2.2 Transfer learning

Machine learning approaches assume that the feature distribution between training data and test data is the same. If such assumption fails, the performance of prediction models will be poor. To overcome this problem, transfer learning techniques transfer knowledge extracted from a source project to construct models to the target project.

Some approaches have been presented to address the problem setting, given sufficient labeled source data and insufficient labeled target data.

Dai et al. (2007) proposed TrAdaBoost, which applies boosting to knowledge transfer. The target and source data are considered in isolation. AdaBoost (Freund and Schapire 1997) is used to increase the weight of misclassified target examples for the target data. The weighted majority algorithm is used to decrease the weight of misclassified source examples.

According to Shi et al. (2008), TrAdaBoost shows poor performance if source data are irrelevant. To address this issue, Yao and Doretto (2010) proposed the MultiSrcTrAdaBoost algorithm, which transfers knowledge from multiple sources to find one source highly related to the target.

Eaton and DesJardins (2011) presented a task-based boosting approach named TransferBoost. The boosting mechanism in this method is applied at both the instance level and the task level. If source tasks show positive transferability to the target task, they get higher weight. The weights of individual example within each source task are fine-tuned by AdaBoost.

Like these studies, methods that improve the prediction performance using WP data could be introduced into CPDP.

2.3 Class imbalance learning

The class imbalance issue indicates that the number of non-defective examples is much more than that of defective examples (Tan et al. 2005). According to Hall et al. (2012) who performed a systematic literature review on SDP, class imbalance may be related to poor performance of classification models. They pointed out that class imbalance should be addressed in more SDP studies.

The correct classification of the minority example is of greater value even though its frequency is low. If defective examples are not identified, testing efforts may not be directed and thus software quality could be significantly low. The goal of the class imbalance learning is to obtain the predictor that can produce high accuracy for the minority class without jeopardizing the accuracy of the majority class.

Various methods at data and algorithm levels have been proposed to identify the examples of the minority class effectively. Sampling techniques are widely used as a data-level method. Cost-sensitive learning methods assigning distinct costs to the training examples are proposed as an algorithm-level method.

Fan et al. (1999) proposed an AdaCost. AdaCost is a misclassification cost-sensitive boosting method. The algorithm tackles the class imbalance problem by assigning the misclassification cost of instances from majority class and minority class differently. The method that the AdaCost approach took is to conservatively decrease the weights of costly correct classifications and aggressively increase the weights of costly incorrect classifications.

Chawla et al. (2002) proposed SMOTE. The algorithm SMOTE, which stands for synthetic minority over-sampling technique, is a combination of under-sampling the majority class and over-sampling the minority class. In the over-sampling schema, SMOTE produces synthetic minority class examples, rather than just duplicating the existing examples.

Tomek (1976) proposed Tomek links, which could be used as an under-sampling or a data cleaning method. They consider only instances close to the class boundary, unlike previous methods that chose the instances randomly. Applying Tomek links method to under-sampling, the data set can aid the classification by making a fat boundary with a large margin.

Grbac and Goran (Grbac et al. 2013) investigated how the class imbalance of software defect data sets influences the performance stability of machine learning techniques. The results indicated that prediction performance could be unstable due to a high level of imbalance.

Wang et al. (2010) proposed a negative correlation learning algorithm called AdaBoost.NC. This algorithm exploits the ambiguity term in the decomposition to improve diversity.

Wang and Yao (2013) examined various class imbalance learning techniques and their effects on software defect prediction. They carried out an evaluation of resampling techniques, threshold moving, and ensemble algorithms and showed that AdaBoost.NC produced the best overall performance.

Wang and Japkowicz (2009) proposed a support vector machine with asymmetric misclassification cost to solve the skewed vector space problems. The data distribution and the classifier are both modified in their proposed Boosting-SVM algorithm. The class imbalance problem is addressed by using the property of soft margins.

In our study, first, by applying cost-sensitive learning techniques, we focus on locating the defect class having greater value in CPDP. In this way, we aim at presenting a more practical model for CPDP. Second, since the use of WP data for CPDP was not investigated in depth in the existing CPDP studies, we investigate whether WP data can be used to improve the prediction performance for CPDP.

3 TCSBoost approach

We propose a TCSBoost approach, which exploits the similarity weight. This approach aims at providing high performance by assigning distinct weights drawn from distributional characteristics and class imbalance. Figure 1 shows the overall defect prediction process using the TCSBoost method.

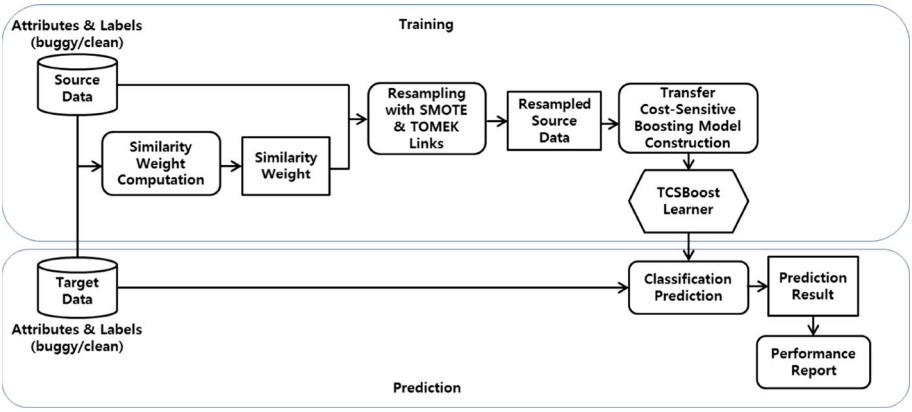

Fig. 1 Overall process with transfer cost-sensitive boosting approach

3.1 Defect data set preparation

Source and target defect data are prepared for training and testing, respectively. An example of data is marked as clean if there is no defect in it. If there is at least one defect, it is marked as buggy.

3.2 Similarity weight computation

In the second phase, the similarity weight is computed by dividing the number of similar attributes by the total number of attributes. This measure, employed in several CPDP studies (Chen et al. 2015; Ma et al. 2012; Ryu et al. 2014), is a simple way to identify the distributional characteristics between data sets.

Let us say a_{ij} is the jth attribute of x_i, given a sequence $x_i = \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$. The maximum value and minimum value of jth attribute in the test set are calculated as below:

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

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

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

where a_{ij} is the jth attribute of instance x_i and $h(a_{ij}) = \begin{cases} 1 & \text{if } \min_j \leq a_{ij} \leq \max_j \\ 0 & \text{otherwise} \end{cases}$.

3.3 Training data set resampling

SMOTE (Chawla et al. 2002) and Tomek links (Tomek 1976) are used in the proposed approach to enhance the representation of the defect class instances. In the light of the CP settings, distributional characteristics are considered when employing them. By introducing those resampling methods, the probability of detection (PD) can be enhanced even though the probability of a false alarm (PF) becomes worse. PD and PF are described in detail in the performance report phase.

SMOTE is a method that over-samples the minority class. In our approach, SMOTE is applied by taking into account the distributional differences. SMOTE is only applied to similar instances (with similarity weight = 1), since different instances are considered as negatively affecting to the prediction performance.

Our approach used Tomek links (Tomek 1976) as an under-sampling method. Ii and Ij are two different class instances and d(Ii, Ij) is the distance between Ii and Ij. A (Ii, Ij) is defined as a Tomek link if there is no instance Il, such that d(Ii, Il) < d(Ii, Ij) or d(Ij, Il) < d(Ii, Ij). Either one of the instances in a Tomek link is noise, or both instances are borderline. Thus, Tomek links can be utilized as an under-sampling method, meaning that only instances belonging to the majority class are eliminated.

It may not be helpful to under-sample the majority class instances based on a minority class instance belonging to a different distribution. Thus, the minority class instances in

different distributions are excluded in the process of under-sampling. Based on the minority instances from the same distribution, the majority instances from all distributions are under-sampled. By repeatedly carrying out Tomek link-based under-sampling, overlapping around the class boundary can be reduced, and thus PD values can be increased (and usually PF values are also increased). We repeated Tomek link-based under-sampling five times.

3.4 Transfer cost-sensitive boosting model construction

We present a TCSBoost dealing with feature distributional differences between source and target projects, as well as the class imbalance between buggy and clean classes. Algorithm 1 describes our proposed algorithm.

Algorithm 1. TCSBoost algorithm

Input parameters

S: source project data

T: target project data

SW: similarity weight

Local variables

X_{train}: training set of input sequence

M: the maximum number of iterations

N: the number of training examples in X_{train}

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

w: the data weight vector

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

α: the weight coefficient

c_n: the cost factor ($0\leq c_n\leq 1$)

Function calls:

h: base learner

Train(X): train a base learner h

Classify(X,h): classify X by the learner h

I: an indicator function where I(false) = 0, I(true) = 1

β: the cost adjustment function

1. Initialize

   $X_{train} = S \cup T$

   $w_n = 1/N$ for n = 1, …, N.

2. For m = 1, 2, …, M:

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

   $h_m \leftarrow \text{Train}(X_{train})$

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

   $$\alpha_m = \lambda \ln\{\frac{1-\varepsilon_m}{\varepsilon_m}\}$$

   $w_n \leftarrow w_n \exp\{-\alpha_m t_n h_m \beta(n)\}$ where $\beta(n) = \beta(\text{sign}(t_n, h_m), c_n)$

Output

the hypothesis $H = \text{sign}[\sum_{j=1}^{m} \alpha_j h_j]$

As input parameters, source project data (S), a small amount of target project data (T), and the similarity weight (SW) are given. In the first step, source project data and a small set of target project data are formed as a training set to produce a final ensemble classifier. The initial weight vector of the training instances is set as 1/N. In the second step, the boosting algorithm is repeatedly applied to the training data. The base classifier (h) uses the data weight vector (w_n) for training. The quantities (ε_m) represent the weighted error rates of each base learner. The weighting coefficient (α_j) gives higher weights to more accurate learners when calculating the final hypothesis (H).

In general boosting algorithms, e.g., AdaBoost (Freund and Schapire 1997) and AdaCost (Fan et al. 1999), the data weight vector (w_n) is increased for misclassified examples and decreased for correctly classified examples in successive iterations. Thus, subsequent learners concentrate on those examples misclassified by previous learners. Under the CPDP environments, we propose different weighting policies within the boosting algorithm. The core concept of the proposed algorithm is about how effectively the distributional characteristics and the class imbalance nature can be combined together. Unlike AdaCost, which only addresses the class imbalance, our approach considers the distributional characteristics as well as the class imbalance. In the approach, we propose a cost adjustment function adapted for CPDP. We assign correct/incorrect classification costs differently, depending on the similarity between training data and test data. We disregard the misclassified instances of different distribution while concentrating on the misclassified instances of similar distribution. To this end, source project data (S) are divided into instances having similar distribution (X_S) and instances having different distribution (X_D) based on the similarity weight, indicating that $S = X_S \cup X_D$. The similarity weight represents how much each instance is similar to test data according to distributional characteristics. If the similarity weight equals to one, all the attributes of an instance are similar to those of test data.

