Information and Software Technology 62 (2015) 67–77

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Negative samples reduction in cross-company software defects prediction

Lin Chen [a], Bin Fang [a,*], Zhaowei Shang [a], Yuanyan Tang [a,b]

[a] Department of Computer Science, Chongqing University, Chongqing 400030, China
[b] Faculty of Science and Technology, University of Macau, Macau, China

ARTICLE INFO

Article history:
Received 28 April 2014
Received in revised form 10 December 2014
Accepted 30 January 2015
Available online 19 February 2015

Keywords:
Cross-company defects prediction
Software fault prediction
Transfer learning

ABSTRACT

Context: Software defect prediction has been widely studied based on various machine-learning algorithms. Previous studies usually focus on within-company defects prediction (WCDP), but lack of training data in the early stages of software testing limits the efficiency of WCDP in practice. Thus, recent research has largely examined the cross-company defects prediction (CCDP) as an alternative solution.
Objective: However, the gap of different distributions between cross-company (CC) data and within-company (WC) data usually makes it difficult to build a high-quality CCDP model. In this paper, a novel algorithm named Double Transfer Boosting (DTB) is introduced to narrow this gap and improve the performance of CCDP by reducing negative samples in CC data.
Method: The proposed DTB model integrates two levels of data transfer: first, the data gravitation method reshapes the whole distribution of CC data to fit WC data. Second, the transfer boosting method employs a small ratio of labeled WC data to eliminate negative instances in CC data.
Results: The empirical evaluation was conducted based on 15 publicly available datasets. CCDP experiment results indicated that the proposed model achieved better overall performance than compared CCDP models. DTB was also compared to WCDP in two different situations. Statistical analysis suggested that DTB performed significantly better than WCDP models trained by limited samples and produced comparable results to WCDP with sufficient training data.
Conclusions: DTB reforms the distribution of CC data from different levels to improve the performance of CCDP, and experimental results and analysis demonstrate that it could be an effective model for early software defects detection.

1. Introduction

Software testing has become one of the most critical and costly phases in software development as size and complexity of software increases. Defect prediction based on static code metrics focuses on detecting defect-prone modules to help test teams accelerate searching for potential defects [1]. Existing studies [2–4] on this issue have proposed many effective prediction models, but they are usually confined to within-company defect prediction (WCDP), which attempts to train predictors based on historical data to detect future defects within the same source. However, a potential problem that cannot be avoided is that, in the early testing process, resources for practitioners are typically limited and may not be sufficient to build reliable predictors. Additionally, collecting historical training data is also time-consuming and difficult work [5]. On the other hand, many open source defect datasets can easily be found such as the PROMISE repository [6]. Can we use these abundant data to predict our specific software defects?

Cross-company defect prediction (CCDP) is an attractive solution for this issue. Different from previous methods, it tries to use cross-company (CC) datasets to build prediction models to detect defects in within-company (WC) data. Many studies have revealed that it can work as well as or even better than WCDP if CC data are carefully selected [7,8]. One of the most interesting methods is filter technology, for example, the Nearest Neighbor (NN) filter [8] returns the most similar samples from CC data as training samples. When the selected CC data have the same distribution as the WC data, it performs well (as shown in Fig. 1a).

However, since CC data are collected from different development environments or application fields, the labels of CC data may be in conflict with WC data even they are close in distance, which tends to generate false prediction results (Fig. 1b). More

* Corresponding author. Tel.: +86 023 65112784.
E-mail addresses: chenlincqu@cqu.edu.cn (L. Chen), fb@cqu.edu.cn (B. Fang), szw@cqu.edu.cn (Z. Shang), yytang@umac.mo (Y. Tang).

http://dx.doi.org/10.1016/j.infsof.2015.01.014
0950-5849/

specifically, if defect-free WC data have some contradictory defect-prone CC neighbors, they may be result in a high false alarm rate. On the other hand, if defect-prone WC data select negative defect-free CC data, it could lead to a low rate of recall.

If we have partial label information for WC data, these negative training samples can be reduced (Fig. 1c), so that misclassified instances can be rectified and the prediction result is moved into the right direction (Fig. 1d).

To overcome the effect of heterogeneity between CC and WC data, in this study, we propose a novel transfer learning method named Double Transfer Boosting (DTB) algorithm, which employs not only CC data but also a small amount of labeled WC data to improve the performance of CCDP.

The remainder of the paper is organized as follows: Section 2 gives a brief review of existing related work on CCDP. Section 3 describes the proposed DTB model. Section 4 reports experimental datasets and performance measures. Section 5 analyzes the experimental results in relation to the research questions. Section 6 discusses the potential threats to validity before we conclude this paper and suggest future work in Section 7.

2. Related work

As mentioned above, local historical training data from the same company is not always available or is difficult to collect in practice. To address this problem, researchers have turned to CCDP as an alternative solution in the last few years.

An early attempt at CCDP reported by Zimmermann et al. [9] was based on large-scale experiments for 12 real-world applications datasets. To investigate whether the CCDP model is able to work well, 622 cross-project predictions were examined. However, the test results were somewhat discouraging with a low success rate of 3.4%. Thus researchers warned that CCDP is still a serious problem. Meanwhile, an interesting phenomenon also has been found: Firefox is a strong defect predictor for Internet Explorer. However, this does not work as well as the opposite direction, even though Firefox and IE are similar applications.

Turhan et al. conducted CCDP experiments using 10 projects collected from two different companies including NASA and SOFTLAB [8]. They found that defect predictors built on all available CC data dramatically increased defect detection ability, but with unacceptably high false alarm rates. They explained that false alarms were raised by irrelevancies in CC data. Furthermore, they proposed a Nearest-Neighbor filter method (named NN filter) to select training data close to WC data. This method achieved better performance than using the raw CC data, but still is worse than WCDP.

Similar to Zimmermann et al.'s work, He et al. investigated CCDP with 34 datasets obtained from 10 open source projects [7]. Rather than using CC data directly, a schema of dataset selection was added before building the prediction models. To assess the performance of CCDP, they created a criterion (recall $\geqslant$ 70% and precision $\geqslant$ 50%) for judging successful predictors. Experimental results indicated that their method achieved promising results, i.e., that 18 out of 34 cases met the criteria, showing that WCDP does not always perform better than CCDP. They pointed out that one potential way to predict defects in projects without local data is to learn predictors from data of other projects.

Rahman et al. found a different way to evaluate the feasibility of CCDP. They only inspected a partial set of files to find out defects and introduced a new performance measure called area under the cost effectiveness curve (AUCEC) [10]. They also drew an optimistic conclusion from experimental results, i.e., that CCDP is no worse than WCDP in terms of AUCEC. However, the process metrics used in their defect model at file level are difficult to obtain in the open repository.

Transfer learning is another solution for CCDP. Ma et al. [11] considered that predictions should be related to the distributional characteristics of datasets. They proposed a novel CCDP model using data transfer method called Transfer Naive Bayes (TNB). As a type of instance-transfer[1] in transfer learning approaches [12], TNB was trained based on re-weighted CC data according to the distribution of WC data. Their results showed that CCDP model built on datasets collected from NASA could find defects in SOFTLAB effectively.

More recently, He et al. shared the same idea of CC data selection based on data similarity to improve the performance of CCDP [13]. Unlike the NN filter [8] picking out some individual instances, their model selected closest cross training datasets with similar distributions. The distance of distribution between CC data and WC data was measured according to the classification accuracy. Feature subset selection was also employed to remove unstable features. Empirical study on datasets including open-source and proprietary projects was conducted to prove the feasibility of CCDP. The results indicated that their data selection method could perform relatively better than the NN filter.

