Autom Softw Eng (2012) 19:167–199
DOI 10.1007/s10515-011-0090-3

An investigation on the feasibility of cross-project defect prediction

Zhimin He · Fengdi Shu · Ye Yang · Mingshu Li · Qing Wang

Received: 15 December 2010 / Accepted: 27 June 2011 / Published online: 13 July 2011

Abstract Software defect prediction helps to optimize testing resources allocation by identifying defect-prone modules prior to testing. Most existing models build their prediction capability based on a set of historical data, presumably from the same or similar project settings as those under prediction. However, such historical data is not always available in practice. One potential way of predicting defects in projects without historical data is to learn predictors from data of other projects. This paper investigates defect predictions in the cross-project context focusing on the selection of training data. We conduct three large-scale experiments on 34 data sets obtained from 10 open source projects. Major conclusions from our experiments include: (1) in the best cases, training data from other projects can provide better prediction results than training data from the same project; (2) the prediction results obtained using

Z. He (✉) · F. Shu · Y. Yang · M. Li · Q. Wang
Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing 100190, China
e-mail: hezhimin@itechs.iscas.ac.cn

F. Shu
e-mail: fdshu@itechs.iscas.ac.cn

Y. Yang
e-mail: ye@itechs.iscas.ac.cn

M. Li
e-mail: mingshu@iscas.ac.cn

Q. Wang
e-mail: wq@itechs.iscas.ac.cn

Z. He
Graduate University Chinese Academy of Sciences, Beijing 100190, China

M. Li
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

training data from other projects meet our criteria for acceptance on the average level, defects in 18 out of 34 cases were predicted at a *Recall* greater than *70%* and a *Precision* greater than *50%*; (3) results of cross-project defect predictions are related with the distributional characteristics of data sets which are valuable for training data selection. We further propose an approach to automatically select suitable training data for projects without historical data. Prediction results provided by the training data selected by using our approach are comparable with those provided by training data from the same project.

Keywords Defect prediction · Cross-project · Data characteristics · Machine learning · Training data

1 Introduction

Defect prediction focuses on detecting fault-prone modules precisely and helps to allocate limited resources in testing and maintenance (Menzies et al. 2007b, 2010; Lessmann et al. 2008; D'Ambros et al. 2010; Tosun et al. 2010; Carvalho et al. 2010; Weyuker and Ostrand 2008). Studies on this issue usually trained predictors from data of historical releases in the same project (i.e., defects distributional data and software metrics such as static code features, code change histories, and process metrics) and predicted defects in the upcoming releases, or reported the results of cross-validation on the same data set (Lessmann et al. 2008; Tosun et al. 2009; Weyuker et al. 2009; Weyuker and Ostrand 2008). To construct such a defect predictor we first need to collect a lot of historical data of the same project, that is, training data for the predictor. However, in practice, such kind of historical data is not always available, because either it does not yet exist or was not well collected (Zimmermann et al. 2009; Turhan et al. 2009; Jureczko and Madeyski 2010). That means defect prediction based on historical data of the same project is impossible in some cases. On the other hand, there are many public defect data sets available,[1] especially in the open source world. A potential way of predicting defects in projects without historical data is to make use of these public data sets as training data.

Cross-project defect prediction refers to predicting defects in a project using prediction models trained from historical data of other projects (Zimmermann et al. 2009; Watanabe et al. 2008). There are some studies focusing on this issue and their results show that cross-project defect prediction is still a serious challenge (Zimmermann et al. 2009; Turhan et al. 2009). For machine learning based predictions, the effect of a predictor depends on two factors: the training data and the learning algorithm. Consequently there are two potential ways for cross-project defect prediction:

(1) Finding the best suitable training data for the project to be predicted. Ideally we may find a training data presenting the same defect pattern with the target project, which will lead to acceptable prediction results.

[1] Public data sets for defect prediction are available at the PROMISE Repository (http://promisedata.org/repository), NASA MDP (http://mdp.ivv.nasa.gov/repository.html) etc.

(2) Using learning algorithms with high generalization ability to construct the defect predictor. This method assumes that there exists a general defect pattern between different data sets. If we can learn this pattern successfully, we can predict defects in other projects.

In this paper, we investigate cross-project defect prediction by focusing on the former way.[2] More specifically, we try to find empirical evidences to answer the following questions:

Q1: Can training data from other projects provide acceptable prediction results?

Before starting cross-project defect prediction, we have to confirm that it is possible to use training data from other projects to predict defects in the target project with acceptable performance. In other words, for a given project without historical data, we may find suitable training data which can be used to well predict its defects from other projects. Similar questions have been investigated by Zimmermann et al. (2009) and Turhan et al. (2009, 2010), and we extend their work to provide more empirical evidences.[3]

Q2: Does training data from the same project always lead to better prediction results than training data from other projects?

It is a common tendency to suppose that training data in the same project may leads to better prediction results than training data from other projects since different releases in a same project usually share similar contexts, e.g., the process, developers, and organization (Koru and Liu 2005; Wahyudin et al. 2008). However, there is little empirical evidence to support this assumption directly. It is one of our objectives to investigate this question in a more systematic and quantitative way.

Q3: Are characteristics of data sets valuable for selecting suitable training data for cross-project defect prediction from various available data sets?

If there are suitable cross-project training data for a target project, the following question is how we can get these suitable training data automatically. Characteristics of data sets are potential determinants of acceptable cross-project defect prediction (Zimmermann et al. 2009; Watanabe et al. 2008). We need to verify whether the suitable training data can be precisely identified by the characteristics of data sets.

Aiming at the above questions, we conducted 3 experiments on 34 public data sets obtained from 10 projects. All these data sets are available at the PROMISE Data Repository (Boetticher et al. 2007). We employed 5 machine learning algorithms to

[2] Previous studies of Menzies et al. (2008, 2010) show that defect predictors hit "performance ceiling", some inherently higher bounds of performance that cannot be break through by simply adopting different learning algorithms. Their observations decrease the hope of solving cross-project defect prediction problem by using learning algorithms with high generalization ability. Though our study mainly focuses on the first potential way, we partly consider the second way by training predictors based on 5 different learning algorithms (see Sect. 3.2).