The labeled target data having the same distribution (T) and source project instances having similar distribution (X_S) are treated in the same way. For the two data sets, we conservatively decrease the weights of instances that are correctly classified, and conservatively increase the weights of costly misclassifications. For source project data having the different distribution (X_D), we conservatively decrease the weights of instances that are correctly classified and conservatively decrease the weights of costly misclassifications.

The cost adjustment function ($\beta(n)$) is defined as $\beta(n) = \beta(\text{sign}(t_n, h_m), c_n)$, where $\text{sign}(t_n, h_m)$ is positive for correct classification (β_+) and negative for misclassification (β_-). We applied the cost adjustment function in accordance with the proposed weighting policy to the experiments:

- The cost adjustment function for the instances having the same or similar distribution
  $\beta_+(c) = -0.25 \cdot c + 0.25$
  $\beta_-(c) = 0.25 \cdot c$
- The cost adjustment function for the instances having different distributions
  $\beta_+(c) = -0.25 \cdot c + 0.25$
  $\beta_-(c) = 0.25 \cdot c - 0.5$

3.5 Classification prediction

In this phase, the final ensemble classifier predicts whether unseen data in a target project are defective or not. The training and test sets are extracted from the same project in WPDP. Thus, they have the same space feature and the same distribution feature. In CPDP,

Table 1 Confusion matrix

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

the training and test sets come from different projects. Thus, although they have the same feature space, they might have different feature distributions. Such different feature distributions are known as the main obstacle to prediction performance.

3.6 Performance report

The learner built on the software defect data sets, which have the imbalanced nature, is typically evaluated by both the performance on the buggy class and the overall performance. The performance on the buggy class is commonly measured by the probability of detection (PD) and the probability of a false alarm (PF). Confusion matrix for the performance evaluation is given in Table 1. True positive (TP) is the number of buggy modules correctly predicted as buggy. True negative (TN) is the number of clean modules correctly predicted as clean. False positive (FP) is the number of clean modules predicted as buggy. False negative (FN) is the number of buggy modules predicted as clean. PD is computed by: $\text{PD} = \frac{\text{TP}}{(\text{TP}+\text{FN})}$. PD, also known as the recall, represents the proportion of appropriate instances retrieved. PF is calculated by: $\text{PF} = \frac{\text{FP}}{(\text{FP}+\text{TN})}$. The performance of PF, also called the false positive rate, is better when its value is lower, in contrast to PD.

Geometric mean (G-mean) and Balance are utilized for the overall evaluation of predictors in the imbalanced context. G-mean is the geometric mean of recall values of the defect class (PD) and the non-defect class (1-PF) such as the following: $\text{G-mean} = \sqrt{\text{PD}(1-\text{PF})}$. A predictor with a high G-mean would have high accuracies on both classes and thus be a good predictor. The measure Balance is introduced by calculating the Euclidean distance from the real (PD, PF) point to (1, 0), since the point ($\text{PD} = 1, \text{PF} = 0$) is the ideal point where all defects are detected without missing. By definition, $\text{Balance} = 1 - \frac{\sqrt{(0-\text{PF})^2+(1-\text{PD})^2}}{\sqrt{2}}$. To sum up, in our experiment, PD and PF are computed for the defect class. PD and PF are desired to be high and low, respectively, for a good predictor. Then, G-mean and Balance are used to assess the overall performance. Like PD, higher G-mean and Balance are desired. The evaluation of the overall performance aims at showing how well the learner can balance the performance between the buggy and the clean classes.

4 Experimental setup

To answer the following two research questions, we set up comparative experiments.

- RQ1: Is the combination of transfer learning and cost-sensitive learning more effective than each of them for CPDP?

- RQ2: Is a small amount of WP data not sufficient to build a WPDP model effective for CPDP?

For RQ1, we compared our approach with Naïve Bayes, Naïve Bayes by applying SMOTE (Chawla et al. 2002), Burak Filter (Turhan et al. 2009), AdaCost (Fan et al. 1999), and TransferBoost (Eaton and DesJardins 2011). AdaCost is a cost-sensitive learning method utilizing the cost adjustment function. As a transfer learning method, TransferBoost uses WP data to enhance the performance.

For RQ2, we investigate how 5, 10, and 25 % of target training data can affect the prediction performance of CPDP. Labeled target training data are used as the training set of all classification models, along with source project data.

To compare the performance of our approach with other classifiers, 15 data sets (Marian 2010; Jureczko and Madeyski 2010) from the PROMISE repository (Menzies et al. 2012) are employed. Since G-mean and Balance are related to the values of both PD and PF, they are commonly considered effective performance measures in the context of the class imbalance. Thus, G-mean and Balance are chosen for the experiments. There is a trade-off between the defect detection performance (e.g., PD) and the overall performance (e.g., G-mean and Balance) (Wang and Yao 2013). Consequently, obtaining a learner to provide high accuracy for the minority class without severely lowering the accuracy of the majority class is important. PD is also considered to be a practically useful measure, according to Menzies et al. (2007).

To answer RQ1, we formalize the following hypotheses:

- $H1_0$: TCSBoost is not better than transfer learning and class imbalance learning techniques.
- $H1_A$: TCSBoost is better than transfer learning and class imbalance learning techniques.

We also need to consider the context of class imbalance while evaluating prediction performance. If the overall performances among classifiers are statistically the same, a classifier producing better PD values is considered more effective than the others. If the PD

Table 2 Projects in the Jureczko data set

Project	# Instances	# Buggy	% Buggy	Description
ant	125	20	16	Open source
arc	234	27	11.5	Academic
camel	339	13	3.8	Open source
e-Learning	64	5	7.8	Academic
jedit	272	90	33.1	Open source
log4j	135	34	25.2	Open source
lucene	195	91	46.7	Open source
poi	237	141	59.5	Open source
prop-6	660	66	10	Proprietary
redaktor	176	27	15.3	Academic
synapse	157	16	10.2	Open source
systemdata	65	9	13.8	Open source
tomcat	858	77	9	Open source
xalan	723	110	15.2	Open source
xerces	162	77	47.5	Open source

Table 3 Features of the Jureczko data sets

Type	Features
Chidamber and Kemerer (1994)	Weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object classes (CBO), response for a class (RFC), lack of cohesion in methods (LCOM)
Henderson-Sellers (1995)	Lack of cohesion in methods (LCOM3)
Bansiya and Davis (2002)	Number of public methods (NPM), data access metric (DAM), measure of aggregation (MOA), measure of functional abstraction (MFA), cohesion among methods of class (CAM)
Tang et al. (1999)	Inheritance coupling (IC), coupling between methods (CBM), average method complexity (AMC)
Martin (1994)	Afferent couplings (Ca), efferent couplings (Ce)
McCabe (1976)	Maximum McCabe's cyclomatic complexity (Max(CC)), average McCabe's cyclomatic complexity (Avg(CC))
Lines of code	Lines of code (LOC)

performances among classifiers are statistically the same, a classifier producing better overall performance values is considered more effective than the others.

The following hypotheses are formalized to answer RQ2.

- $H2_0$: The prediction performance is not increased as the percentage of WP data used as a training set increases.
- $H2_A$: The prediction performance is increased as the percentage of WP data used as a training set increases.

We consider WP data are effective if the prediction performance increases as the percentage of WP data used as a training set increase.

4.1 Data collection

The Jureczko data sets were originally collected by Jureczko and Madeyski (2010). The instances of data sets are represented with 20 independent attributes; static code features based mainly on Chidamber and Kemerer (1994) object-oriented metrics and one dependent label that indicates whether the instance is defective or not. We chose 15 data sets among all of the Jureczko data sets obtained from the PROMISE repository (Menzies et al. 2012), as Chen et al. (2015) did. As shown in Table 2, the selected 15 data sets include a proprietary project built for customer solutions, three academic projects done by students, and 11 open-source projects collected from Apache projects. In Table 3, features of the Jureczko datasets are described.

We chose each of these data set to be test data, and used the remaining data sets as training data, to set up the CPDP setting. Each experiment of CPDP was iteratively performed 30 times.

4.2 Learning algorithms

We employed Naïve Bayes as a base learner for the TCSBoost and implemented the WEKA machine learning toolkit (Hall et al. 2009). We applied z-score normalization to all of the training and test data. Other learning algorithms, i.e., Naïve Bayes, Naïve Bayes with SMOTE, Burak Filter, AdaCost, and TransferBoost, were compared with our proposed

method. As the base learner for AdaCost and TransferBoost, we utilized Naïve Bayes from the WEKA machine learning toolkit as well. The reason why we chose Naïve Bayes is that it generally shows high prediction performance in SDP compared with other classification models (Hall et al. 2012; Ryu et al. 2014). We applied z-score normalization to all of training and test data before running those algorithms, except for the Burak Filter. For that relevancy filter, we normalized all instances with the log-normal transformation method, as shown in Turhan et al. (2009).

4.3 Parameter configurations

Lambda (λ) is an empirical user-defined parameter in the boosting algorithm, which is used for the penalty magnitude during each iteration. For AdaCost and TransferBoost, we set the value of λ as 0.5 and the maximum number of iterations (M) in the algorithm was set as 50. The value was carefully adjusted, since if it is too large, over-fitting problem can occur, and if it is too small, the effectiveness of boosting algorithm can be reduced. Considering efficiency and effectiveness, M is set as 30 and λ is set as 0.6 for TCSBoost. For TCSBoost and AdaCost, the cost factors of the majority class are set as 0.5 and 0.1, respectively. The cost factor of the minority class is fixed as 1.0. We performed all the combinations of the cost factors in our experiments. In our experiments, AdaCost performed best when the cost factor of the majority class was 0.1. For AdaCost, as suggested by Fan et al. (1999), the cost adjustment function was used as the following: $\beta_+(c) = -0.5 \cdot c + 0.5$, $\beta_-(c) = 0.5 \cdot c + 0.5$. For TransferBoost, source project data were divided into ten tasks by using k-means clustering algorithm.