Zhang et al. built a universal defect prediction model for CCDP from diverse datasets [14]. To address variations of software metrics between different projects, the original metrics values were discretized by context-aware rank transformation according to similar degree of context factors, then, the universal model was built on the transformed metrics in the same scales. The results of rank transformation showed a performance comparable to log transformation. When compared with WCDP, their universal model improved predictive performance with higher Recall and AUC values, but it also suffered from a higher false positive rate.

The studies above all focus on using only CC data to build proper prediction models. Recently, Turhan et al. introduced a mixed model for CCDP for if limited amounts of WC data exist [15]. In their study, the within and cross data were combined together as training data, and the Naive Bayes classifier with NN filter was applied to build the prediction model. Their results implied that the mixed model could be comparable to WCDP. However, their study is a post-facto method because they randomly mixed different amounts of (from 10 to all of NN filtered samples) CC data with a smaller ratio of WC data to train the predictor and only reported the best performance. Furthermore, how to select the best subsets of CC data is still unanswered in their paper. Despite these issues, the paper suggests that even small amounts of WC data could improve the results of CCDP.

3. Methodology

In this study, the proposed Double Transferring Boosting (DTB) algorithm is built based on mixed training samples consisting of CC data and partial WC data. However, rather than using a simple combination of these data as the mixed model [15], we employ two-levels of transfer learning methods to "reshape" CC data to build a high-performance predictor. Its main steps are as follows: First, training datasets from other sources are preprocessed by the NN filter and data oversampling (SMOTE). Next, these preprocessed data are re-weighted by the first transfer method with data gravitation. Finally, limited amounts of labeled WC data (i.e., 10% of the total WC data) are mixed with re-weighted data to build the prediction model using the transfer boosting learning algorithm. Fig. 2 gives the framework of the proposed model.

[1] According to Ref. [12], main approaches to transfer learning includes instance-transfer, feature-transfer, parameters-transfer and relational-knowledge-transfer. The instance-transfer method refers to learn the knowledge from different source data by re-weighting labeled instances.

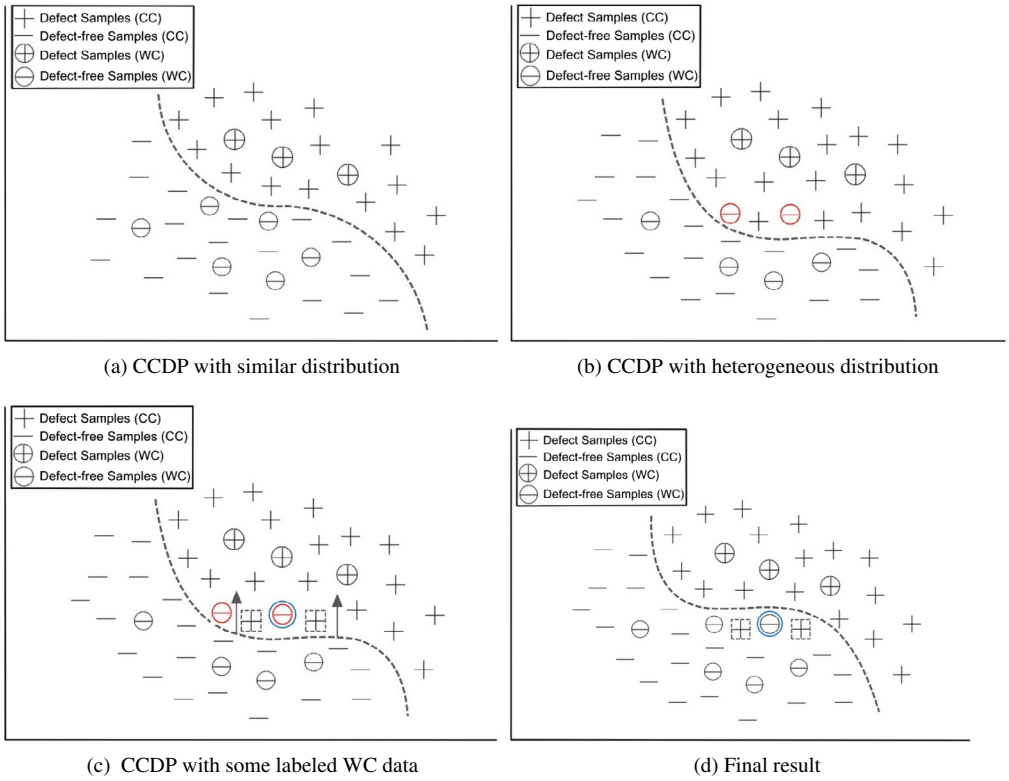

Fig. 1. Illustration of CCDP results with different distributions. Note that the line of discriminator is trained by labeled CC data: (a) All WC data are classified correctly. (b) The points with a red circle in WC data are wrongly labeled. (c) When partly labeled information about WC data (the point with double circles) is introduced, negative samples (points in dotted box) in CC data can be reduced. (d) The line of discriminator is moved into the correct position.

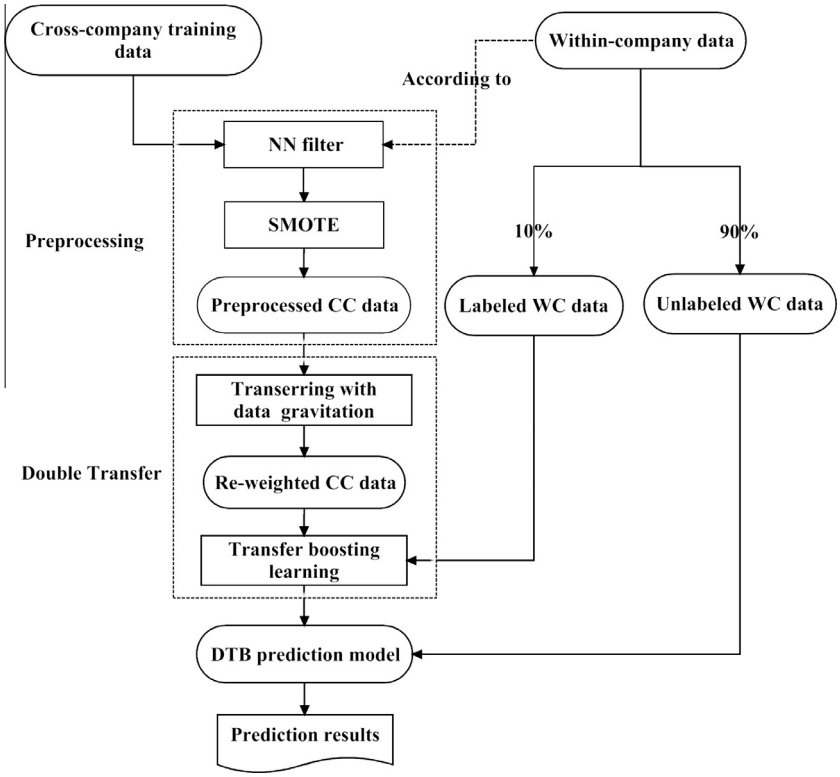

Fig. 2. Double Transfer Boosting (DTB) defects prediction model.

3.1. Data preprocessing

3.1.1. NN filter

Previous work [9,8] found that using raw CC data directly would increase false alarm rates due to irrelevant samples in CC data, thus data preprocessing is necessary before building the predictor in CCDP. The Nearest-Neighbor (NN) filter proposed by Turhan et al. [8] is applied in our prediction model to avoid these irrelevancies.

The NN filter assumes that the most valuable samples in CC data are those similar to WC data, so CC data can be filtered by selecting the most similar samples. The process of filtering is based on the *K*-Nearest Neighbors (KNN) algorithm [16], which has been widely adopted for classification and regression. In this case, it finds out the most similar K samples from CC data for every instance in WC data according to their distances (i.e., Euclidean distance).