[3] Section 2 provides more details and discussions of the relationship and differences between our study and some previous related studies.

construct prediction models by using the open source data mining tool WEKA.[4] Main contributions of this paper include:

(1) The verification of the usability of cross-project data for defect prediction. If selected carefully, training data from other projects may provide acceptable prediction results.
(2) The verification of a counter-intuitive conclusion that training data from other projects may provide better prediction results than those from the same project.
(3) The identification of distributional characteristics of a data set that are informative for selecting suitable training data.

The rest of this paper is structured as follows: Sect. 2 summarizes some related work; Sect. 3 describes the methodology of this study, including the data, learning algorithms, and the performance evaluation criteria; Sects. 4 to 6 describe the three experiments we conducted, respectively; Sect. 7 discusses threats to the validity of our results. Finally we conclude this study in Sect. 8.

2 Related work

This research has its roots in defect prediction and cross-project/cross-company defect prediction. The following sections discuss these related work, respectively.

2.1 Defect prediction

As one of the most critical quality assurance activities, software testing is time consuming and labor-intensive while resources available for testing are usually limited (Turhan et al. 2009; Ostrand et al. 2005). Empirical evidences show that the distribution of software defects is imbalanced, i.e., a small number of modules contain most of defects (Fenton and Ohlsson 2000; Weyuker et al. 2008, 2009; Li et al. 2010). Hence we need to allocate testing resources strategically to improve the efficiency of testing. Defect prediction has been proved to be effective for optimizing testing resource allocation, for it can identify modules that are more likely to contain defects prior to testing (Tosun et al. 2010).

In the past decade, various defect prediction models have been proposed and machine learning techniques have been more and more popular in constructing defect predictors (Menzies et al. 2007b; Ohlsson and Alberg 1996; Nagappan and Ball 2005; Nagappan et al. 2006; Ostrand et al. 2005; Weyuker et al. 2008; Catal and Diri 2009; Hassan and Holt 2005; Wahyudin et al. 2008; Carvalho et al. 2010; D'Ambros et al. 2010; Jureczko and Spinellis 2010). However, most prediction models reported are intra-project applicable, i.e., learning from data of historical releases and then applying to the upcoming release in the same project. The application field of these models is restricted for data of historical releases is unavailable sometimes.

This research aims to extend the application field of defect prediction. It is unlike most previous software defect prediction research for its focus on a cross-project context.

[4] Available at http://www.cs.waikato.ac.nz/ml/weka/.

2.2 Cross-project/cross-company defect prediction

The problem of predicting defects in a cross-project context drew the attention of many researchers in recent years. To the best of our knowledge, studies on cross-project defect prediction do not show a conclusive picture so far. However, there are considerable studies focusing on this issue.

Zimmermann et al. (2009) ran 622 cross-project defect predictions on 12 real world applications and found that only 21 predictions worked successfully (all *Precision*, *Recall*, and *Accuracy* are greater than 75%). That means cross-project defect prediction will fail in most cases if not selecting training data carefully. They also found that cross-project defect prediction is not symmetry. For example, data of *Firefox* can predict *Internet Explorer* defects well (*Precision* equal to 76.47% and *Recall* equal to 81.25%) but the opposite direction does not work (*Recall* equal to 4.12%). What's more, they addressed how different factors influence cross-project defect prediction. They argued that characteristics of data and process are crucial factors for the success of cross-project defect prediction rather than domain. Consequently they concluded that simply using historical data of projects in the same domain does not lead to good prediction results. Zimmermann et al. pointed out that cross-project defect prediction is a serious challenge and more attention should be paid to this problem.

Turhan et al. (2009, 2010) studied a similar problem—cross-company defect prediction, that is, using data from other companies to build defect predictors for local projects. They analyzed 10 projects which were treated as projects from 8 different companies (7 NASA projects from 7 different companies and 3 projects from a Turkish software company). In their experiments only static code features were used to build defect predictors. They concluded that cross-company data increase the probability of defect detection (*pd*) at the cost of increasing false positive rate (*pf*). Their experimental results also show that nearest neighbor filtering can help to reduce *pf* when using cross-company data. However, using historical data from the same company is still a better choice for building defect predictors. Turhan et al. further pointed out that defect predictors can be learned from small amount of local data (e.g., 100 examples). So they suggested a two-phase approach for companies without local historical data: use nearest neighbor filtered cross-company data to build defect predictors at first, and after collecting enough local data, switch to predictors trained from local data. The data filtering method introduced by Turhan et al. was then be applied to effort prediction (Kocaqueli et al. 2010). Kocaqueli et al. argued that, after data filtering, the imported data may provide comparable results than the local data.

Watanabe et al. (2008) tried to adapt fault prediction models for inter project prediction. They trained a prediction model from a Java project and applied it to a C++ project. A method called "metrics compensation" was introduced to adapt the prediction model. Their "metrics compensation" method is based on the observation that average values of each metric are different between different data sets. They assume that adjusting the average values of each metric in the training set and the test set to the same level can increase the performance of defect prediction. More specifically, if the defect prediction model was trained from historical data of project A and

then used for predicting defects in project B, the test data should be compensated as follows: (*Each compensated metric value of project B*) = (*Each metric value of project B*) × (*average value of that metric of project A*)/(*average value of that metric of project B*). Surprisingly, both recall and precision increase after model adapting. They argued that in the case of similar domain and similar size, it is possible to reuse prediction model between projects developed with different programming languages. Since what Watanabe et al. did is actually a kind of data transformation, it implied that data characteristics may be a key influencing factor for cross-project defect prediction. One drawback of Watanabe's work is that they only analyzed two projects in their case study, which weakens the generalization ability of their conclusions.