5 Experimental results

Tables 4, 5, 6, 7, 8, and 9 show the prediction performance of classification models using CP data, as well as 5, 10, and 25 % of WP data during the training phase. Values in boldface represent the best performers for each experiment.

In Table 4, the median performance values of classification models using 5 % of WP data in terms of PD and PF are given. Figure 2 shows scatter plots of median PD and PF values of six models over 15 data sets using 5, 10, and 25 % of WP data. Since high PD and low PF are desired, the better predictor has more points at the bottom right of the areas.

NB shows the worst PD (0.210) and the best PF (0.054), and thus, all points are positioned at the bottom left of the areas in Fig. 2. NB with SMOTE (NB + SMOTE) shows low performance (0.375) even though it produces 75 % higher PD than NB. As opposed to NB and NB + SMOTE, TransferBoost produces the best PD (0.937) and the worst PF (0.620), showing more points at the top right of the areas. Through the nearest-neighbor-based relevancy filtering technique, Burak Filter reduces the PF, but the PF value is still high (0.512). AdaCost and TCSBoost show more balanced performance with regard to PD and PF. AdaCost (PD: 0.5, PF: 0.153) and TCSBoost (PD: 0.593, PF: 0.340) show higher PD than NB (0.210) and NB + SMOTE (0.375), while showing lower PF than TransferBoost (0.620) and Burak Filter (0.512). The practically useful predictor should provide high accuracy for the defect class without lowering the accuracy of the clean class, meaning that acceptably high PD and low PF values are desired. Compared with AdaCost, TCSBoost shows 18 % higher defect detection performance in terms of the total median PD.

Table 5 shows the median G-mean and Balance values of each classifier on the data sets. On the whole, TCSBoost outperforms other classification models (G-mean: 0.641; Balance: 0.636). SMOTE was effective for improving the prediction performance (G-

Table 4 Median PD and PF performance of classification models using 5 % of WP data

Target Data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF
ant	0.210	**0.100**	0.473	0.195	0.578	0.350	0.947	0.505	**1.000**	0.620	0.842	0.400
arc	0.192	**0.050**	0.384	0.086	0.615	0.512	0.384	0.081	**0.980**	0.887	0.461	0.235
camel	0.500	**0.045**	0.500	0.093	0.666	0.617	0.500	0.100	**1.000**	0.848	0.583	0.385
e-Learning	0.000	**0.000**	0.250	0.017	**1.000**	0.500	0.500	0.017	0.500	0.080	0.500	0.107
jedit	0.305	**0.104**	0.411	0.132	**0.847**	0.491	0.411	0.115	0.693	0.436	0.588	0.381
log4j	0.187	**0.010**	0.250	0.031	0.937	0.666	0.406	0.062	**1.000**	0.942	0.593	0.125
lucene	0.104	**0.030**	0.244	0.080	**0.639**	0.323	0.279	0.151	0.581	0.333	0.546	0.282
poi	0.134	**0.043**	0.208	0.065	**0.865**	0.615	0.473	0.153	0.805	0.494	0.708	0.340
prop-6	0.158	**0.054**	0.365	0.143	0.650	0.374	0.587	0.294	**1.000**	0.989	0.634	0.364
redaktor	0.115	**0.070**	0.269	0.161	**1.000**	0.859	0.269	0.122	**1.000**	0.978	0.730	0.535
synapse	0.333	**0.082**	0.600	0.149	**0.933**	0.671	0.733	0.175	0.866	0.443	0.866	0.470
systemdata	0.375	**0.000**	0.375	0.037	0.750	0.471	0.750	0.358	**0.937**	0.820	0.500	0.132
tomcat	0.506	**0.122**	**0.739**	0.196	0.219	0.221	0.712	0.216	0.726	0.246	**0.739**	0.225
xalan	0.567	**0.270**	0.663	0.322	0.932	0.647	0.668	0.405	**0.971**	0.857	0.788	0.478
xerces	0.219	**0.148**	0.287	0.246	**0.410**	0.598	0.287	0.222	0.383	0.493	0.328	0.308
Median	0.210	**0.054**	0.375	0.132	0.750	0.512	0.500	0.153	**0.937**	0.620	0.593	0.340

Table 5 Median G-mean and Balance performance of classification models using 5 % of WP data

Target data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal
ant	0.435	0.437	0.621	0.605	0.613	0.612	0.670	0.627	0.542	0.519	**0.710**	**0.695**
arc	0.427	0.427	0.591	0.560	0.547	0.546	**0.594**	0.561	0.325	0.371	0.590	**0.582**
camel	**0.689**	**0.644**	0.672	0.639	0.506	0.505	0.669	0.638	0.386	0.399	0.602	0.602
e-Learning	0.000	0.292	0.495	0.469	0.694	**0.646**	**0.700**	**0.646**	0.495	0.472	0.674	0.640
jedit	0.525	0.504	0.596	0.572	**0.656**	**0.636**	0.603	0.576	0.594	0.571	0.604	0.604
log4j	0.430	0.425	0.492	0.469	0.559	0.526	0.621	0.578	0.232	0.332	**0.725**	**0.701**
lucene	0.318	0.366	0.473	0.462	**0.662**	**0.662**	0.492	0.481	0.478	0.467	0.623	0.621
poi	0.358	0.387	0.445	0.439	0.579	0.555	0.632	0.611	0.523	0.502	**0.683**	**0.682**
prop-6	0.386	0.403	0.558	0.539	**0.638**	**0.638**	0.611	0.604	0.097	0.299	0.636	0.636
redaktor	0.327	0.372	0.474	0.470	0.375	0.392	0.481	0.473	0.145	0.307	**0.582**	**0.576**
synapse	0.553	0.525	0.717	0.699	0.553	0.522	**0.779**	**0.775**	0.690	0.669	0.682	0.659
systemdata	0.612	0.558	0.606	0.557	0.629	0.622	**0.688**	**0.679**	0.409	0.416	0.665	0.637
tomcat	0.665	0.639	**0.770**	**0.768**	0.413	0.426	0.740	0.737	0.738	0.736	0.757	0.757
xalan	0.641	0.636	**0.669**	**0.669**	0.571	0.539	0.629	0.628	0.370	0.392	0.641	0.630
xerces	0.436	0.440	0.461	0.463	0.409	0.409	0.473	0.472	0.445	0.448	**0.476**	**0.477**
Median	0.435	0.437	0.591	0.557	0.571	0.546	0.629	0.611	0.445	0.448	**0.641**	**0.636**

Table 6 Median PD and PF performance of classification models using 10 % of WP data

Target data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF
ant	0.222	**0.095**	0.500	0.191	0.611	0.361	0.944	0.473	**0.972**	0.622	0.833	0.404
arc	0.166	**0.053**	0.375	0.086	0.625	0.516	0.416	0.104	**0.958**	0.840	0.458	0.236
camel	0.500	**0.044**	0.500	0.092	0.666	0.617	0.500	0.102	**1.000**	0.977	0.583	0.389
e-Learning	0.000	**0.000**	0.250	0.018	**1.000**	0.509	0.250	0.018	0.750	0.396	0.500	0.113
jedit	0.320	**0.103**	0.407	0.128	0.851	0.484	0.419	0.115	**0.975**	0.847	0.604	0.387
log4j	0.193	**0.010**	0.258	0.032	0.935	0.659	0.419	0.065	**1.000**	0.989	0.580	0.114
lucene	0.097	**0.031**	0.243	0.074	0.658	0.319	0.280	0.138	**0.761**	0.515	0.542	0.281
poi	0.133	**0.046**	0.212	0.058	0.870	0.604	0.559	0.208	**0.984**	0.988	0.700	0.337
prop-6	0.152	**0.056**	0.355	0.140	0.644	0.368	0.397	0.196	**1.000**	0.986	0.627	0.364
redaktor	0.125	**0.067**	0.250	0.156	**1.000**	0.858	0.250	0.126	**1.000**	0.992	0.750	0.529
synapse	0.285	**0.078**	0.642	0.141	0.928	0.669	0.714	0.173	**1.000**	0.992	0.857	0.464
systemdata	0.375	**0.000**	0.375	0.040	0.750	0.480	0.750	0.360	**0.812**	0.710	0.500	0.120
tomcat	0.492	**0.123**	0.739	0.196	0.224	0.222	0.724	0.243	**0.760**	0.307	0.739	0.226
xalan	0.555	**0.259**	0.646	0.307	0.929	0.643	0.656	0.384	**0.959**	0.806	0.792	0.476
xerces	0.217	**0.144**	0.275	0.243	0.420	0.578	0.289	0.210	**0.492**	0.605	0.318	0.308
Median	0.217	**0.056**	0.375	0.128	0.750	0.516	0.419	0.173	**0.972**	0.840	0.604	0.337

Table 7 Median G-mean and Balance performance of classification models using 10 % of WP data

Target data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal
ant	0.445	0.444	0.635	0.621	0.610	0.608	0.668	0.625	0.559	0.533	**0.710**	**0.696**
arc	0.398	0.409	0.585	0.553	0.549	0.548	0.587	0.574	0.384	0.402	**0.591**	**0.582**
camel	**0.689**	**0.644**	0.671	0.639	0.511	0.510	0.668	0.638	0.148	0.308	0.600	0.600
e-Learning	0.000	0.292	0.495	0.469	**0.686**	0.639	0.497	0.469	0.607	0.574	0.672	**0.640**
jedit	0.538	0.514	0.597	0.571	**0.657**	**0.639**	0.611	0.582	0.384	0.399	0.601	0.601
log4j	0.437	0.429	0.499	0.474	0.564	0.531	0.631	0.587	0.104	0.300	**0.718**	**0.693**
lucene	0.307	0.361	0.472	0.461	**0.668**	**0.668**	0.497	0.484	0.450	0.449	0.627	0.622
poi	0.357	0.386	0.444	0.441	0.580	0.559	0.653	0.641	0.075	0.300	**0.683**	**0.682**
prop-6	0.379	0.399	0.553	0.533	**0.636**	**0.635**	0.565	0.552	0.114	0.302	0.632	0.632
redaktor	0.340	0.379	0.461	0.459	0.376	0.393	0.468	0.462	0.086	0.298	**0.589**	**0.580**
synapse	0.516	0.492	0.739	0.726	0.554	0.524	**0.772**	**0.766**	0.088	0.298	0.682	0.658
systemdata	0.612	0.558	0.600	0.557	0.636	0.629	**0.692**	**0.690**	0.503	0.490	0.663	0.636
tomcat	0.656	0.630	**0.767**	**0.765**	0.417	0.428	0.743	0.740	0.714	0.714	0.754	0.753
xalan	0.640	0.635	**0.672**	**0.671**	0.574	0.542	0.631	0.631	0.431	0.428	0.643	0.632
xerces	0.437	0.439	0.458	0.460	0.419	0.419	**0.472**	**0.471**	0.439	0.444	0.464	0.466
Median	0.437	0.439	0.585	0.553	0.574	0.548	0.631	0.587	0.384	0.402	**0.643**	**0.632**