After NN filtering, a new filtered CC dataset with a total of $K \times N$ instances will be generated, where N is the number of instances in WC data and K is the parameter of the KNN method. In this study, we choose K as 10 in our experiments. Note that duplicate samples may exist in this filtered dataset, as some instances in WC data may have some common neighbors in CC data. Thus, the final filtered CC training data can be formed by using only unique ones.

3.1.2. Data oversampling

Software defects have the typical characteristic of unbalanced distribution, because the number of defective modules is usually smaller than the number of non-defective ones. The skewness of distribution may cause difficulties for learning, as most traditional learning algorithms assume that each class in a dataset is equal, however, when they are trained by an unbalanced dataset in which the minority class is under-represented compared with the majority class, these classifiers tend to favor the majority class and have less ability to classify the minority class. On the other hand, recognition of the minority class, such as identifying software defect in this study, is more important in practice.

Many balance techniques such as data oversampling have been proven to reduce the effects of imbalance [17]. In this paper, we use the synthetic minority oversampling technique (SMOTE) to process the defects dataset to improve defects detection ability. SMOTE can generate new artificial minority class instances synthetically based on feature similarities [18] to provide more balanced class regions for the predictor.

3.2. Double Transfer Boosting model

A key challenge in CCDP is that distributions between CC data and WC data are usually variable [13], which may often give rise to significant misclassification, causing high levels of false alarms in particular.

The transfer learning method is an emerging and satisfactory solution for this issue. Different from traditional machine learning algorithms which assume that distributions of training and testing data must be from the same source, transfer learning relaxes this limitation to allow domains, tasks, or distributions used in training and testing to be different [12]. Regarding the issue of CCDP, domains and tasks are the same as for software prediction, but training and testing data are heterogeneous. In other words, our goal is to try to acquire the knowledge from one or more different data sources (CC data) to build the predictor, but apply it to local testing data (WC data).

In this section, we propose a new double transfer learning model that organically combines two-levels of data transfer methods to bridge the distribution gap between CC and WC data. The details are described as follows.

3.2.1. Data gravitation

The first level of our transfer model is aimed at changing the entire distribution of CC data by applying the data gravitation method [19]. The mechanics of this method rest on re-weighting instances according to a type of "force" called data gravitation, such as the gravity between two objects in physics.

In this paper, data gravity between the CC data and WC data is estimated based on the characteristics of feature attributes [11]. Suppose that an instance x_i can be described by k attributes $\{a_{i1}, ,a_{ik}\}$; for all WC data, we can define two vectors $MaxAttr = \{Max_1, \ldots, Max_k\}$ and $MinAttr = \{Min_1, \ldots, Min_k\}$ to represent its' distribution of attribute values, where the Max_i in $MaxAttr$ stores the maximum value of the ith attribute, while Min_i is the minimum value of the ith attribute.

Then, for each instance x_i in CC data, degrees of similarity S_i can be defined in Eq. (1).

$$S_i = \sum_{i=1}^{k} h(a_{ij}) \tag{1}$$

where a_{ij} is the jth attribute in x_i, and $h(a_{ij}) = 1$, if $Min_j \leqslant a_{ij} \leqslant Max_j$; otherwise, $h(a_{ij}) = 0$.

According to the formulation of data gravitation [19], weight w_i corresponding to x_i in CC data can be calculated by Eq. (2).

$$w_i = S_i/(k - S_i + 1)^2 \tag{2}$$

where k is the number of attributes. The weight w_i of CC data is proportional to similarity S_i, the largest weight will be assigned when $S_i = k$.

Each instance in CC data can be re-weighted by Eq. (2), their weights indicate how similar corresponding samples are to WC data so that the distribution of CC data can be reshaped to be close to WC data.

3.2.2. Transfer boosting learning

The second transfer level in our model is transfer boosting learning. As mentioned in Fig. 1, the performance of CCDP could be affect by some negative CC samples which are difficult to eliminate. Thus, the Transfer Boosting method named TrAdaBoost [20], introduced by Dai et al., is applied to refine CC data and build an effective defects classifier with a small portion of labeled WC data in our model.

The general idea of a boosting method is to train and combine a set of weak learners sequentially to build a stronger ensemble learner. AdaBoost is the most widely used boosting algorithm and was first introduced by Freund and Schapire [21]. As an iterative algorithm, AdaBoost increases the weights of misclassified training samples to highlight these errors in each training round, so that the probability of misclassification will be reduced in the next run.

TrAdaBoost extends the traditional AdaBoost method so that it can be applied to different data distributions. There are two different types labeled training data in TrAdaBoost: one part of labeled data has the same distribution as the test data, but it is possible that these data are inadequate to build a good classifier alone. Another part of the labeled data may have a different distribution from the test data, and those data are usually abundant. Obviously, in CCDP, these two types of training data can be considered as a small amount of WC data and CC data, respectively.

Formally, let $L_{wc} = \{x_1, \ldots, x_m\}$ be labeled WC datasets, and $L_{cc} = \{x_1, \ldots, x_n\}$ be CC datasets, where m and n are the sizes of L_{wc} and L_{cc}, thus the mixed training datasets $L = L_{wc} \cup L_{cc}$ $=\{x_1, ,x_m, x_{m+1}, x_{m+n}\}$, and $x_i \in L_{wc}$, if $1 \leqslant i \leqslant m$; otherwise, $x_i \in L_{cc}$. $y_i \in \{0, 1\}$ is the label of x_i. During the training process, these two datasets L_{wc} and L_{cc} will be treated separately by different weight update strategies.

The dataset L_{wc}, which has the same distribution and can be considered the more valuable training data, is trained by the traditional AdaBoost algorithm, that is, the misclassified instances in L_{wc} will be given larger weights to rectify the classifier. Specifically, at each iteration of epoch t, the weights $w_{wc}^t = \{w_1^t, \ldots, w_m^t\}$ for L_{wc} will be updated corresponding to the error function ε_t defined by

$$\varepsilon_t = \sum_{i=1}^{m} \frac{w_i^t |h_t(x_i) - y_i|}{\sum_{i=1}^{m} w_i^t} \tag{3}$$

where h_t is the base classifier trained by datasets L under the current distribution D_t. In our paper, Naive Bayes [22] is chosen to be the base learner due to its effectiveness in defects detection [23].

Set $\beta_t = \varepsilon_t/(1-\varepsilon_t)$, each weight w_i^{t+1} $(1 \leqslant i \leqslant m)$ in w_{wc}^{t+1} for next round can be updated as

$$w_i^{t+1} = w_i^t \beta_t^{|-h_t(x_i)-y_i|} \tag{4}$$

On the other hand, if an instance x_i in dataset L_{cc} which has diverse data distributions, is poorly predicted by h_t, it may belong to the negative ones that affect the accuracy of the classifier, thus the weight of this instance is decreased through multiplying the Hedge(β) [20] defined in the following:

$$\beta = 1/(1+\sqrt{2\ln n/T}) \tag{5}$$

where n is the size of L_{cc}, and T is the maximum number of iteration. Note that $\beta \in (0,1]$.

Thus, the weight w_i^{t+1} in $w_{cc}^{t+1} = \{w_1, \ldots, w_n\}$ for L_{cc} will be updated as

$$w_i^{t+1} = w_i^t \beta^{|h_t(x_i)-y_i|} \tag{6}$$

By applying TrAdaBoost in our CCDP model, after several fixed iterations T, negative training instances can be reduced, and the samples fitting WC data better are left to help the learning algorithm to train the classifier.