Jureczko and Madeyski (2010) tried to divide various projects into different groups by using clustering techniques. According to their definition, a group is a set of projects with similar characteristics, and a defect prediction model should work well for all projects that belong to a same group. Jureczko's research work point out a potential way for cross-project defect prediction. If we can identify such groups successfully, we can predict defects for project without historical data by reusing defect prediction models trained from other projects belonging to the same group.

Nagappan et al. (2006) studied the generalization ability of prediction models for post-release defects. After investigating five Microsoft systems, they found that no single set of metrics fits all projects and predictors are accurate only when they are learned from the same or similar projects. Menzies et al. (2007b) also found that the best static code attributes for defect prediction vary from data set to data set, implying different data characteristics between different data sets. This finding was further supported by Watanabe et al. (2008). They argued that there is significant difference between data sets, using a C4.5 decision tree, they successfully recognized which project a class belongs to with *Accuracy* greater than 90%. What's more, Wahyudin et al. (2008) argued that prediction models learned from one project are usually not applicable for other projects because of the unique context for each project. Different data characteristics, in addition to different contexts of projects (e.g., process, developers, programming language) make cross-project defect prediction a big challenge.

This research mainly extends the work of Zimmermann et al. (2009) and Turhan et al. (2009, 2010) and focus on the problem of cross-project defect prediction. It provides some further empirical evidences for the study of cross-project defect prediction. By conducting large-scale experiments on publically available data sets, this research overcome some disadvantages of previous research work, e.g., it is hard to repeat Zimmermann's experiments because the data they used are not accessible for public.

3 Methodology

3.1 Data

In our experiments, 34 releases of 10 open source projects are investigated, which make 34 different defect data sets. These data sets were collected by Jureczko and

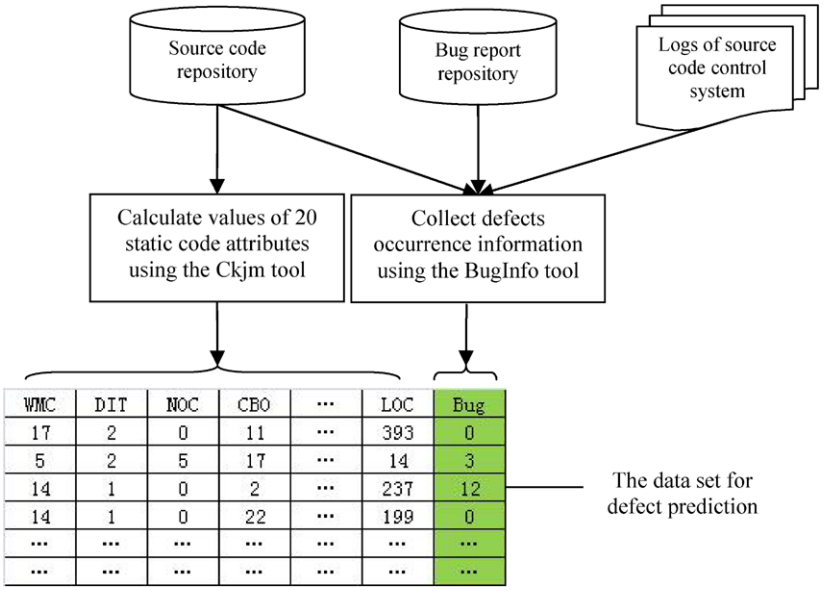

Fig. 1 The procedure of data collection

Madeyski (2010), Jureczko and Spinellis (2010) with the help of two tools: BugInfo and Ckjm. Each instance in a data set represents a class of the release and consists of two parts: instance features including 20 static code attributes and a labeled attribute *bug* indicating how many defects in this class. In our experiments, we consider a class as defect-free (DF) if the value of *bug* is equal to 0; otherwise, defect-prone (DP). The goal of defect predictions in our experiments is to identify defect-prone classes precisely for a given release.

The data collection procedure is illustrated in Fig. 1, it mainly includes two steps: calculating feature values for each instance and collecting information about defects occurrence (Jureczko and Madeyski 2010). The BugInfo tool analyzes logs of source code control systems and decides whether a commit is a bugfix by using regular expression matching technology. The 21 attributes (20 feature attributes and 1 labeled attribute) are listed in Table 1. More details of these attributes can be found in Jureczko and Madeyski (2010) and Jureczko and Spinellis (2010). The 34 data sets are listed in Table 2, where #class represents the number of classes in the release, #DP the number of defect-prone classes, and DP% the ratio of defect-prone classes.

3.2 Prediction models

We employ five machine learning algorithms to construct predictors. They are Naive Bayes (NB), J48 (a C4.5 decision tree), Support Vector Machine (SVM), Decision Table (DT), and Logistic. These 5 learners have been widely used in the context of defect prediction (Lessmann et al. 2008; Turhan et al. 2009; Menzies et al. 2007b; Watanabe et al. 2008).

Table 1 Descriptions of data attributes

Attribute	Description
WMC	Weighted Methods per Class
DIT	Depth of Inheritance Tree
NOC	Number of Children
CBO	Coupling Between Object classes
RFC	Response for a Class
LCOM	Lack of Cohesion in Methods
LCOM3	Lack of Cohesion in Methods, different from LCOM
NPM	Number of Public Methods
DAM	Data Access Metric
MOA	Measure of Aggregation
MFA	Measure of Function Abstraction
CAM	Cohesion among Methods of class
IC	Inheritance Coupling
CBM	Coupling Between Methods
AMC	Average Method Complexity
Ca	Afferent couplings
Ce	Efferent couplings
Max _CC	Maximum McCabe's Cyclomatic Complexity values of methods in the same class
Avg _CC	Mean McCabe's Cyclomatic Complexity values of methods in the same class
LOC	Lines of Code
Bug	Number of bugs detected in the class