Table 8 Median PD and PF performance of classification models using 25 % of WP data

Target data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF	PD	PF
ant	0.200	**0.094**	0.466	0.189	0.833	0.461	0.533	0.233	**1.000**	0.936	0.866	0.405
arc	0.200	**0.058**	0.375	0.086	0.600	0.509	0.400	0.096	**1.000**	0.919	0.450	0.219
camel	0.500	**0.045**	0.500	0.090	0.700	0.610	0.500	0.108	**1.000**	0.950	0.600	0.377
e-Learning	0.000	**0.000**	0.250	0.022	**1.000**	0.500	0.500	0.022	**1.000**	0.954	0.500	0.090
jedit	0.358	**0.102**	0.417	0.125	**0.850**	0.496	0.484	0.117	0.753	0.411	0.641	0.451
log4j	0.180	**0.013**	0.200	0.026	0.960	0.657	0.720	0.394	**1.000**	0.624	0.580	0.118
lucene	0.102	**0.038**	0.250	0.076	**0.676**	0.320	0.250	0.128	0.308	0.141	0.536	0.269
poi	0.141	**0.041**	0.216	0.062	0.853	0.618	0.471	0.124	**1.000**	1.000	0.693	0.312
prop-6	0.163	**0.054**	0.346	0.132	0.653	0.369	0.346	0.139	**0.959**	0.890	0.632	0.344
redaktor	0.100	**0.071**	0.250	0.160	**1.000**	0.848	0.250	0.107	**1.000**	1.000	0.700	0.508
synapse	0.333	**0.079**	0.666	0.141	0.916	0.669	0.833	0.273	**1.000**	1.000	0.833	0.462
systemdata	0.285	**0.000**	0.428	0.035	0.785	0.476	**0.857**	0.404	**0.857**	0.630	0.571	0.119
tomcat	0.465	**0.122**	0.724	0.188	0.224	0.225	0.706	0.188	**0.844**	0.530	0.741	0.225
xalan	0.548	**0.241**	0.627	0.280	0.914	0.636	0.658	0.341	**1.000**	0.939	0.810	0.444
xerces	0.215	**0.125**	0.275	0.218	0.396	0.601	0.275	0.218	**0.887**	0.882	0.318	0.296
Median	0.200	**0.058**	0.375	0.125	0.833	0.509	0.500	0.139	**1.000**	0.919	0.632	0.312

Table 9 Median G-mean and Balance performance of classification models using 25 % of WP data

Target data	Naïve Bayes		NB + SMOTE		Burak Filter		AdaCost		TransferBoost		TCSBoost	
	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal	G	Bal
ant	0.420	0.428	0.621	0.603	0.628	0.588	0.659	0.635	0.250	0.337	**0.718**	**0.698**
arc	0.432	0.432	0.584	0.553	0.549	0.547	**0.598**	0.571	0.283	0.349	0.593	**0.581**
camel	**0.689**	**0.644**	0.673	0.640	0.517	0.514	0.665	0.638	0.221	0.327	0.611	0.611
e-Learning	0.000	0.292	0.494	0.469	0.690	**0.646**	**0.699**	**0.646**	0.213	0.325	0.674	0.640
jedit	0.566	0.540	0.597	0.575	0.653	**0.631**	**0.655**	0.626	0.643	0.619	0.599	0.597
log4j	0.422	0.420	0.444	0.434	0.561	0.531	0.675	0.659	0.568	0.531	**0.719**	**0.692**
lucene	0.314	0.365	0.473	0.464	**0.675**	**0.673**	0.470	0.463	0.484	0.476	0.624	0.619
poi	0.369	0.392	0.449	0.444	0.567	0.549	0.637	0.615	0.000	0.292	**0.687**	**0.684**
prop-6	0.391	0.406	0.546	0.527	**0.644**	**0.643**	0.537	0.521	0.324	0.369	0.643	**0.643**
redaktor	0.306	0.362	0.463	0.460	0.389	0.400	0.477	0.465	0.000	0.292	**0.585**	**0.580**
synapse	0.552	0.524	**0.748**	**0.738**	0.550	0.520	0.739	0.732	0.000	0.292	0.690	0.663
systemdata	0.534	0.494	0.646	0.595	0.638	0.626	0.636	0.597	0.563	0.542	**0.709**	**0.685**
tomcat	0.637	0.611	**0.767**	**0.763**	0.420	0.431	0.738	0.731	0.629	0.608	0.756	0.754
xalan	0.646	0.637	**0.672**	**0.672**	0.576	0.545	0.653	0.653	0.245	0.335	0.667	0.655
xerces	0.431	0.436	0.456	0.459	0.409	0.409	0.473	0.472	0.323	0.371	**0.481**	**0.479**
Median	0.431	0.432	0.584	0.553	0.567	0.547	0.653	0.626	0.283	0.349	**0.667**	**0.643**

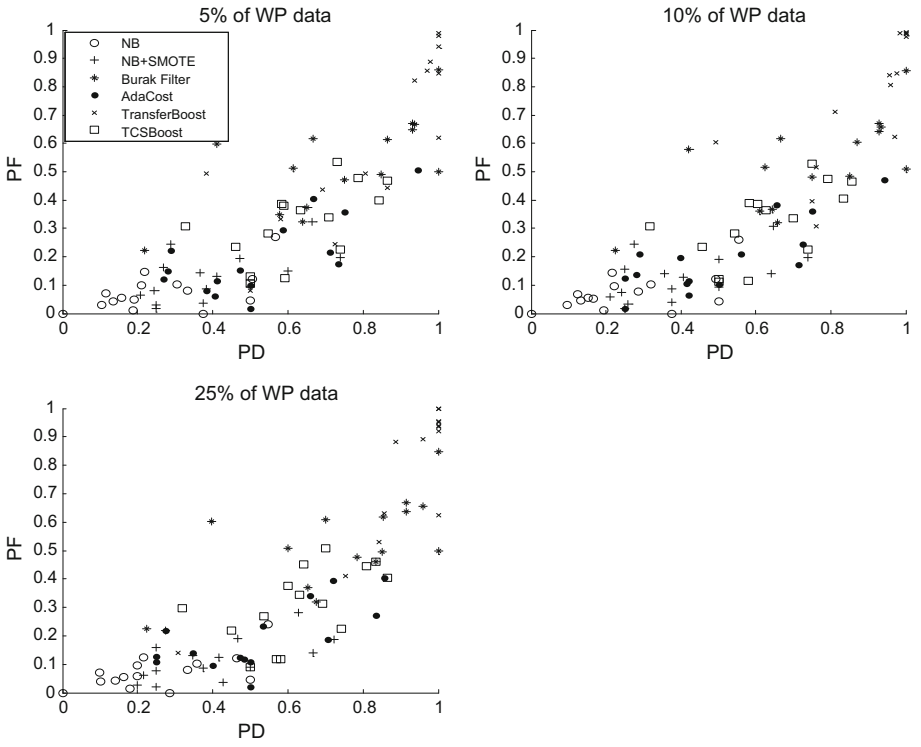

Fig. 2 Scatter plots of median PD and PF values of six models over 15 data sets using 5, 10, and 25 % of WP data

mean: 0.591; Balance: 0.557) compared to NB (G-mean: 0.435; Balance: 0.437). AdaCost, a cost-sensitive learning approach, is the second best predictor.

In Table 6, the median performance values of classification models using 10 % of WP data in terms of PD and PF are shown. The performance results among six classification models are consistent with the results using 5 % of WP data.

In Table 7, G-mean and Balance are used as performance measures for classification models using 10 % of WP data. The performance results are similar to the results using 5 % of WP data. TCSBoost outperforms other classification models overall.

Table 8 shows the median performance values of classification models using 25 % of WP data in terms of PD and PF. The performance results among six predictors are consistent with the results using 5 and 10 % of WP data.

Table 9 shows G-mean and Balance values used as performance measures for classification models using 25 % of WP data. The performance results are similar to the results using 5 and 10 % of WP data. The overall performance of TCSBoost is superior to those of other classification models.

5.1 RQ1

The following RQ1 is answered in this subsection.

- RQ1: Is the combination of the transfer learning and the cost-sensitive learning more effective than each of them alone for CPDP?

We analyzed the performance results based on the research of D'Ambros et al. (2011) and Menzies et al. (2010), evaluating the variability of the predictors across multiple runs. The first, second (median), and third quartile of each CP case were calculated and sorted by their medians and then displayed by using mini box plot. A bar represents the first–third quartile range and a circle indicates the median. The minimum and maximum values are not displayed. For each performance measure, the 30 data points for each target project were merged. Four hundred and fifty data points for 15 projects are used to compute the first, second, and third quartile. The performance results for individual projects using 5 % of WP data are illustrated in Appendix. To investigate which predictors are better than others, the Wilcoxon rank-sum test (Wilcoxon 1945) at a 1 % significance level was performed. In addition, A-statistics effect size test (Vargha and Delaney 2000) recommended by (Arcuri and Briand 2011) was performed to evaluate the magnitude of the improvement. Based on the guidelines from Vargha and Delaney (2000), the A-statistics greater than 0.64 for PD, G-mean, and Balance (or less than 0.36 for PF) indicates a medium effect size. For the two tests, 450 data points for 15 projects were used.

Figures 3, 4, and 5 show mini box plots of median PD, PF, G-mean, and Balance values of six models for all the projects using 5, 10, and 25 % of WP data, sorted by median.