The final hypothesis of the transfer boosting algorithm can be expressed as follows:

$$H(x) = \begin{cases} 1, & \sum_{t=\lceil T/2 \rceil}^{T} \ln(1/\beta_t) h_t(x) \geqslant \frac{1}{2} \sum_{t=\lceil T/2 \rceil}^{T} \ln(1/\beta_t) \\ 0, & otherwise \end{cases} \tag{7}$$

In the end, Algorithm 1 presents the pseudo-code of the proposed DTB learning method for CCDP. As shown in Algorithm 1, both transfer methods are based on the re-weighted strategy. However, the process of data transferring is different: the data gravitation method, which can be regarded as *global transferring*, reshapes all CC data according to the distribution character of WC data, it also gives the more appropriate initial weights for TrAdaBoost. On the other hand, TrAdaBoost can be considered as *local transferring*, it reduces the part of CC data which are in conflict with WC data.

Therefore, the integration of superior features from two transferring methods in our algorithm can make them work in an integrated fashion to refine CC data from different levels, allowing construction of a more accurate defects prediction model.

Algorithm 1. Framework of Double Transfer Boosting (DTB) defect prediction model.

Input: Training dataset L including two labeled data: the CC data sets L_{cc} and a small ratio of WC data sets L_{wc};
Output: The lable of testing WC data L_t

1 Set m = size of L_{wc}, n = size of L_{cc}, k = number of attribute
2 Set every weight in $w_{wc} = (w_1, \ldots, w_m)$ as k for L_{wc}, and calculated weights $w_{cc} = (w_1, \ldots, w_n)$ for L_{cc} according to Eq. (2)
3 Combining weights w_{wc} and w_{cc} to set initial weights: $w^1 = (w_1^1, \ldots, w_m^1, w_{m+1}^1, \ldots, w_{m+n}^1)$
4 **for** $t = 1 \ldots T$ **do**
5 $t = 1 \ldots T$ Calculate distribution $D_t = w_i^t / (\sum_{i=1}^{n+m} w_i^t)$
6 Train the base classifier h_t on L using distribution D_t.
7 Update weights in w_{wc}^{t+1} and w_{cc}^{t+1} according to Eqs. (4) and (6), respectively.
8 Update weights $w^{t+1} = (w_{wc}^{t+1}, w_{cc}^{t+1})$ for the next round
9 **end**
10 Get the final classifier $H(x)$ which is assembled with the set of h_t $(1 \leqslant t \leqslant T)$ described in Eq. (7)
11 **Return** label L_t by using $H(x)$ to predict testing WC data.

4. Experiment setup

4.1. Datasets

Fifteen datasets are selected form the PROMISE data repository [6] in this paper. These data were collected by Jureczko and Madeyski [24] and Jureczko and Spinellis [25]. Each instance in dataset consists of two parts: 20 independent code attributes and a number attribute indicating how many defects are in this class. In our study, an instance is labeled as 1 if it has one or more defects, otherwise, instances with no defect are labeled as 0. Static code attributes are based mainly on object-oriented metrics including weighted methods per class, number of children, lines of code, etc. A more detailed description is provided in [24].

Our selection criteria for the simulation experimental datasets are as follows:

(1) The datasets are available from public sources so that model application and validation can be simpler.
(2) Each selected dataset has the same attributes, so it can be applied immediately and easily without losing any attribute information.
(3) For the multi-version datasets, we select only the first version, as our work focuses on the initial stages of software life.

Table 1 shows detailed information for the 15 selected datasets, including names, the number of instances, the number and percent of defects, and project descriptions. The whole datasets consist of 11 open-source projects, that were collected from different parts of open-source Apache projects; 3 academic projects, which were developed by some computer science students during one year; and a proprietary project that was built for customer solutions.

4.2. Performance measures

Considering the imbalanced characteristic of software defect datasets and following the recommendations in [26–28], evaluation of the defect predictors performance should be carried out using specific measures. First, a confusion matrix of predictors needs to be calculated first: True Positive (TP): the numbers of defective instances are correctly classified; True Negative (TN): the numbers of non-defective instances are correctly classified; False Positive (FP): the numbers of non-defective instances are misclassified; False Negative (FN): the numbers of defective instances are misclassified.

Table 1
The 15 data sets selected form the PROMISE repository [6], sorted in order of the name. (Defect%: the percentage of defective modules.)

Name	# Instances	# Defects	Defect%	Descriptions
ant	125	20	16	Open-source
arc	234	27	11.5	Academic
camel	339	13	3.8	Open-source
elearn	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
redactor	176	27	15.3	Academic
synapse	157	16	10.2	Open-source
system	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

And then, four frequently-used performance measures based on the confusion matrix are applied in our empirical study, which are described in detail below:

(1) Probability of detection (PD), also called recall, is the percentage of defects that are classified correctly within the defect class. PD = 1 is an ideal case, where all defects have been detected:

$$PD = TP/(TP + FN)$$

(2) Probability of false alarm (PF) is the percentage of non-defective instances are misclassified within the non-defect class. A higher PF means more time and cost will be wasted to determine the true defects:

$$PF = FP/(FP + TN)$$

(3) G-measure is a trade-off measure that balances the performance between PD and PF. A good predictor should have high accuracies in both classes, and thus lead to a high G-measure value:

$$G\text{-}measure = (2 \times PD \times (1 - PF))/(PD + (1 - PF))$$

(4) Matthews Correlation Coefficient (MCC) [29] is another balanced measure for binary classes, which takes into account all true and false positives and negatives. It returns a value ranging within the interval [−1, 1]. A perfect prediction yields the highest coefficient of +1, while an inverse prediction returns the lowest value of −1. MCC can be calculated by the following formulation:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

5. Research questions and results

5.1. Research questions

To demonstrate the utility of the proposed DTB model, we conducted some empirical studies to find answers to the following research questions:

5.1.1. RQ1: Can the DTB model perform better than other models in CCDP experiments?

The primary goal of this paper is to improve the performance of software defect prediction in the context of CCDP, thus, we need to verify whether the proposed DTB model can outperform other CCDP models. For this purpose we compared our model with four state-of-the-art methods used in CCDP. More details are provided below:

(1) Naive Bayes (NB) classifier is based on Bayes theorem [22]. Menzies et al. [1] reported that this so-called "Naive" method could perform well compared with more complex predictors in WCDP. NB has also been widely applied in prior work as a basic predictor [8,7] to investigate feasibility of CCDP.
(2) Nearest-Neighbor (NN) model is a filter technique [8]. It can exclude irrelevant instances by eliminating dissimilar samples. After NN filtering, the NB classifier is applied to build the prediction model in this paper.
(3) Nearest-Neighbor filter with WC data (NN + WC) is the mixed-project model proposed in [15], it directly mixed NN filtered CC data with a portion of WC data as training data. In our experiments, 10% of WC data are used for CCDP.
(4) Transfer Naive Bayes (TNB) is a transfer learning method for CCDP [11]. It re-weights CC data by the data gravitation method used in our first step of data transferring, but different from DTB, TNB uses weighted Naive Bayes as the prediction model.

Algorithm 2. The procedure of CCDP experiments

```
1   Dataset = {ant,arc,camel,elearn,jedit,log4j,lucene,poi,
    prop6,redact-or,synapse,system,tomcat,xalan,xerces}
2   for data ∈ Dataset  do
3     //Data preparing
4     WC_data = data
5     CC_data = Dataset-data
6     for t = 1...20  do
7         WC_trian = randomly select 10% of WC_data
8         WC_test = WC_data-WC_train
9        NN_data = Apply NN filter on CC_data
10        Mix_data = NN_data + WC_train
11        // Build prediction model
12        NB_model = Train NB learner with
          CC_data + log-operation
13        NN_model = Train NB learner with
          NN_data + log-operation
14        NN + WC_model = Train NB learner with
          Mix_data + log-operation
15         TNB_model = Train TNB learner with
           CC_data + discretization
16        DTB_model = >Train DTB learner with
          Mix_data + SMOTE
17        // Test each model on WC_test and record
          the performance
18        PD,PF,G-measure,MCC = Apply each model
          on WC_test data
19    end
20    Calculate the mean of performance for each model
      on this dataset
21  end
22  Return the overall average performance of each
    model for total 15 datasets
```

The pseudo-code for the procedure of comparison experiments is shown in Algorithm 2. It can be simply interpreted as follows:

To simulate CCDP, one dataset should be selected from the total dataset as WC data, and the rest are considered as CC data (Lines 1–5).