The Naive Bayes learner is based on probability theory and assumes that features of the data set are independent of each other. Although the independence assumption is often violated in reality, the Naive Bayes learner has been proved to be effective for defect prediction (Menzies et al. 2007b). The C4.5 Decision Tree learner is an extension of the ID3 algorithm. It builds the tree structure from the training data by using the concept information entropy. Leaves in the tree structure represent classifications and branches represent judgment rules. More details about the C4.5 Decision Tree learner can be found in Quinlan (1993). The SVM learner is designed based on Vapnik-Chervonenkis dimension and the Structural Risk Minimization principle (Vapnik 1998). It performs classification by finding the optimal hyper-plane that maximally separates samples in two different classes. Comparative studies conducted by Lessmann et al. (2008) show that the SVM learner performs equally with the Naive Bayes learner in the context of defect prediction. The Decision Table learner is a rule based learner whose result can be easily understood. The Logistic is a classical regression method in statistics. More details of the comparison of defect prediction models based on different learners can be found in Lessmann et al. (2008).

According to the suggestion of Menzies et al. (2007b), we do not introduce any attribute selection techniques when constructing defect prediction models in our experiments.

Table 2 Summary of the 34 data sets

Release	# classes	# DP	DP (%)	Release	# classes	# DP	DP (%)
ant-1.3	125	20	0.160	poi-1.5	237	141	0.595
ant-1.4	178	40	0.225	poi-2.0	314	37	0.118
ant-1.5	293	32	0.109	poi-2.5	385	248	0.644
ant-1.6	351	92	0.262	poi-3.0	442	281	0.636
ant-1.7	745	166	0.223	synapse-1.0	157	16	0.102
camel-1.0	339	13	0.038	synapse-1.1	222	60	0.270
camel-1.2	608	216	0.355	synapse-1.2	256	86	0.336
camel-1.4	872	145	0.166	velocity-1.4	196	147	0.750
camel-1.6	965	188	0.195	velocity-1.5	214	142	0.664
ivy-1.1	111	63	0.568	velocity-1.6	229	78	0.341
ivy-1.4	241	16	0.066	xalan-2.4	723	110	0.152
ivy-2.0	352	40	0.114	xalan-2.5	803	387	0.482
jedit-3.2	272	90	0.331	xalan-2.6	885	411	0.464
jedit-4.0	306	75	0.245	xerces-init	162	77	0.475
lucene-2.0	195	91	0.467	xerces-1.2	440	71	0.161
lucene-2.2	247	144	0.583	xerces-1.3	453	69	0.152
lucene-2.4	340	203	0.597	xerces-1.4	588	437	0.743

3.3 Performance evaluation

In the context of defect prediction, defect-prone classes are usually considered as positive instances (Jiang et al. 2008). Consequently the 4 categories of defect prediction results are defined as below:

- TP (True Positive): defect-prone classes that are classified correctly;
- FN (False Negative): defect-prone classes that are wrongly classified to be defect-free;
- TN (True Negative): defect-free classes that are classified correctly;
- FP (False Positive): defect-free classes that are wrongly classified to be defect-prone.

The goal of defect predictions in our study is to precisely detect defect-prone classes, so we use two accuracy indicators: *Recall* and *Precision*, to assess the prediction results.

$$Recall = \frac{TP}{TP + FN}, \qquad Precision = \frac{TP}{TP + FP} \tag{1}$$

Values of *Recall* and *Precision* range from 0 to 1 and higher values indicate better prediction results. In the best case, both *Recall* and *Precision* are equal to 1, which means the predictor detects all defect-prone classes without FN or FP.

In practice, the values of *Recall* and *Precision* are usually mutually exclusive, i.e., high *Recall* value is often with low *Precision* value, and it is hard to achieve both high *Recall* and *Precision* at the same time. To assess the prediction results considering

both *Recall* and *Precision*, many alternative performance indicators such as *AUC*, *G-mean*, and *F-measure* are proposed (Jiang et al. 2008). In this paper, we employ the *F-measure* to evaluate the overall performance of a defect prediction model.

$$F = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{2}$$

The *F-measure* has been widely used in the field of Information Retrieval (IR) and Data Mining. It integrates *Recall* and *Precision* in a single indicator. Parameter β indicates the weight assigned to *Recall* and it assume any non-negative value. Higher β value means higher weight. If β is equal to 1, *Recall* and *Precision* are given the equal weight. Here in our experiments, we treat *Recall* and *Precision* equally, so we set β to 1.[5]

In this paper, we consider prediction results with *Recall* greater than 70% and *Precision* greater than 50% as acceptable results. There is no unified standard for judging whether results of a defect prediction is acceptable. Different studies may use different thresholds for judging acceptable (or successful) defect prediction, e.g., Zimmermann et al. (2009) defined good predictors as those with all *Recall*, *Precision*, and *Accuracy* greater than 75%. Overall, the thresholds of *Recall* and *Precision* are based on previous studies of some other researchers and our own research experience on defect prediction. Our reasons for using these thresholds are as below:

(1) Previous studies of Menzies et al. show that defect predictors learned from static code features hit a "performance ceiling", i.e., some upper bounds of performance (Menzies et al. 2008, 2010). They believe it is because the information provided by static code features is limited. It is hard to archive very high prediction performance in our experiments since we also use static code feature to detect defect-prone modules in this study. What's more, Menzies et al. (2007a, 2007b) reported their experiment results of defect predictions on NASA MDP data sets to be *mean (pd)* equal to 71% and *mean (pf)* equal to 25%, and they argued these prediction results are useful in practice. Considering these empirical evidences above, we think 70% is a reasonable threshold for *Recall*.
(2) As shown in Sect. 2.1, distribution of defect-prone instances and defect-free instances is often imbalanced, it is difficult to archive high *Precision* value in defect prediction (Menzies et al. 2007a; Zhang and Zhang 2007). So we set a lower threshold (50%) for *Precision*.