Tables 10, 11, and 12 show the comparison of TCSBoost with classification models using 5, 10, and 25 % of WP data, based on the Wilcoxon's rank-sum test at a 1 % significance level and A-statistics effect size test. The boldface in the table shows the

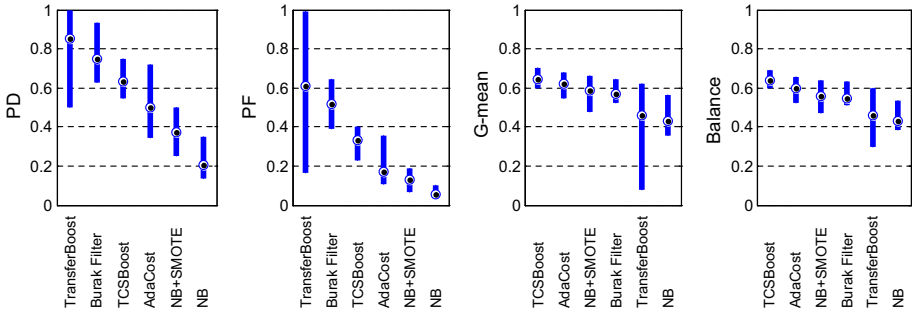

Fig. 3 Mini box plots of median PD, PF, G-mean, and Balance values of six models over 15 data sets using 5 % WP data

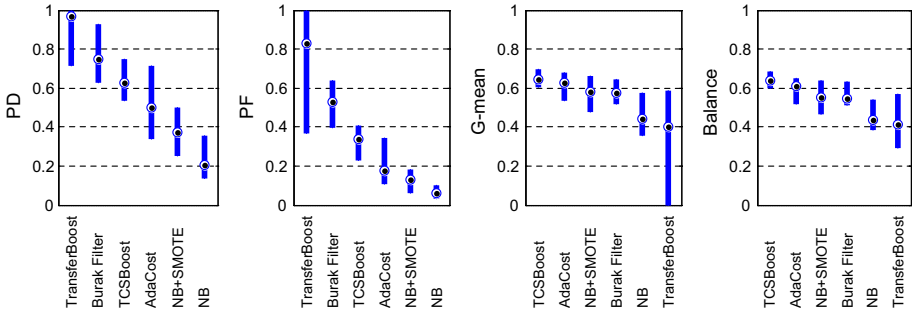

Fig. 4 Mini box plots of median PD, PF, G-mean, and Balance values of six models over 15 data sets using 10 % WP data

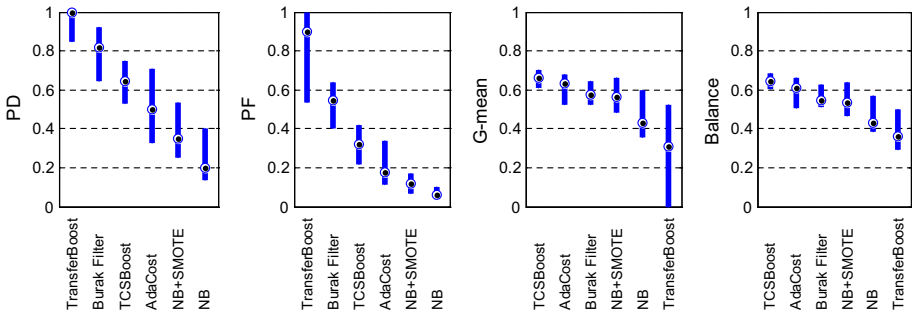

Fig. 5 Mini box plots of median PD, PF, G-mean, and Balance values of six models over 15 data sets using 25 % WP data

Table 10 Comparison of TCSBoost with classification models using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001
	A-statistics	**0.95**	**0.84**	0.32	**0.65**	0.35
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**
	A-statistics	0.95	0.87	**0.16**	0.68	**0.34**
G-mean	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.86**	**0.71**	**0.76**	0.60	**0.80**
Balance	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.88**	**0.74**	**0.77**	0.63	**0.82**

Table 11 Comparison of TCSBoost with classification models using 10 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001
	A-statistics	**0.95**	**0.84**	0.31	**0.64**	0.21
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**
	A-statistics	0.94	0.87	**0.16**	0.68	**0.21**
G-mean	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.86**	**0.70**	**0.75**	0.58	**0.83**
Balance	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.89**	**0.73**	**0.76**	0.61	**0.84**

significantly better result of TCSBoost with p value < 0.01 or A-statistics > 0.64 (For PF, A-statistics < 0.36).

In Table 10, the Wilcoxon's rank-sum test showed that the differences in G-mean and Balance values between TCSBoost and other predictors were statistically significant with p value $\ll 0.001$. However, in case of AdaCost, the effect size for G-mean and Balance is 0.6 and 0.63, respectively, indicating a small effect. In this case, we additionally consider PD values for the evaluation of H1, as we describe in Sect. 4. In terms of PD, the effect size is 0.65, indicating a medium effect. Therefore, we reject $H1_0$. In cases of 10 and 25 % of WP data, H1 can be

Table 12 Comparison of TCSBoost with classification models using 25 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001
	A-statistics	**0.95**	**0.84**	0.30	**0.65**	0.16
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**
	A-statistics	0.94	0.86	**0.14**	0.67	**0.15**
G-mean	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.87**	**0.71**	**0.75**	0.61	**0.87**
Balance	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**0.90**	**0.73**	**0.77**	**0.64**	**0.88**

evaluated in the same way. Noticeably, the G-mean and Balance performance of TCSBoost shows the least variability, indicating that its performance is stable regardless of WP data.

The experimental results show that TCSBoost provides significantly higher defect prediction power than those of the models we compared with. TCSBoost is worse than other classification models in terms of PF. However, it shows better performance in terms of PD, G-mean and Balance. PD is considered more valuable than PF. Thus, the combination of the transfer learning and the cost-sensitive learning can be considered a practical approach that can effectively identify defects.

5.2 RQ2

The following RQ2 is addressed in this subsection.

- RQ2: Is a small amount of WP data not sufficient to build a WPDP model effective for CPDP?

In Figs. 6, 7, 8, and 9, mini box plots of median PD, PF, G-mean, and Balance values of six models over 15 data sets using 5, 10, and 25 % of WP data are shown.

Tables 13, 14, and 15 show the comparison of the model using 25 % of WP data with the model using 5 or 10 % of WP data. The boldface in the table shows the significantly better result of the model using 25 % of WP data with p value < 0.01 or A-statistics > 0.64 (For PF, A-statistics < 0.36).

In Tables 13, 14, and 15, the Wilcoxon's rank-sum test showed that the difference in performance values between 25 and 5 % of WP data was not statistically significant, with p value > 0.01. The case between 25 and 10 % of WP data was the same as well. Therefore, we fail to reject $H2_0$. The performances of all predictors except for TransferBoost remain unchanged as the percentage of WP data increase. The performance of

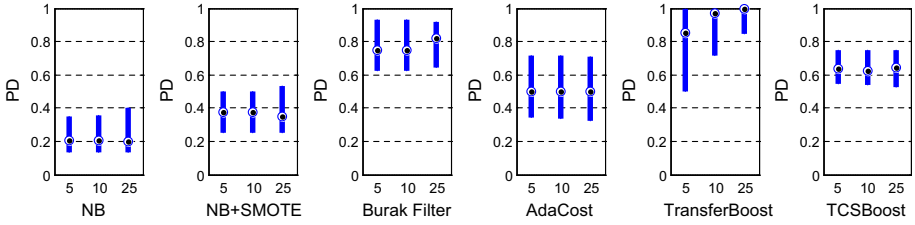

Fig. 6 Mini box plots of median PD values of six models over 15 data sets using 5, 10, and 25 % WP data

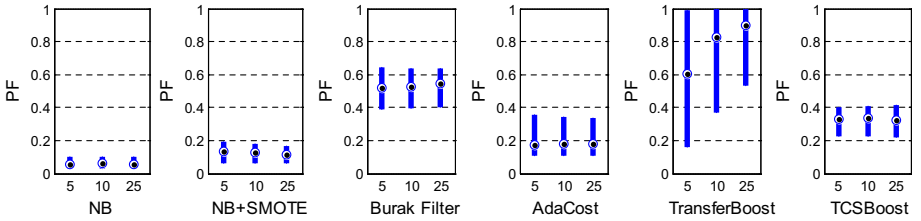

Fig. 7 Mini box plots of median PF values of six models over 15 data sets using 5, 10, and 25 % WP data

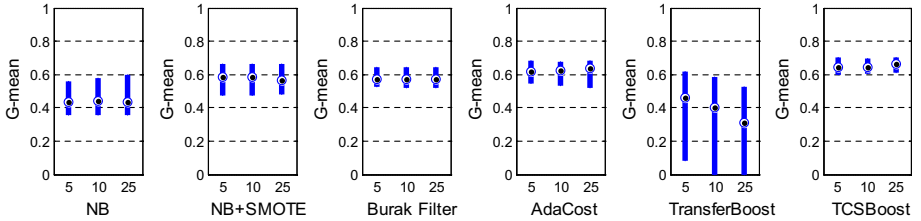

Fig. 8 Mini box plots of median G-mean values of six models over 15 data sets using 5, 10, and 25 % WP data

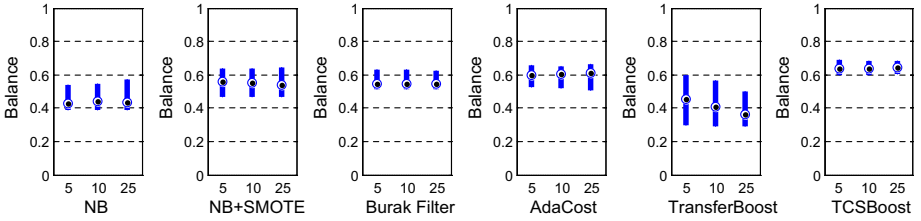

Fig. 9 Mini box plots of median Balance values of six models over 15 data sets using 5, 10, and 25 % WP data

Table 13 Comparison of NB and NB + SMOTE using 25 % of WP data with those using 5 and 10 % of WP data

NB				NB + SMOTE			
25 % vs.		5 %	10 %	25 % vs.		5 %	10 %
PD	*p* value	0.882	0.828	PD	*p* value	0.647	0.823
	A-statistics	0.49	0.49		A-statistics	0.49	0.49
PF	*p* value	0.856	0.987	PF	*p* value	0.418	0.579
	A-statistics	0.50	0.49		A-statistics	0.48	0.48
G-mean	*p* value	0.859	0.826	G-mean	*p* value	0.886	0.983
	A-statistics	0.49	0.49		A-statistics	0.49	0.50
Balance	*p* value	0.912	0.767	Balance	*p* value	0.682	0.959
	A-statistics	0.49	0.49		A-statistics	0.49	0.50

Table 14 Comparison of Burak Filter and AdaCost using 25 % of WP data with those using 5 and 10 % of WP data

Burak Filter				AdaCost			
25 % vs.		5 %	10 %	25 % vs.		5 %	10 %
PD	*p* value	0.425	0.468	PD	*p* value	0.685	0.705
	A-statistics	0.51	0.51		A-statistics	0.49	0.49
PF	*p* value	0.535	0.434	PF	*p* value	0.915	0.919
	A-statistics	0.51	0.51		A-statistics	0.49	0.50
G-mean	*p* value	0.696	0.947	G-mean	*p* value	0.973	0.800
	A-statistics	0.50	0.50		A-statistics	0.50	0.49
Balance	*p* value	0.723	0.933	Balance	*p* value	0.996	0.876
	A-statistics	0.50	0.49		A-statistics	0.49	0.49

Table 15 Comparison of TransferBoost and TCSBoost using 25 % of WP data with those using 5 and 10 % of WP data

TransferBoost				TCSBoost			
25 % vs.		5 %	10 %	25 % vs.		5 %	10 %
PD	*p* value	$\ll$**0.001**	$\ll$**0.001**	PD	*p* value	0.923	0.611
	A-statistics	0.63	0.56		A-statistics	0.49	0.50
PF	*p* value	$\ll$**0.001**	0.069	PF	*p* value	0.280	0.257
	A-statistics	0.61	0.53		A-statistics	0.47	0.47
G-mean	*p* value	$\ll$**0.001**	0.110	G-mean	*p* value	0.290	0.126
	A-statistics	0.41	0.46		A-statistics	0.52	0.52
Balance	*p* value	$\ll$**0.001**	0.101	Balance	*p* value	0.266	0.096
	A-statistics	0.41	0.46		A-statistics	0.52	0.53

TransferBoost decreases as the percentage of WP data increases. This indicates that simply adding a specific percentage of WP data may not help to enhance the prediction performance for CPDP. Thus, we assume that it is necessary to utilize a small amount of WP data systematically, aimed at providing the high prediction performance for CPDP.