Table 2
Comparative results for RQ1: the mean value of PD and PF on the 15 datasets, where first line in the table shows the abbreviations of five predictors (the detail can be found in Section 5.1.1), and Total Avg. in last line is the overall average performance of each model for all 15 datasets. Relatively best performances in each row of the table are denoted in boldface.

No.	Dataname	DTB		TNB		NN		NN + WC		NB	
		PD	PF	PD	PF	PD	PF	PD	PF	PD	PF
1	ant	0.835	0.341	0.908	0.541	0.481	**0.302**	0.481	**0.302**	**1.000**	0.797
2	arc	0.625	**0.299**	0.726	0.386	0.795	0.640	**0.841**	0.658	0.640	0.568
3	camel	0.626	0.316	0.625	**0.293**	**0.767**	0.689	**0.767**	0.690	0.702	0.646
4	elearn	0.705	**0.196**	**1.000**	0.430	0.980	0.427	0.990	0.438	**1.000**	0.666
5	jedit	0.595	**0.169**	0.477	0.183	0.903	0.630	**0.909**	0.631	0.875	0.722
6	log4j	0.824	0.268	0.657	**0.154**	**0.937**	0.715	**0.937**	0.716	**0.937**	0.600
7	lucene	0.605	0.268	0.591	**0.234**	0.752	0.524	0.734	0.496	**0.775**	0.712
8	poi	0.698	0.373	0.411	**0.243**	**0.915**	0.703	0.914	0.701	0.895	0.744
9	prop-6	0.698	0.357	0.521	**0.347**	0.861	0.653	0.853	0.651	**0.894**	0.782
10	redactor	0.924	0.639	0.622	**0.524**	**1.000**	0.900	**1.000**	0.896	1.000	0.842
11	synapse	0.844	0.490	0.735	**0.425**	0.931	0.773	0.931	0.774	**0.938**	0.824
12	system	0.722	0.344	0.556	0.268	0.784	0.401	0.778	0.394	0.912	0.665
13	tomcat	0.716	**0.216**	0.918	0.593	0.695	0.376	0.718	0.360	**0.920**	0.647
14	xalan	0.739	0.377	0.606	**0.354**	0.942	0.662	**0.949**	0.666	0.944	0.701
15	xerces	0.373	0.293	0.313	**0.274**	**0.426**	0.660	0.424	0.635	0.415	0.651
	Total Avg.	0.702	**0.330**	0.644	0.350	0.811	0.604	0.815	0.601	**0.856**	0.704

Lines 7–10 are the data preparing stage, in which WC data will be randomly divided into two parts: one uses 10% of WC data as the labeled WC_train data, and the remainder are taken as the test data. The NN_data is the subset of CC data with NN filtering, and the Mix_data is the NN_data mixed with WC_train data.

Lines 11–16 are the training stage of prediction models. There are some corresponding data preprocesses for each predictor: the log-operation means that numeric values will be transformed by their logarithms operation, since prior research [1,30] have suggested that doing so will improve performance of the Naive Bayesian model. The discretization in Line 15 means all numeric features are discretized by the MDL-based discretization scheme [31], which is recommended in the TNB model [11]. SMOTE is the data balance processing for our model that was described in Section 3.1.2;

Lines 18 is the performance result of each model applied to the test data. Noted that, to keep the results fair, although the training sample data for each model are different,[2] the test data are the same ones selected from the 90% of WC data.

All of the above steps will be repeated 20 times to avoid sample bias; then, Line 20 calculates the mean values of performance for each model.

5.1.2. RQ2: Can the DTB model be comparable to or even better than WCDP?

The answer to this question will help us determine whether we should adopt the CCDP method in practice. In this study, two different situations of WCDP will be investigated and compared with the DTB model, thus we split RQ2 into two following sub-questions:

RQ 2.1: Can the DTB model perform better than WCDP with only limited training samples?

In the early phases of defects detection, abundant WC data are usually difficult to obtain for training predictors. However, a few papers report the performance of WCDP in this case. In this study, we want to clarify how well WCDP performs with little training data. Additionally, because our model also employs limited amount of WC data to improve prediction, a comparison between the DTB model and WCDP in this case can prove the effectiveness of our model.

RQ 2.2: Can the DTB model be comparable to WCDP with sufficient samples?

Previous studies [8,15] have suggested that it is difficult for the performance of CCDP to reach that of WCDP. If the DTB model can perform as well as WCDP, it will be worthwhile to apply in practice.

Many WCDP models have been investigated extensively, however, no clear consensus yet exists on which model generally performs best. Lessmann et al. note that the performance of most methods is not significantly different according to their large-scale empirical comparison, but they have confirmed that Random Forests (RF) is effective for defect prediction [32]. Meanwhile, Hall et al. find that simple modeling techniques such as Naive Bayes (NB) or Logistic Regression (LR) seem to perform relatively well [23]. Thus, to address RQ2, three modeling algorithms (Random Forests, Naive Bayes and Logistic Regression) are chosen to construct predictors in our WCDP experiments.

To simulate these two cases of WCDP, we selected 10% and 60% of WC data, respectively, in WCDP experiments. More specifically, 10% or 60% samples were randomly selected from each dataset to train the predictors, and the instances left would be used for testing. This process were repeated 20 times to avoid sample bias, and the final report of performance, including the mean values of G-measure and MCC, was recorded and compared with our model.

5.2. Results

5.2.1. Results for RQ1

The comparison results for CCDP models are shown in Tables 2–4 with different performance measures.

Table 2 presents experimental results of PD and PF on the 15 datasets, and Fig. 3 illustrates these results with scatter plots. Note that the defect model has more points distributed at bottom right if it has higher recall (PD) and fewer false alarms (PF).

The figure shows that the NB model directly applied to CC data takes up the top right of the areas in Fig. 3, indicating that it achieved the best PD (total average is 0.856), but the worst false alarm rates (total average is 0.704). These results are consistent with findings in Turhan et al.'s work [8] applied to other datasets. After filtering some irrelevant instances in CC data, the NN model

[2] Our DTB model and NN + WC are trained by CC data mixed with 10% of WC data, while TNB, NN and NN + WC are built using only CC data.

Table 3
Comparative results for RQ1: the mean value of G-measure and MCC for CCDP on the 15 datasets, where the first line in the table shows the abbreviations of five predictors, adn Total Avg. in the second line from the bottom is the average performance of each model for all 15 datasets. The last line is a summary of Wilcoxon rank sum test for DTB compared with other models. Relatively best performances in each row of the table are denoted in boldface.