4 Experiment 1. Can cross-project data provide acceptable prediction results?

4.1 Design

Some previous studies show that cross-project defect prediction remains a challenging issue. We need to verify that whether prediction results provided by cross-project

[5] In practice, some users may consider high *Recall* is more important than high *Precision*, they should set β to a value greater than 1 in this case. Setting β to 1 is the default operation in the IR field.

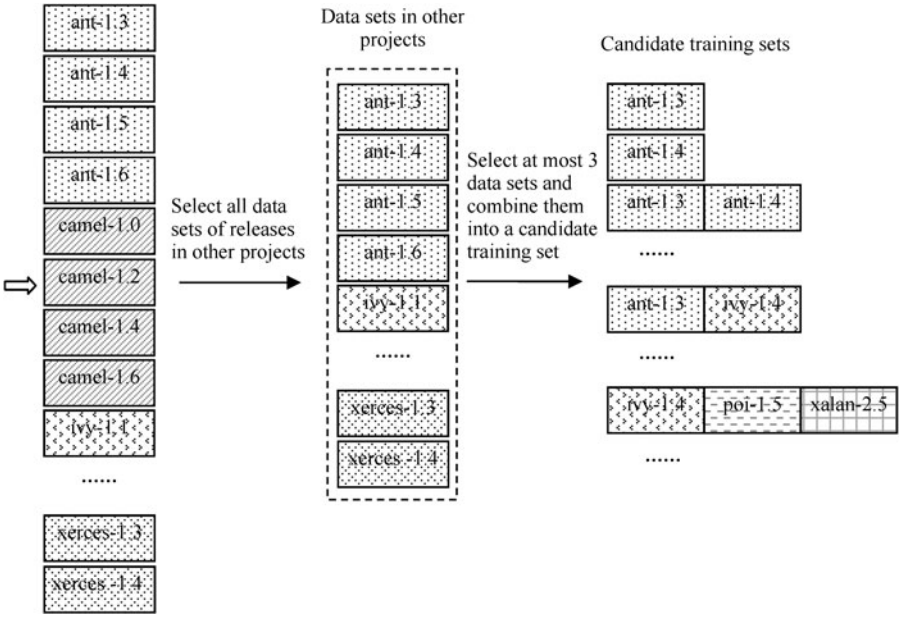

Fig. 2 The procedure of generating training sets in scenario A—an example of release *camel-1.2*

data are acceptable. To answer Q1 presented in Sect. 1, in Experiment 1, we designed a scenario of defect prediction in the cross-project context as follows:

Scenario A: selecting training data from data sets of other projects in an exhaustive way. In this scenario, we use combinations of data sets in other projects to train prediction models. We test all the combinations that consist of no more than 3 data sets in other projects; the procedure of generating training sets is illustrated in Fig. 2. The combination that can provide the best prediction result then is selected as the most suitable training set. For example, for the given data set of release *camel-1.2*, we use the combinations of data sets outside the *camel* project as training data (e.g., ⟨*ant-1.3*⟩, ⟨*ivy-1.4*⟩, ⟨*ant-1.3, ivy-1.4*⟩, ..., ⟨*ivy-1.4, poi-1.5, xalan-2.5*⟩, ⟨*poi-2.5, ant-1.6, jedit-4.0*⟩, ...). If the use of combination ⟨*ant-1.3, ivy-1.4*⟩ as the training data yields the best prediction result for release *camel-1.2*, we say ⟨*ant-1.3, ivy-1.4*⟩ is the most suitable training data for data set *camel-1.2* in the cross-project context. We must state that in this paper the comparisons of prediction results are based on the *F-measure*, so the most suitable training data in Experiment 1 is the training data that can provide prediction results with the highest F value.

The reason why we only test combinations consisting of no more than 3 data sets is the huge number of candidates. For the given data set of release *camel-1.2*, we select training data from 30 data sets in other projects (see Table 1). So the number of combination consisting of no more than N data sets can be calculated as:

$$M(N) = \sum_{i=1}^{N} C_{30}^{i} = \sum_{i=1}^{N} \frac{30!}{i! * (30-i)!} \tag{3}$$

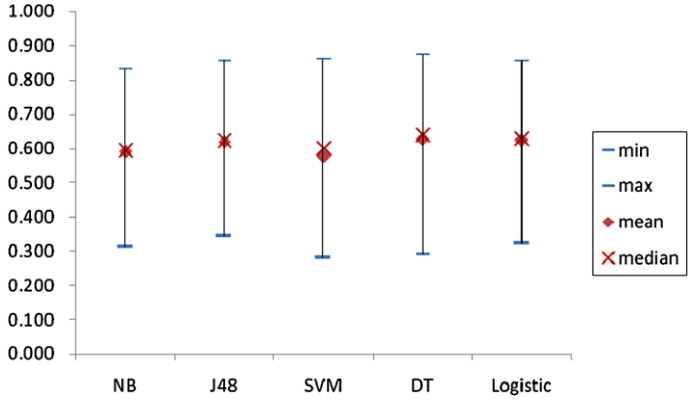

Fig. 3 Comparison of F values obtained from the most suitable training data

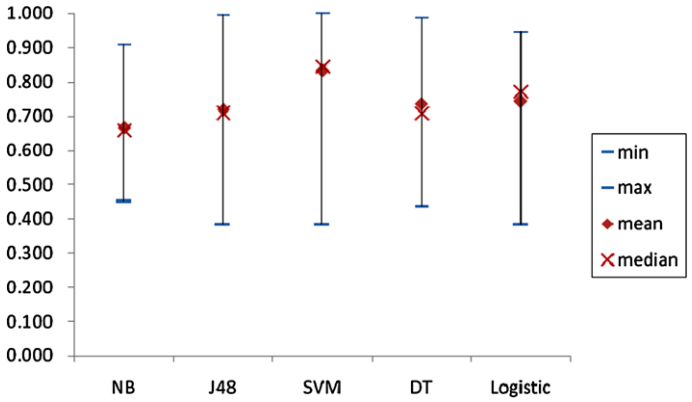

Fig. 4 Comparison of *Recall* obtained from the most suitable training data

Since $M(1) = 30$, $M(2) = 465$, $M(3) = 4525$, $M(4) = 31930$, $M(5) = 174436$ ... and our computing power is limited, we only test the combinations consisting of no more than 3 data sets.