6 Threats to validity

6.1 Construct validity

The range between minimum and maximum values is used to compute the similarity weights. This method may not be sufficient for identifying the distributional characteristics between source and target projects.

We assign classification costs differently according to the similarity weights based on the feature distributional characteristics and class imbalance. Depending on the distributional difference and class imbalance, the degree of increasing/decreasing the data weights is determined via the cost adjustment function. This mechanism may be too simple to reflect both the distributional difference and the class imbalance sufficiently.

There are several configurations in the proposed algorithm, e.g., the maximum number of iteration (M), the penalty magnitude in each boosting iteration (λ), and the cost factor (c). Since we have chosen single values for M, λ, and c, experiments with other values may lead to different conclusions.

6.2 External validity

Open-source software project data from Promise repository were used to validate our proposed method. Our findings might not be generalizable to other closed software projects, due to their different distributional characteristics.

6.3 Statistical conclusion validity

We performed Wilcoxon's rank-sum test at a 1 % significance level to check statistically significant differences. It is usually recommended for comparing the performance between two classifiers. A-statistics effect size test is employed to evaluate the magnitude of improvement. This method is recommended for assessing randomized algorithms in software engineering (Arcuri and Briand 2011).

7 Conclusion and future work

Software defect prediction plays an important role in improving software quality by directing resource allocation to buggy modules. In cases where local data are not sufficient, CPDP employing defect data from other projects is used to improve the prediction performance. Typically, to tackle the distributional difference, the similarity weights based on distributional characteristics between the source and the target projects are computed and then transferred to build a classification model.

Software defect data sets have the class imbalance problem. The imbalanced distribution can lead to poor prediction performance of specific models. Techniques known to be effective for dealing with the class imbalance are data sampling, cost-sensitive learning, and ensemble methods.

Applicability of the cost-sensitive learning and a small amount of WP data for CPDP is investigated in this paper. We calculate the classification cost based on distributional similarity and class imbalance, and they are used to focus on/disregard instances that increase/decrease the prediction performance. In addition, we investigate whether a small amount of WP data can be used to enhance the prediction performance of six classification models for CPDP.

Through Wilcoxon's rank-sum test, we show that TCSBoost considering distributional difference and the class imbalance is better in terms of the overall performance and the probability of detection (PD). Since our approach is particularly better in the PD and the overall performance (i.e., G-mean and Balance), it is practically useful in the context of software defect prediction. We show that simply adding a small amount of WP data for training may not be helpful for CPDP. To sum up, by directing resources on defective modules correctly, our approach can help reduce the cost of software quality assurance control.

We can further enhance TCSBoost in several ways. Other transfer learning techniques that might be more effective for our approach can be studied. In addition, we can study whether there exist optimal class imbalance learning techniques for CPDP that maximize the PD while minimizing the PF.

Acknowledgments This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science, ICT and Future Planning (MSIP)) (No. NRF-2013R1A1A2006985) and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.R0101-15-0144, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices).

Appendix

Figures 10, 11, 12, and 13 show mini box plots of median PD, PF, G-mean, and Balance values of six models for each target project over 15 data sets using 5 % of WP data. Tables 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, and 30 show the comparison of TCSBoost with classification models for each target project using 5 % of WP data. The boldface in the table shows the significantly better result of TCSBoost with p value < 0.01 or A-statistics > 0.64 (For PF, A-statistics < 0.36).

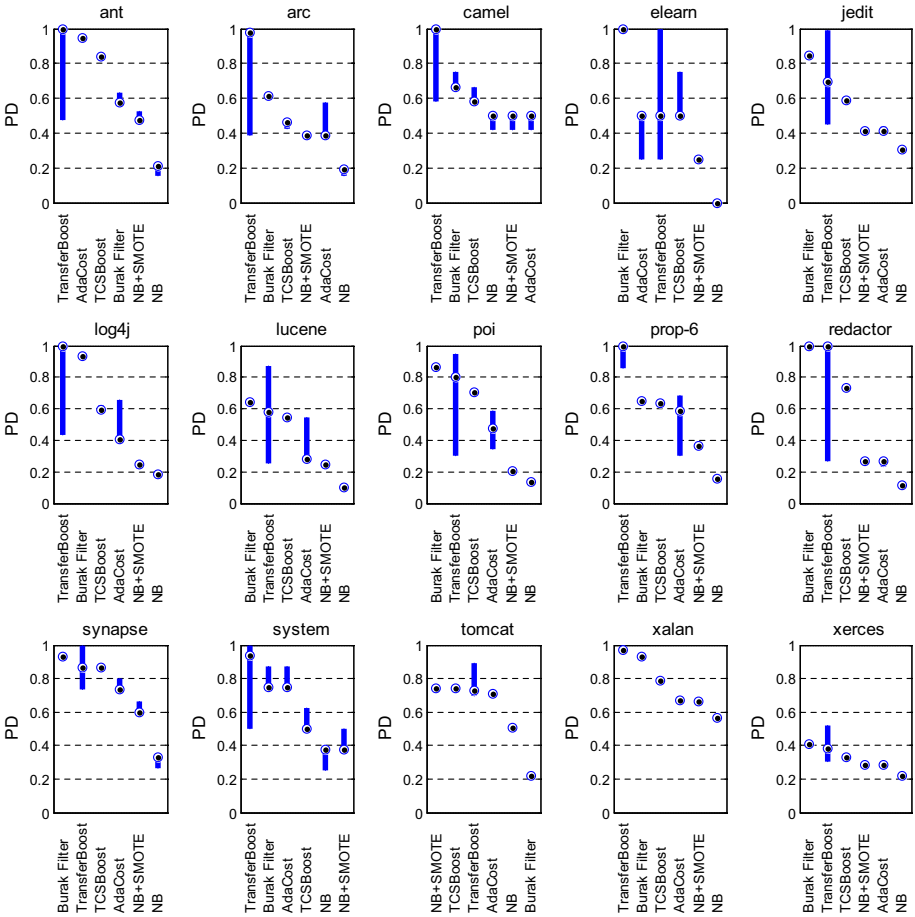

Fig. 10 Mini box plots of median PD values of six models over 15 data sets using 5 % WP data

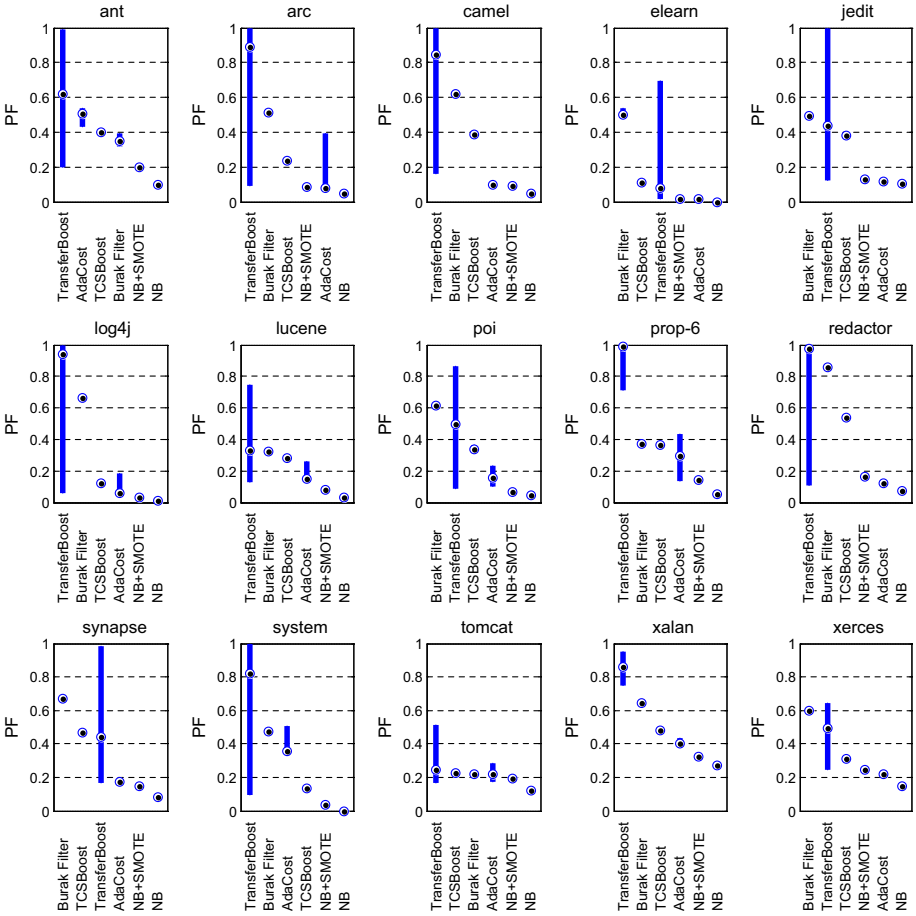

Fig. 11 Mini box plots of median PF values of six models over 15 data sets using 5 % WP data