No.	Dataname	G-measure					MCC				
		DTB	TNB	NN	NN + WC	NB	DTB	TNB	NN	NN + WC	NB
1	ant	**0.735**	0.610	0.569	0.569	0.337	**0.367**	0.276	0.124	0.123	0.196
2	arc	0.660	**0.665**	0.495	0.486	0.515	**0.234**	0.226	0.109	0.127	0.047
3	camel	0.648	**0.663**	0.442	0.441	0.471	0.129	**0.141**	0.037	0.037	0.019
4	elearn	**0.742**	0.726	0.723	0.717	0.501	**0.331**	0.314	0.309	0.301	0.195
5	jedit	**0.692**	0.603	0.525	0.524	0.422	**0.433**	0.324	0.300	0.302	0.177
6	log4j	**0.773**	0.740	0.437	0.436	0.561	0.491	**0.499**	0.239	0.236	0.316
7	lucene	0.658	**0.667**	0.583	0.598	0.420	0.341	**0.407**	0.245	0.251	0.077
8	poi	**0.656**	0.533	0.449	0.450	0.398	**0.324**	0.169	0.283	0.278	0.198
9	prop-6	**0.669**	0.580	0.495	0.495	0.350	**0.193**	0.109	0.139	0.137	0.084
10	redactor	0.505	**0.539**	0.181	0.188	0.273	**0.226**	0.071	0.127	0.128	0.169
11	synapse	0.632	**0.645**	0.365	0.364	0.297	**0.219**	0.204	0.119	0.120	0.099
12	system	0.676	0.632	0.679	**0.681**	0.490	**0.269**	0.217	**0.269**	0.268	0.174
13	tomcat	**0.748**	0.564	0.658	0.676	0.510	**0.325**	0.194	0.208	0.223	0.169
14	xalan	**0.676**	0.625	0.497	0.494	0.455	**0.262**	0.190	0.225	0.227	0.199
15	xerces	**0.488**	0.437	0.378	0.392	0.379	**0.085**	0.047	−0.228	−0.217	−0.225
	Total Avg.	**0.664**	0.615	0.498	0.501	0.425	**0.282**	0.226	0.167	0.169	0.126
		win/tie/lose	9/4/2	13/2/0	13/2/0	15/0/0	win/tie/lose	9/5/1	13/2/0	13/2/0	15/0/0

Table 4
Overview of comparison results for RQ1: comparing the performance of DTB with CCDP models across all the 15 datasets with Wilcoxon rank sum test. The last line is effect size with Hedges'g and the boldface in the table showing the significantly better results of the DTB model with p-value <0.05 or Hedges'g >0.5.

DTB vs.		TNB	NN	NN + WC	NB
G-measure	p-value	**0.044**	**0.001**	**0.003**	**1.61E−05**
	Hedges'g	**0.590**	**1.435**	**1.413**	**2.809**
MCC	p-value	0.106	**0.021**	**0.025**	**5.75E−04**
	Hedges'g	0.472	**0.909**	**0.902**	**1.311**

achieved a 10% reduction over NB in terms of PF on average. However, performance of the NN model with mixed 10% of WC data seems not to be changed significantly, leading to many overlap points in Fig. 3.

The group of transfer methods including DTB and TNB generated much lower PF (total average is 0.330 and 0.350, respectively) than the other three models, which also led to the result that most points are distributed at the bottom in Fig. 3. Furthermore, the DTB model also improved by approximately 6% of total average PD compared with TNB and yielded more bottom right scatter points.

A predictor with high-PD and high-PF would be useful in some critical industrial situations [33]. However, the false alarm rates higher than 60% (the overall average PF of NN is 60.4%, and NB is 70.4%) may be inefficient in early stages of software testing. Therefore, the DTB model with a much lower PF and an acceptable PD, could save time and cost to speed up bug fixing in this situation.

Table 3 gives the detailed results of two balanced performance measures including G-measure and MCC. Moreover, the Wilcoxon rank sum test at the 5% significance level is conducted to analyze whether there is a significant difference between DTB and other models. The summary statistical results are shown in the bottom line of table, which count how many datasets performed by the DTB model are significantly different (win or lose) or no different (tie) from other models.

Table 3 shows that, in the aspect of G-measure, the DTB model significantly outperforms the last three models including NN, NN + WC and NB, as it wins 15,13 and 13 datasets, respectively, and loses 0 datasets; compared with the TNB model, the DTB model wins 9 datasets, ties in 4 datasets, and only loses 2 datasets. Similar observations can be obtained for MCC, the only difference is that the number of lost datasets is decreased by one compared

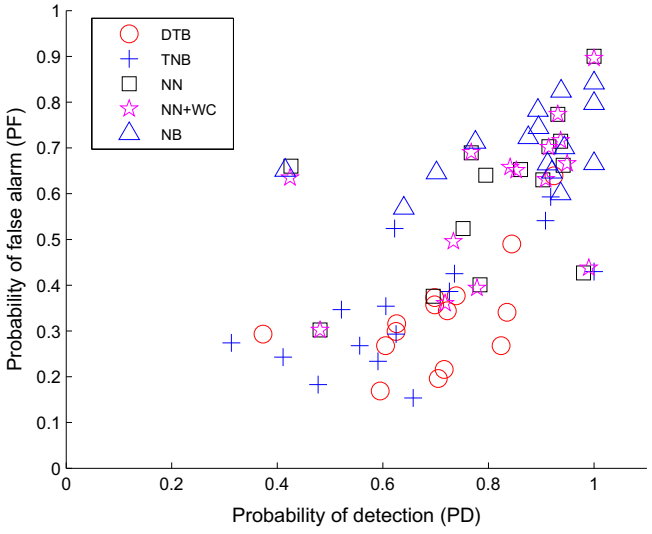

Fig. 3. Scatter plots of average PD and PF for five CCDP models on 15 datasets.

with the TNB model. Furthermore, overall performances of the DTB model across all 15 datasets are compared with that of other models based on the Wilcoxon rank sum test (p-value < 0.05). In addition, Hedges'g [34] is employed to demonstrate the effect size.

The overview of comparison results is presented in Table 4. The table shows that, in terms of G-measure, DTB performs statistically significantly better than all the other models (p-value are all less than 0.05). Moreover, the effect size of Hedges'g values for the last three models are greater than 1.0 (Hedges'g = 1.435, 1.413 and 2.809, respectively), which can be interpreted as a large improvement. As for the TNB model, the Hedges'g = 0.590 also can be considered a medium-size effect (i.e., greater than 0.50, but less than 1.0).

In regard to MCC measure, our model continues maintaining an advantage in comparison with NN, NN + WC and NB models by at least a medium effect(Hedges'g = 0.909, 0.902 and 1.311 respectively), and is comparable to the TNB model (p-value = 0.106).

5.2.2. Results for RQ2

In this section, we simulated two different situations of WCDP and examined three WCDP models, including Random Forests

Table 5
Comparative results for RQ2.1: mean values of G-measure and MCC for WCDP using limited training data (10% from total WC data) on the 15 datasets. Please note that the superscripts beside the numbers indicate the significantly different performance between DTB and WCDP models, and the last line is a summary statistical result for the DTB model.

Name	G-measure			MCC		
	RF	LR	NB	RF	LR	NB
ant	0.328 ↑	0.393 ↑	0.332 ↑	0.170 ↑	0.047 ↑	0.139 ↑
arc	0.237 ↑	0.329 ↑	0.336 ↑	0.138 →	0.015 ↑	0.111 ↑
camel	0.052 ↑	0.191 ↑	0.239 ↑	0.057 ↑	0.025 ↑	0.078 ↑
elearn	0.187 ↑	0.167 ↑	0.177 ↑	0.110 ↑	0.046 ↑	0.073 ↑
jedit	0.635 →	0.579 ↑	0.646 →	0.420 →	0.184 ↑	0.390 →
log4j	0.496 ↑	0.463 ↑	0.551 ↑	0.360 ↑	0.171 ↑	0.371 ↑
lucene	0.597 ↑	0.506 ↑	0.603 ↑	0.295 ↑	0.204 ↑	0.297 →
poi	0.598 ↑	0.525 ↑	0.552 ↑	0.275 →	0.199 ↑	0.221 ↑
prop-6	0.176 ↑	0.396 ↑	0.507 ↑	0.100 ↑	0.075 ↑	0.101 ↑
redactor	0.415 →	0.443 →	0.517 ↓	0.310 ↓	0.169 ↑	0.238 →
synapse	0.144 ↑	0.218 ↑	0.243 ↑	0.079 ↑	0.070 ↑	0.069 ↑
system	0.113 ↑	0.185 ↑	0.216 ↑	0.033 ↑	0.032 ↑	0.105 ↑
tomcat	0.268 ↑	0.443 ↑	0.667 ↑	0.203 ↑	0.144 ↑	0.272 ↑
xalan	0.279 ↑	0.455 ↑	0.586 ↑	0.189 ↑	0.127 ↑	0.219 ↑
xerces	0.560 ↓	0.549 ↓	0.491→	0.330 ↓	0.131 ↓	0.119 ↓
win/tie/lost	12/2/1	13/1/1	12/2/1	10/3/2	14/0/1	11/3/1

Table 6
Comparative results for RQ2.2: mean values of G-measure and MCC for WCDP using sufficient training data (60% from total WC data) on the 15 datasets. The superscripts indicate the significantly different performance between the DTB and WCDP models. The last line is a summary statistical result for DTB.