4.2 Results and discussions

To assess the prediction results for all 34 data sets, we calculate the mean values of the F, *Recall*, and *Precision* of prediction results provided by the most suitable training data. Higher mean F value indicates better prediction results. The results are illustrated in Figs. 3, 4, 5 and Table 4.

Among all the predictors based on the 5 learning algorithms we studied, predictors based on *DT* learning algorithm provide the best prediction results, with *mean (F)* equal to *0.627, mean (Recall)* equal to *0.735* and *mean (Precision)* equal to *0.560.* Detailed prediction results of the *DT* predictors are illustrated in Table 3. Defects in 18 releases can be well predicted by cross-project training data with *Recall* greater than *0.7* and *Precision* greater than *0.5*, meaning more than half of the releases can

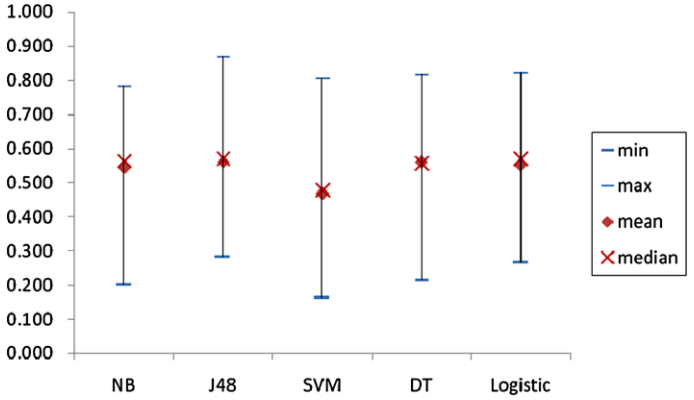

Fig. 5 Comparison of *Precision* obtained from the most suitable training data

find their suitable training data from other projects. Since we only tried training data consisting of no more than 3 data sets so far, the ratio of well predicted releases may increase if we increase the number of data sets, e.g., training sets consisting of 4, 5, or more data sets. As we can see from Figs. 3 to 5, predictors based on *J48*, *DT*, and *Logistic* algorithm may provide acceptable prediction results in an average level. However, the *min (F)* values of predictions in scenario A are still low, which means acceptable prediction results are unavailable for some data sets in this scenario. Another observation is that predictors based on *SVM* algorithm yield the highest *mean (Recall)* value while the lowest *mean (Precision)* value. Intuitively, it means that predictors based on *SVM* algorithm trend to detect more defect-prone modules with more FP defined in Sect. 3.3. Users who consider *Recall* more important than *Precision* may consider prediction results produced by *SVM* based predictors better than those produced by other predictors.

Based on the findings above, we answer question Q1:

A1: In the best cases, prediction results provided by training data from other projects are acceptable on the average level. For the 34 data sets we investigated, more than half of them can be predicted with *Recall* greater than *0.7* and *Precision* greater than *0.5*.

5 Experiment 2. Does training data from the same project always provide better prediction results than training data from other projects?

5.1 Design

To answer Q2 presented in Sect. 1, we need examine the differences between prediction results provided by the cross-project training data (i.e., scenario A in Experiment 1) and those provided by training data from the same project. Experiment 2 is designed to consist of following 2 scenarios of defect prediction which predict defects in the intra-release and the intra-project context, respectively:

Table 3 The most suitable training data and prediction result for each data set in scenario A when using the *DT* learning algorithm

Release	The most suitable training set	Recall	Precision	*F*
ant-1.3	jedit-3.2 + jedit-4.0 + synapse-1.0[a]	0.700	0.583	0.636
ant-1.4	poi-2.0 + velocity-1.5 + velocity-1.6	0.575	0.434	0.495
ant-1.5	lucene-2.0 + xalan-2.4 + xalan-2.5	0.563	0.486	0.522
ant-1.6	camel-1.6 + jedit-3.2 + xalan-2.5	0.707	0.650	0.677
ant-1.7	camel-1.0 + lucene-2.0 + synapse-1.2	0.705	0.544	0.614
camel-1.0	ivy-2.0 + lucene-2.2 + poi-1.5	0.462	0.214	0.293
camel-1.2	lucene-2.2 + poi-3.0 + xerces-1.4	0.935	0.391	0.552
camel-1.4	lucene-2.2 + synapse-1.0	0.552	0.349	0.428
camel-1.6	ant-1.5 + ivy-1.1 + lucene-2.4	0.521	0.334	0.407
ivy-1.1	ant-1.4 + lucene-2.2 + velocity-1.5	0.841	0.757	0.797
ivy-1.4	ant-1.4 + camel-1.0 + xerces-init	0.438	0.368	0.400
ivy-2.0	ant-1.5 + ant-1.7 + lucene-2.2	0.700	0.452	0.549
jedit-3.2	ivy-2.0 + lucene-2.2 + synapse-1.2	0.711	0.696	0.703
jedit-4.0	lucene-2.0 + xerces-1.2 + xerces-init	0.587	0.629	0.607
lucene-2.0	poi-3.0	0.780	0.696	0.736
lucene-2.2	jedit-3.2 + velocity-1.4 + xerces-1.4	0.986	0.612	0.755
lucene-2.4	poi-3.0 + xerces-1.4 + xerces-init	0.951	0.643	0.767
poi-1.5	jedit-3.2 + xalan-2.6 + xerces-1.4	0.908	0.719	0.803
poi-2.0	camel-1.0 + velocity-1.5 + xalan-2.6	0.514	0.422	0.463
poi-2.5	ivy-1.4 + xalan-2.5 + xerces-1.4	0.903	0.818	0.858
poi-3.0	ivy-1.1 + lucene-2.4 + xalan-2.5	0.879	0.805	0.840
synapse-1.0	ant-1.6 + ant-1.7 + poi-2.0	0.500	0.533	0.516
synapse-1.1	ant-1.3 + jedit-3.2 + lucene-2.4	0.617	0.561	0.587
synapse-1.2	ivy-1.1 + jedit-3.2 + poi-3.0	0.698	0.625	0.659
velocity-1.4	ant-1.4 + lucene-2.2 + xerces-1.4	0.966	0.798	0.874
velocity-1.5	xerces-1.2 + xerces-1.4 + xerces-init	0.944	0.720	0.817
velocity-1.6	ivy-1.1 + lucene-2.0 + poi-2.5	0.795	0.544	0.646
xalan-2.4	ant-1.6 + camel-1.2 + camel-1.4	0.609	0.416	0.494
xalan-2.5	lucene-2.2 + poi-1.5 + velocity-1.5	0.835	0.555	0.667
xalan-2.6	lucene-2.2 + poi-3.0 + velocity-1.5	0.864	0.597	0.706
xerces-init	ivy-1.1 + poi-2.5 + velocity-1.4	0.883	0.535	0.667
xerces-1.2	lucene-2.0 + velocity-1.4 + velocity-1.5	0.859	0.242	0.378
xerces-1.3	jedit-3.2 + synapse-1.1 + velocity-1.6	0.565	0.513	0.538
xerces-1.4	lucene-2.4 + velocity-1.4	0.947	0.807	0.872
Average	–	0.735	0.560	0.627