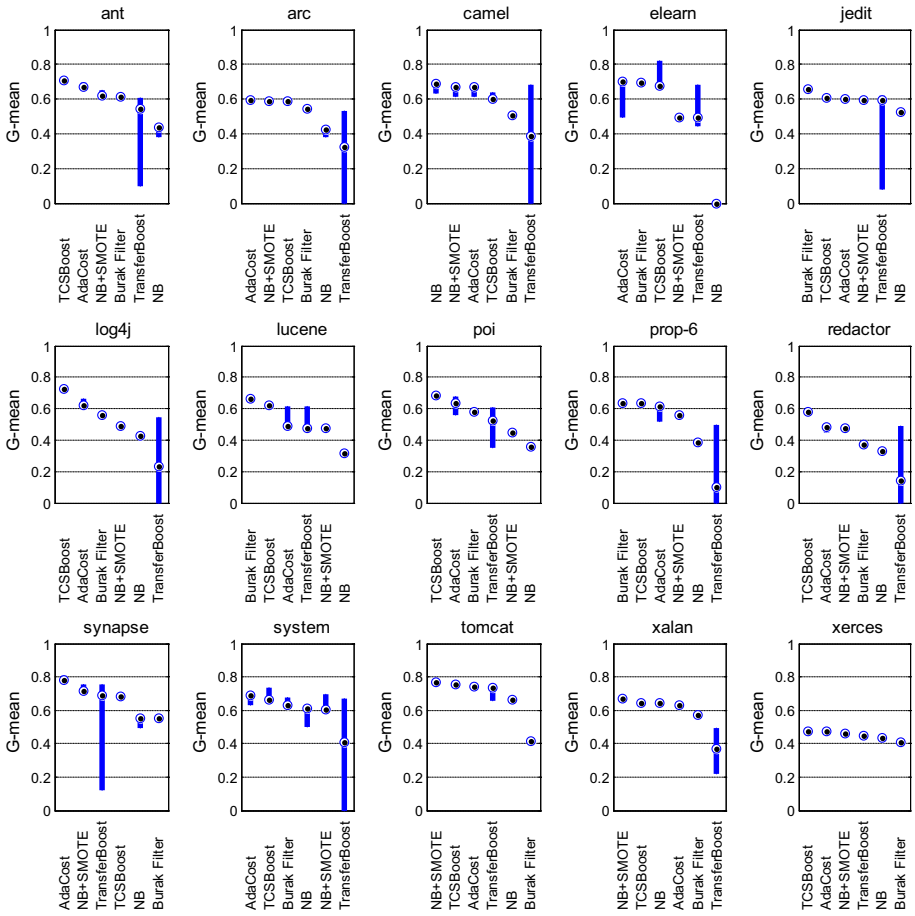

Fig. 12 Mini box plots of median G-mean values of six models over 15 data sets using 5 % WP data

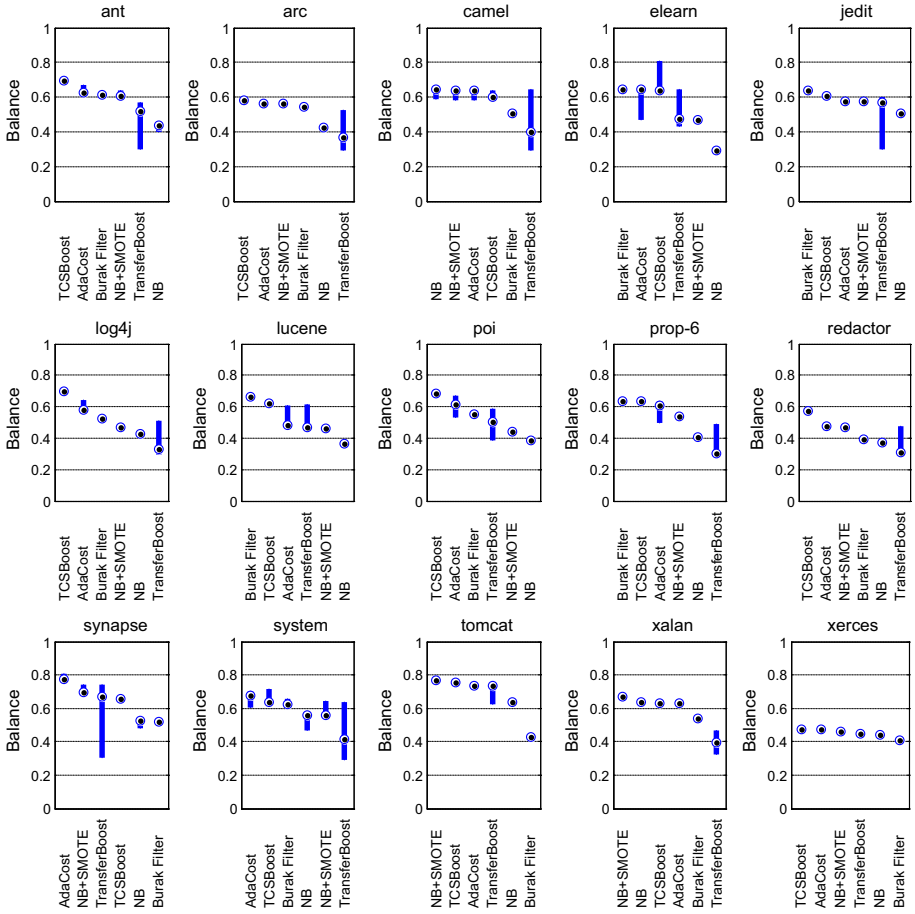

Fig. 13 Mini box plots of median Balance values of six models over 15 data sets using 5 % WP data

Table 16 Comparison of TCSBoost with classification models for the ant project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	≪0.001	0.108
	A-statistics	**1.0**	**1.0**	**0.96**	0.20	0.38
PF	*p* value	≪0.001	≪0.001	≪0.001	**≪0.001**	0.180
	A-statistics	1.0	1.0	0.80	**0.2**	0.39
G-mean	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.83**	**0.97**
Balance	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.97**	**0.94**

Table 17 Comparison of TCSBoost with classification models for the arc project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	0.174	0.006
	A-statistics	**1.0**	**1.0**	0.0	0.6	0.3
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	0.182	**0.007**
	A-statistics	1.0	1.0	**0.0**	0.6	0.3
G-mean	p value	$\ll$**0.001**	0.958	$\ll$**0.001**	0.228	$\ll$**0.001**
	A-statistics	**1.0**	0.50	**0.99**	0.41	**0.87**
Balance	p value	$\ll$**0.001**	**0.003**	$\ll$**0.001**	0.416	$\ll$**0.001**
	A-statistics	**1.0**	**0.71**	**0.97**	0.56	**0.89**

Table 18 Comparison of TCSBoost with classification models for the camel project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001
	A-statistics	**0.97**	**0.98**	0.17	**0.93**	0.25
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	0.026
	A-statistics	1.0	1.0	**0.0**	0.96	0.33
G-mean	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	0.140
	A-statistics	0.17	0.18	**0.99**	0.18	0.61
Balance	p value	0.053	0.864	$\ll$**0.001**	0.946	0.016
	A-statistics	0.35	0.48	**1.0**	0.50	**0.68**

Table 19 Comparison of TCSBoost with classification models for the e-learning project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	**0.001**	0.808
	A-statistics	**1.0**	**1.0**	0.02	**0.72**	0.51
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	0.400
	A-statistics	1.0	1.0	**0.0**	0.83	0.56
G-mean	p value	$\ll$**0.001**	$\ll$**0.001**	0.645	0.117	$\ll$**0.001**
	A-statistics	**1.0**	**1.0**	0.53	0.61	**0.80**
Balance	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	0.075	$\ll$**0.001**
	A-statistics	**1.0**	**1.0**	**0.71**	0.63	**0.79**

Table 20 Comparison of TCSBoost with classification models for the jedit project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	≪**0.001**	≪**0.001**	≪0.001	≪**0.001**	0.053
	A-statistics	**1.0**	**1.0**	0.0	**0.97**	0.35
PF	*p* value	≪0.001	≪0.001	≪**0.001**	≪0.001	0.238
	A-statistics	1.0	1.0	**0.04**	1.0	0.41
G-mean	*p* value	≪**0.001**	0.023	≪0.001	0.917	0.257
	A-statistics	**1.0**	**0.67**	0.0	0.5	0.58
Balance	*p* value	≪**0.001**	≪**0.001**	≪0.001	≪**0.001**	**0.001**
	A-statistics	**1.0**	**0.95**	0.0	**0.86**	**0.73**

Table 21 Comparison of TCSBoost with classification models for the log4j project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	≪**0.001**	≪**0.001**	≪0.001	**0.004**	0.006
	A-statistics	**1.0**	**1.0**	0.0	**0.7**	0.3
PF	*p* value	≪0.001	≪0.001	≪**0.001**	0.007	**0.001**
	A-statistics	1.0	1.0	**0.0**	0.7	**0.26**
G-mean	*p* value	≪**0.001**	≪**0.001**	≪**0.001**	≪**0.001**	≪**0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.96**	**1.0**
Balance	*p* value	≪**0.001**	≪**0.001**	≪**0.001**	≪**0.001**	≪**0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.92**	**0.99**

Table 22 Comparison of TCSBoost with classification models for the lucene project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	≪**0.001**	≪**0.001**	≪0.001	**0.001**	0.396
	A-statistics	**1.0**	**1.0**	0.0	**0.74**	0.43
PF	*p* value	≪0.001	≪0.001	≪**0.001**	≪0.001	0.518
	A-statistics	1.0	1.0	**0.03**	0.78	0.45
G-mean	*p* value	≪**0.001**	≪**0.001**	≪0.001	≪**0.001**	≪**0.001**
	A-statistics	**1.0**	**1.0**	0.0	**0.81**	**0.77**
Balance	*p* value	≪**0.001**	≪**0.001**	≪0.001	≪**0.001**	≪**0.001**
	A-statistics	**1.0**	**1.0**	0.0	**0.81**	**0.77**

Table 23 Comparison of TCSBoost with classification models for the poi project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	≪0.001	**≪0.001**	0.174
	A-statistics	**1.0**	**1.0**	0.0	**0.99**	0.39
PF	*p* value	≪0.001	≪0.001	**≪0.001**	≪0.001	0.174
	A-statistics	1.0	1.0	**0.0**	0.99	0.39
G-mean	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.86**	**0.93**
Balance	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**0.91**	**0.94**