Name	G-measure			MCC		
	RF	LR	NB	RF	LR	NB
ant	0.364 ↑	0.533 ↑	0.588 ↑	0.201 ↑	0.195 ↑	0.281 ↑
arc	0.485 ↑	0.445 ↑	0.559 ↑	0.212 ↑	0.176 ↑	0.285 →
camel	0.058 ↑	0.160 ↑	0.618 →	0.045 ↑	0.049 ↑	0.282 ↓
elearn	0.260 ↑	0.266 ↑	0.535 →	0.319 →	0.158 ↑	0.294 →
jedit	0.719 ↓	0.704 →	0.634 ↑	0.634 ↓	0.453 →	0.409 →
log4j	0.650 ↑	0.592 ↑	0.679 ↑	0.481 →	0.316 ↑	0.515 →
lucene	0.672 →	0.656 →	0.636 →	0.365 →	0.323 →	0.316 →
poi	0.721 ↓	0.640 ↑	0.631 ↑	0.476 ↓	0.353 ↓	0.300 →
prop-6	0.380 ↑	0.487 ↑	0.588 ↑	0.239 →	0.153 ↑	0.164 ↑
redactor	0.618 ↓	0.555 ↓	0.668 ↓	0.519 ↓	0.308 ↓	0.336 ↓
synapse	0.166 ↑	0.468 ↑	0.687 ↓	0.141 ↑	0.249 →	0.295 ↓
system	0.257 ↑	0.597 ↑	0.670 →	0.281 →	0.355 ↓	0.402 ↓
tomcat	0.288 ↑	0.559 ↑	0.724 ↑	0.239 ↑	0.269 ↑	0.296 →
xalan	0.343 ↑	0.498 ↑	0.723 ↓	0.250 →	0.251 →	0.342 ↓
xerces	0.755 ↓	0.739 ↓	0.620 ↓	0.526 ↓	0.493 ↓	0.377 ↓
win/tie/lost	10/1/4	11/2/2	7/4/4	5/6/4	7/4/4	2/7/6

(RF), Logistic regression (LR) and Naive Bayes (NB). Experimental results including mean values of G-measure and MCC are shown in the Tables 5 and 6, respectively. Please note that Wilcoxon rank sum test at the 5% significance level is applied to stress the performance difference in the tables using superscripts: the significantly better (or worse) values of DTB compared with the WCDP model are denoted by '↑' (or '↓'), while no difference is indicated by '→'. The last line summarizes the comparison results by counting how many datasets processed by DTB are significantly different (win or lost) or no different (ties) from the corresponding WCDP models.

First, to answer RQ2.1, Table 5 reports the performance of WCDP when the training sample size was limited to 10% of total WC data. Although these selected classifiers had been proved effective in WCDP, the results seem to be not satisfactory. The G-measure values of most datasets rarely achieved 60% (the best predictor is the NB model, which has only three datasets greater than 60%). Moreover, DTB model generally performs better than all WCDP models, as it wins at least 12 datasets in terms of G-measure and shows a clear advantage in more than 10 datasets in terms of MCC.

Table 7
Comprehensive comparison results for RQ2: comparing the performance of DTB with three WCDP models under two different situations across all 15 datasets with Wilcoxon rank sum test and Hedges'g effect size. Boldface entries in the table indicate significantly better performance of the DTB model with p-value <0.05 or Hedges'g >0.5.

DTB vs.			RF	LR	NB
WCDP(10%)	G-measure	p-value	**2.33E−05**	**1.60E−05**	**2.23E−04**
		Hedges'g	**2.108**	**2.351**	**1.617**
	MCC	p-value	0.082	**8.13E−05**	**0.036**
		Hedges'g	**0.658**	**1.8623**	**0.855**
WCDP(60%)	G-measure	p-value	**0.010**	**0.003**	0.147
		Hedges'g	**1.238**	**1.207**	0.377
	MCC	p-value	0.561	0.351	0.213
		Hedges'g	−0.324	0.271	−0.456

To answer the RQ2.2, the same WCDP models were trained but using sufficient samples (i.e., 60% from total WC data). Test results are shown in Table 6.

Observing Table 6, the performance of each model is generally improved as the number of training samples increased. These results reflect that lack of sufficient WC data could be an obstacle to creating a good predictor in WCDP and would hinder effectiveness of WCDP in the early stages of software testing.

From the summary results in the last line of Table 6, we can infer the following:

The DTB model outperforms RF and LR on more than 10 datasets and loses less than 4 datasets based on G-measure. When turning to MCC, the advantage narrows, but our model is still comparable as it loses no more than 4 out of 15 datasets.

Regarding to the NB model, it presents the best outputs among all WCDP models, and our model shows a comparable G-measure performance as DTB wins 7 datasets, ties 4 datasets and loses 4 datasets. However, the results of MCC are in favor of the NB model, as the DTB model wins 2 datasets, ties 7 datasets and loses 6 datasets.

To obtain well-founded conclusions for RQ2, we carried out the statistical tests to compare overall performances of DTB with corresponding WCDP models across all 15 datasets. The results are presented in Table 7.

The overall statistical test results suggest the following.

Comparing with WCDP trained by limited samples (10% of WC data), DTB performs better than all WCDP models with a large improvement effect in terms of G-measure, and achieves significantly better MCC values than the LR and NB models, following by comparable MCC performance with RF. These results also support the observation from Table 5 that the DTB model was the winner of most datasets.

Comparing with WCDP trained by sufficient samples (60% of WC data), our model produces better performance than the RF and LR models as it improves G-measure with a large effect size (Hedges'g = 1.238 for RF and 1.207 for LR) and has comparable MCC results. Our model also matches the total average performance of NB according to G-measure and MCC (p-value = 0.147 and 0.213, respectively).

5.3. Answers to research questions

(1) **RQ1:** Can the DTB model perform better than the other models in CCDP experiment?
Answer: Compared with NN, NN + WC and NB models, the DTB model achieved much fewer false alarms and generated statistically significantly better balanced performance in terms of G-measure and MCC with at least medium effect; While comparing with the TNB model, our model performed significantly better on most experimental datasets (i.e. 9 out

of 15), achieving better overall G-measure and comparable MCC.
All evidence suggests that our model has better overall performance than other models in CCDP experiments.

(2) **RQ2:** Can the DTB model be comparable to or even better than WCDP?
Answer: The WCDP experiments in two different cases were conducted for the comparison with the DTB model. From the statistical test results, we conclude that our model performs significantly better than WCDP based on a limited number of data, and can be comparable to WCDP with abundant samples. In addition, considering that it will consume a great deal of resources and time to collect and analyze a large number of WC samples, we argue that the DTB model could be worth considering as a viable alternative to WCDP.