[a]Combine historical data of release *jedit-3.2*, *jedit-4.0*, and *synapse-1.0* into a single training set

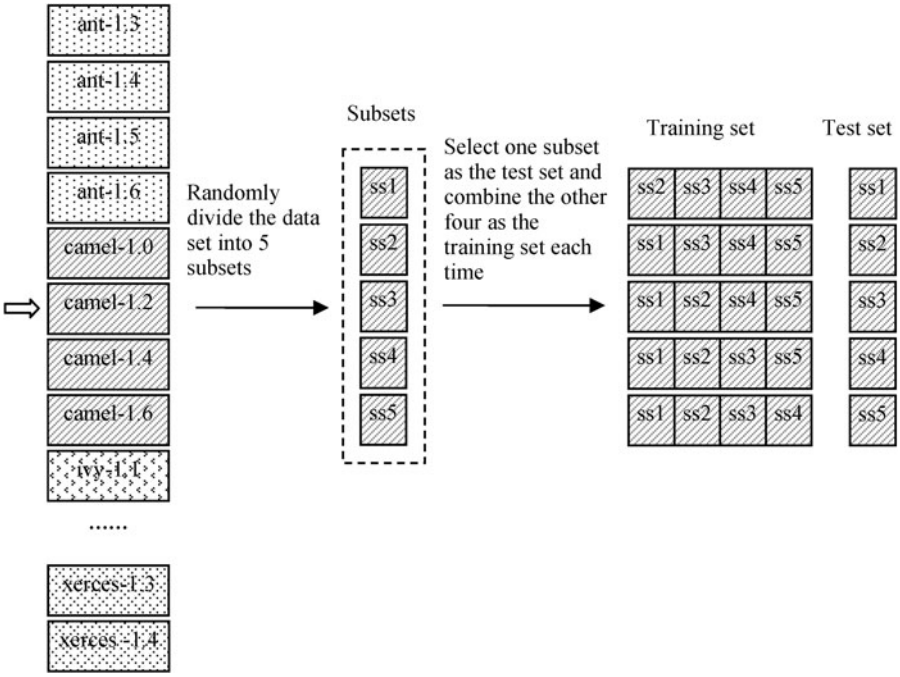

Fig. 6 The procedure of 5-folder cross validation in scenario B—an example of release *camel-1.2*

Scenario B: 5-folder cross-validation. In this scenario, each data set is divided into 5 subsets randomly. For each validation, we select one subset as the test set and combine the other four subsets as the training set (see Fig. 6). Cross-validation is a commonly used approach for assessing the performance of prediction models in the data mining field, and it has been employed in many previous studies of defect prediction (Lessmann et al. 2008). Scenario B aims to simulate defect prediction in an intra-release context, i.e., training prediction model from part of historical data in the same release and then apply it to the rest. Expectantly, defect predictions in this scenario will lead to good prediction results, for contexts of training data and test data are almost the same.

Scenario C: selecting training data from data sets in the same project. In this scenario, we try all combinations of data sets in the same project excluding the test set (see Fig. 7). As with scenario A, the combination providing the best prediction results (i.e. prediction results with the highest F value) is selected as the most suitable training data. For example, for the given data set of release *camel-1.2*, we test all combinations of other data sets in the *camel* project: ⟨*camel-1.0*⟩, ⟨*camel-1.4*⟩, ⟨*camel-1.6*⟩, ⟨*camel-1.0, camel-1.4*⟩, ⟨*camel-1.0, camel-1.6*⟩, ⟨*camel-1.4, camel-1.6*⟩, ⟨*camel-1.0, camel-1.4, camel-1.6*⟩. We must point out that in scenario C the orders of releases in the same project are not considered. For example, data of release *camel-1.6* is used to predict defects in release *camel-1.2*. The orders of releases are discarded for the purpose of enriching training sets. If considering release order strictly, some releases will get less or even no training sets from the same project,