Table 24 Comparison of TCSBoost with classification models for the prop-6 project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	≪0.001	0.071	≪0.001
	A-statistics	**1.0**	**1.0**	0.20	0.63	0.13
PF	*p* value	≪0.001	≪0.001	**≪0.001**	0.007	**≪0.001**
	A-statistics	1.0	1.0	**0.12**	0.7	**0.13**
G-mean	*p* value	**≪0.001**	**≪0.001**	0.130	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	0.38	**0.78**	**0.96**
Balance	*p* value	**≪0.001**	**≪0.001**	0.153	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	0.39	**0.79**	**0.99**

Table 25 Comparison of TCSBoost with classification models for the redaktor using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	≪0.001	**≪0.001**	0.165
	A-statistics	**1.0**	**1.0**	0.0	**1.0**	0.4
PF	*p* value	≪0.001	≪0.001	**≪0.001**	≪0.001	0.180
	A-statistics	1.0	1.0	**0.0**	1.0	0.4
G-mean	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**1.0**	**0.98**
Balance	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	**1.0**	**1.0**	**1.0**	**1.0**	**0.97**

Table 26 Comparison of TCSBoost with classification models for the synapse project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	0.749
	A-statistics	**1.0**	**1.0**	0.10	**0.93**	0.52
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	1
	A-statistics	1.0	1.0	**0.0**	0.83	0.5
G-mean	p value	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	1
	A-statistics	**1.0**	0.0	**1.0**	0.16	0.5
Balance	p value	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$0.001	1
	A-statistics	**1.0**	0.0	**1.0**	0.16	0.5

Table 27 Comparison of TCSBoost with classification models for the systemdata project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$0.001	$\ll$0.001	0.008
	A-statistics	**1.0**	**0.87**	0.0	0.06	0.31
PF	p value	$\ll$0.001	$\ll$0.001	$\ll$**0.001**	$\ll$**0.001**	0.133
	A-statistics	1.0	1.0	**0.0**	**0.06**	0.38
G-mean	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	0.367	$\ll$**0.001**
	A-statistics	**1.0**	**0.74**	**0.78**	0.56	**0.79**
Balance	p value	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**	0.383	$\ll$**0.001**
	A-statistics	**1.0**	**0.74**	**0.75**	0.56	**0.79**

Table 28 Comparison of TCSBoost with classification models for the tomcat project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	p value	$\ll$**0.001**	0.279	$\ll$**0.001**	$\ll$**0.001**	0.561
	A-statistics	**1.0**	0.57	**1.0**	**0.81**	0.54
PF	p value	$\ll$0.001	$\ll$0.001	0.087	0.958	0.411
	A-statistics	1.0	0.97	0.62	0.50	0.43
G-mean	p value	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**1.0**	0.17	**1.0**	**0.76**	**0.8**
Balance	p value	$\ll$**0.001**	$\ll$0.001	$\ll$**0.001**	$\ll$**0.001**	$\ll$**0.001**
	A-statistics	**1.0**	0.20	**1.0**	**0.86**	**0.86**

Table 29 Comparison of TCSBoost with classification models for the xalan project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	≪0.001	**≪0.001**	≪0.001
	A-statistics	**1.0**	**1.0**	0.0	**1.0**	0.16
PF	*p* value	≪0.001	≪0.001	**≪0.001**	≪0.001	**≪0.001**
	A-statistics	1.0	1.0	**0.0**	0.96	**0.10**
G-mean	*p* value	0.646	≪0.001	**≪0.001**	**≪0.001**	**≪0.001**
	A-statistics	0.53	0.0	**1.0**	**0.92**	**0.99**
Balance	*p* value	≪0.001	≪0.001	**≪0.001**	0.370	**≪0.001**
	A-statistics	0.22	0.0	**1.0**	0.56	**0.97**

Table 30 Comparison of TCSBoost with classification models for the xerces project using 5 % of WP data

TCSBoost vs.		NB	NB + SMOTE	Burak Filter	AdaCost	TransferBoost
PD	*p* value	**≪0.001**	**≪0.001**	≪0.001	**≪0.001**	0.055
	A-statistics	**1.0**	**0.98**	0.0	**0.98**	0.35
PF	*p* value	≪0.001	≪0.001	**≪0.001**	≪0.001	0.019
	A-statistics	1.0	0.96	**0.0**	0.99	**0.32**
G-mean	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	0.09	**≪0.001**
	A-statistics	**0.98**	**0.86**	**0.99**	0.62	**0.88**
Balance	*p* value	**≪0.001**	**≪0.001**	**≪0.001**	**0.007**	**≪0.001**
	A-statistics	**0.98**	**0.87**	**0.99**	**0.70**	**0.89**

References

Arcuri, A., & Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *33rd International Conference on Software Engineering (ICSE)* (pp. 1–10). doi:10.1145/1985793.1985795.

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software, 83*(1), 2–17. doi:10.1016/j.jss.2009.06.055.

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28*(1), 4–17. doi:10.1109/32.979986.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE : Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357.

Chen, L., Fang, B., Shang, Z., & Tang, Y. (2015). Negative samples reduction in cross-company software defects prediction. *Information and Software Technology, 62*, 67–77. doi:10.1016/j.infsof.2015.01.014.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493. doi:10.1109/32.295895.

D'Ambros, M., Lanza, M., & Robbes, R. (2011). Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empirical Software Engineering,*. doi:10.1007/s10664-011-9173-9.

Dai, W., Yang, Q., Xue, G., & Yu, Y. (2007). Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning* (pp. 193–200). http://dl.acm.org/citation.cfm?id=1273521. Accessed February 25, 2014.

Dejaeger, K. (2013). Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers. *IEEE Transactions on Software Engineering, 39*(2), 237–257. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6175912. Accessed February 25, 2014.

Eaton, E., & DesJardins, M. (2011). Selective transfer between learning tasks using task-based boosting. *AAAI*, 337–342. http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/viewFile/3752@misc/3915. Accessed June 11, 2014.

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software, 81*(5), 649–660. doi:10.1016/j.jss.2007.07.040.

Fan, W., Stolfo, S., Zhang, J., & Chan, P. (1999). AdaCost: misclassification cost-sensitive boosting. *ICML*. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AdaCost+:+Misclassification+Cost-sensitive+Boosting#0. Accessed November 25, 2014.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119–139. doi:10.1006/jcss.1997.1504.

Grbac, T., Mausa, G., & Basic, B. (2013). Stability of Software defect prediction in relation to levels of data imbalance. *SQAMIA*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.8978&rep=rep1&type=pdf. Accessed November 13, 2014.

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering, 38*(6), 1276–1304. doi:10.1109/TSE.2011.103.

Hall, M., Frank, E., & Holmes, G. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18. http://dl.acm.org/citation.cfm?id=1656278. Accessed November 13, 2014.

He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2011). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering,*. doi:10.1007/s10515-011-0090-3.

Henderson-Sellers, B. (1995). Object-oriented metrics: measures of complexity, Prentice-Hall, Inc.

Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th international conference on predictive models in software engineering—PROMISE '10*, 1. doi:10.1145/1868328.1868342.

Jureczko, M., & Spinellis, D. (2010). Using object-oriented design metrics to predict software defects. In *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej* (pp. 69–81).

Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology, 54*(3), 248–256. doi:10.1016/j.infsof.2011.09.007.

Martin, R. (1994). OO design quality metrics. *An analysis of dependencies, 12*, 151–170.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering SE, 2*(4), 308–320. doi:10.1109/TSE.1976.233837.

Mei-Huei, T., Ming-Hung, K., & Mei-Hwa, C. (1999). An empirical study on object-oriented metrics. In *Proceedings sixth international software metrics symposium (Cat. No.PR00403)* (pp. 242–249). IEEE Computer Society. doi:10.1109/METRIC.1999.809745.

Menzies, T., Caglayan, B., He, Z., Kocaguneli, E., Krall, J., Peters, F., & Turhan, B. (2012). The PROMISE Repository of empirical software engineering data. http://openscience.us/repo/.

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 Transactions on Software Engineering,*. doi:10.1109/TSE.2007.70721.

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., & Bener, A. (2010). Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering, 17*(4), 375–407. doi:10.1007/s10515-010-0069-5.

Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. In *35th International Conference on Software Engineering (ICSE)* (pp. 382–391). doi:10.1109/ICSE.2013.6606584.

Ryu, D., Choi, O., & Baik, J. (2014). Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empirical Software Engineering*. doi:10.1007/s10664-014-9346-4.

Shi, X., Fan, W., & Ren, J. (2008). Actively transfer domain knowledge. In *Machine Learning and Knowledge Discovery in Databases*, (60703110) (pp. 342–357). http://link.springer.com/chapter/10.1007/978-3-540-87481-2_23. Accessed November 29, 2014.

Singh, Y., Kaur, A., & Malhotra, R. (2009). Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Journal, 18*(1), 3–35. doi:10.1007/s11219-009-9079-6.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). Introduction to data mining. *Journal of School Psychology, 19*, 51–56. doi:10.1016/0022-4405(81)90007-8.

Tomek, I. (1976). Two modifications of CNN. *IEEE Transaction Systems, Man and Cybernetics*, 769–772. http://ci.nii.ac.jp/naid/80013575533/. Accessed January 26, 2015.

Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering, 14*(5), 540–578. doi:10.1007/s10664-008-9103-7.

Turhan, B., Tosun Mısırlı, A., & Bener, A. (2013). Empirical evaluation of the effects of mixed project data on learning defect predictors. *Information and Software Technology, 55*(6), 1101–1118. doi:10.1016/j.infsof.2012.10.003.

Vargha, A., & Delaney, H. D. (2000). A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics,*. doi:10.3102/10769986025002101.

Wang, S., Chen, H., & Yao, X. (2010). Negative correlation learning for classification ensembles. In *The 2010 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2010.5596702.

Wang, B. X., & Japkowicz, N. (2009). Boosting support vector machines for imbalanced data sets. *Knowledge and Information Systems, 25*(1), 1–20. doi:10.1007/s10115-009-0198-y.

Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability, 62*(2), 434–443. doi:10.1109/TR.2013.2259203.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80–83. http://www.jstor.org/stable/3001968. Accessed October 14, 2014.

Yao, Y., & Doretto, G. (2010). Boosting for transfer learning with multiple sources. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010*, 1855–1862. doi:10.1109/CVPR.2010.5539857.

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (p. 91). doi:10.1145/1595696.1595713.

Duksan Ryu is a Ph.D. student in School of Computing, KAIST. He 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.

Jong-In Jang is an M.S. student in School of Computing, KAIST. He earned a Bachelor's degree in School of Computing from KAIST. His research areas are software reliability engineering and requirements engineering.

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 School of Computing at Korea Advanced Institute of Science and Technology (KAIST). His research activity and interest are focused on software six sigma, software reliability and safety, and software process improvement.