6. Validity threats

Potential threats to validity that can affect the results of the proposed approach are as follows:

6.1. Construct validity

In our study, the static code metrics we used to detect defects are mainly based on object-oriented metrics, which may be different from complexity metrics or size metrics in other papers. However, these metrics have been widely used in previous studies [35,36]. Furthermore, the researchers have investigated some primary software metrics and found that object-oriented metrics can achieve better performance [35]. Our experimental datasets are collected by Jureczko et al. [24,25], they admitted that there could be some mistakes in non-defective labels as not all the defects had been found. This may be a potential threat for defect models training and evaluation.

6.2. Internal validity

Sample selection bias is a common threat in data mining models. It peculiarly occurs in software defects detection, as skewness of distribution exists in software defect datasets [37]. Too many or few defective instances for training or testing may lead to significantly different results. In our study, we randomly select the training samples and repeat that many times to avoid any bias, and report average results to verify the performance of test models.

We compared our work with four state-of-the-art solutions of CCDP. To avoid the comparison bias, those methods were replicated in strict accordance with the authors instructions in related papers. These include different data processing such as a log-filter in NN and NB models and discretization in TNB. In addition, we have checked the experimental results of those approaches applied on the original datasets to ensure their models are able to produce correct outputs for our datasets.

6.3. External validity

External validity addresses concerns regarding whether a study can be generalized to other situations. In fact, it is difficult to draw general conclusions from empirical studies in software engineering as the background and context are variable in software projects [38]. In our work, we try to minimize this threat by testing defect models on different domains datasets including open source projects, academic projects and a proprietary project. However, we still cannot assume that results of our study can be extended in practice to applications due to a lack of real-word software datasets. We will conduct more research on more datasets as subjects in future works. Furthermore, this article provided a complete model description and specification to facilitate easy replication and testing on new datasets by other researchers.

7. Conclusions

CCDP is particularly useful for building an accurate defects prediction model when there is a lack of sufficient local data. However, experience has shown that CCDP is a challenge work, as not all CC data are suitable for building a better model than WC data. This often results in a high probability of false alarms. The variety of distributions between CC and WC data may be the main cause of this difficulty.

To address this issue we proposed a novel Double Transfer Boosting (DTB) algorithm to improve the performance of CCDP. CC data is reformed in two steps of data transferring: (1) the data gravitation re-weights all CC data to increase the distribution similarity with WC data and (2) the transfer boosting removes negative samples in CC data using a small amount of labeled WC samples.

We investigated four other CCDP models and three WCDP models to verify our model. Two balanced measures including G-measure and MCC were mainly applied to evaluate the performance of predictors. Note that different measures generated slightly different evaluation results. However, statistical analyses based on both measures tend to support the same conclusions: (1) DTB presents the best overall performance among all tested CCDP models and (2) DTB performs significantly better than WCDP models trained by limited samples, and also can be comparable to WCDP models with sufficient training data. All these results suggest that the proposed model could be an effective solution for early software testing.

In future work, we will investigate more appropriate methods for data transferring to apply in our model, along with further validation of this study by testing additional CCDP models on more software datasets, real-world datasets in particular.

Acknowledgements

This paper is supported by National Key Basic Research Program of China (973 program 2013CB329103 of 2013CB329100), the Program for Natural Science Foundation of China (No. 61173130, No. 61173129, No. 91420102, No. 61472053, No. 91118005) and Natural Science Foundation of Chongqing (No. CSTC2010BB2217). The authors also would like to thank the anonymous reviewers for their constructive comments.

References

[1] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.
[2] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Syst. Appl. 36 (4) (2009) 7346–7354.
[3] C. Catal, Software fault prediction: a literature review and current trends, Expert Syst. Appl. 38 (4) (2011) 4626–4636.
[4] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, A. Bener, Defect prediction from static code features: current results, limitations, new approaches, Autom. Softw. Eng. 17 (4) (2010) 375–407.
[5] B.A. Kitchenham, E. Mendes, G.H. Travassos, Cross versus within-company cost estimation studies: a systematic review, IEEE Trans. Softw. Eng. 33 (5) (2007) 316–329.
[6] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The Promise Repository of Empirical Software Engineering Data. <promisedata.googlecode.com>.
[7] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Autom. Softw. Eng. 19 (2) (2012) 167–199.
[8] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Softw. Eng. 14 (5) (2009) 540–578.
[9] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM, 2009, pp. 91–100.

[10] F. Rahman, D. Posnett, P. Devanbu, Recalling the imprecision of cross-project defect prediction, in: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, 2012, p. 61.
[11] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inform. Softw. Technol. 54 (3) (2012) 248–256.
[12] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[13] Z. He, F. Peters, T. Menzies, Y. Yang, Learning from open-source projects: an empirical study on defect prediction, in: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2013, IEEE, 2013, pp. 45–54.
[14] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, Towards building a universal defect prediction model, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 182–191.
[15] B. Turhan, A. Tosun Mısırlı, A. Bener, Empirical evaluation of the effects of mixed project data on learning defect predictors, Inform. Softw. Technol. 55 (6) (2013) 1101–1118.
[16] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory 13 (1) (1967) 21–27.
[17] D.J. Drown, T.M. Khoshgoftaar, N. Seliya, Evolutionary sampling and software quality modeling of high-assurance systems, IEEE Trans. Syst. Man Cybernet. Part A: Syst. Hum. 39 (5) (2009) 1097C–1107.
[18] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[19] L. Peng, B. Yang, Y. Chen, A. Abraham, Data gravitation based classification, Inform. Sci. 179 (6) (2009) 809–819.
[20] W. Dai, Q. Yang, G.-R. Xue, Y. Yu, Boosting for transfer learning, in: Proceedings of the 24th International Conference on Machine Learning, ACM, 2007, pp. 193–200.
[21] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: Computational Learning Theory, Springer, 1995, pp. 23–37.
[22] D.D. Lewis, Naive (Bayes) at forty: the independence assumption in information retrieval, in: Machine Learning: ECML-98, Springer, 1998, pp. 4–15.
[23] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (6) (2012) 1276–1304.
[24] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, 2010, pp. 9:1–9:10.
[25] M. Jureczko, D. Spinellis, Using object-oriented design metrics to predict software defects, Models Methods Syst. Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej (2010) 69–81.
[26] Y. Jiang, B. Cukic, Y. Ma, Techniques for evaluating fault prediction models, Empirical Softw. Eng. 13 (5) (2008) 561–595.
[27] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[28] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, Further thoughts on precision, in: 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), IEEE, 2011, pp. 129–133.
[29] B.W. Matthews, Comparison of the predicted and observed secondary structure of T4 phage lysozyme, Biochim. Biophys. Acta (BBA) – Protein Struct. 405 (2) (1975) 442C–451.
[30] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Trans. Softw. Eng. 37 (3) (2011) 356–370.
[31] U. Fayyad, K. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambéry, France, August 28–September 3, 1993, 1993, pp. 1022–1029.
[32] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496.
[33] T. Menzies, A. Dekhtyar, J. Distefano, J. Greenwald, Problems with precision: a response to comments on data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (9) (2007) 637.
[34] V.B. Kampenes, T. Dybå, J.E. Hannay, D.I. Sjøberg, A systematic review of effect size in software engineering experiments, Inform. Softw. Technol. 49 (11) (2007) 1073–1086.
[35] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault prediction metrics: a systematic literature review, Inform. Softw. Technol. 55 (8) (2013) 1397–1418.
[36] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Inform. Sci. 179 (8) (2009) 1040–1058.
[37] B. Turhan, On the dataset shift problem in software engineering prediction models, Empirical Softw. Eng. 17 (1–2) (2012) 62–74.
[38] V.R. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, IEEE Trans. Softw. Eng. 25 (4) (1999) 456–473.