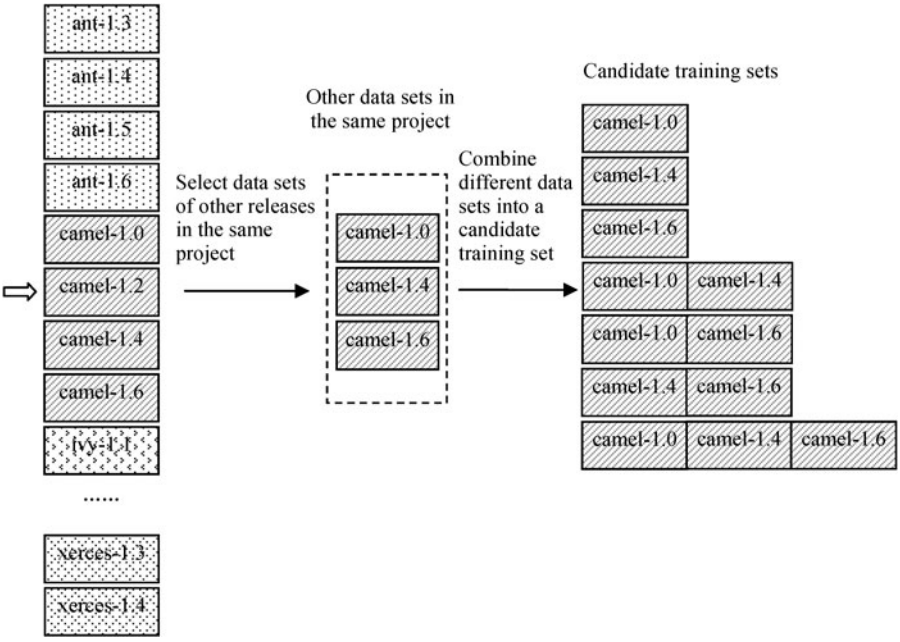

Fig. 7 The procedure of generating training sets in scenario C—an example of release *camel-1.2*

e.g., the first release of each projects. And it makes comparisons of prediction results between different scenarios very hard. What's more, the temporal characteristics of defects are usually ignored in the defect prediction literature, both in the intra-project context and the cross-project/cross-company context (Lessmann et al. 2008; Menzies et al. 2007b, 2010; Turhan et al. 2010). So, to simplify the study, we do not see release order as a constraint for selecting training sets in scenario C.

In Experiment 2, we investigate whether training data from the same project often provide better prediction results than those from other projects, by comparing the prediction results of scenario B and C with those of scenario A in Experiment 1.

5.2 Results

Results of scenario B are illustrated in Figs. 8 to 10 and Table 4. Similar to scenario A, we calculate the mean values of the F, *Recall*, and *Precision* of prediction results. Predictors based on *J48* can provide the best prediction results with *mean (F)* equal to *0.512*, *mean (Recall)* equal to *0.486* and *mean (Precision)* equal to *0.558*. Overall, predictions results given by all 5 learning algorithms do not satisfy our criteria for acceptance (see Sect. 3.3) since *mean (Recall)* values are low.

An interesting observation is that the *min (Recall)* and *min (Precision)* is equal to 0 for predictors based on *J48*, *SVM*, and *DT*. That means the prediction results are totally meaningless for some predictions. Our explanation is that the distributions of the defect-prone classes and defect-free classes are seriously unbalanced in some releases. For example, only 3.8% classes are defect-prone in release *camel-1.0*. To

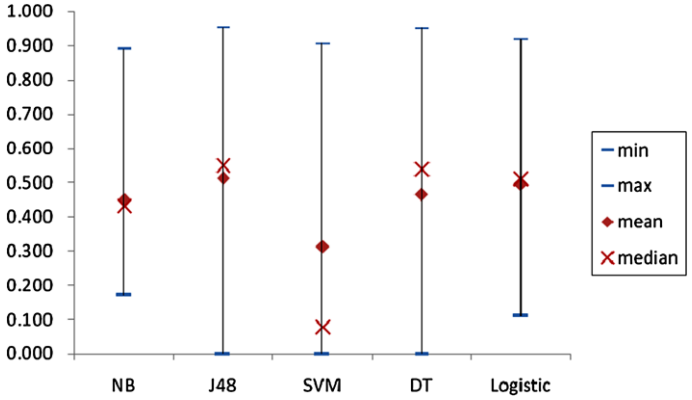

Fig. 8 Comparison of F values of the 5-folder cross-validation in scenario B

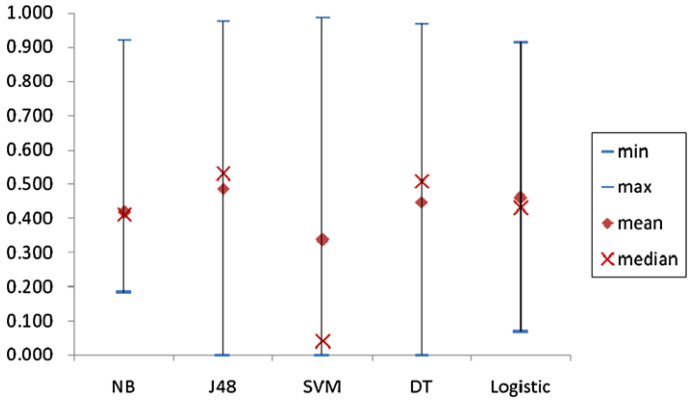

Fig. 9 Comparison of *Recall* values of the 5-folder cross-validation in scenario B

precisely learn patterns from this kind of imbalanced data is extremely hard (Khoshgoftaar et al. 2010).

Figures 11 to 13 show the prediction results in scenario C. In this scenario, the best prediction results are provided by predictor based on *J48*, with *mean (F)* equal to *0.496*, *mean (Recall)* equal to *0.585* and *mean (Precision)* equal to *0.512*.

Prediction results of both scenario B and scenario C do not satisfy our criteria on the average level, which indicates that training data from the same project or even the same release do not always lead to acceptable prediction results. Comparisons of prediction results in scenarios A, B and C are illustrated in Table 4.

Our major observations from Table 4 include that:

(1) Predictions in scenario A get better results than those in scenarios B and C. Particularly, *mean (F)* values in scenario A are much better, ranging from 26.67% to 72.24% higher than scenario C. The *mean (Recall)* values also are better while the *mean (Precision)* values are comparable.

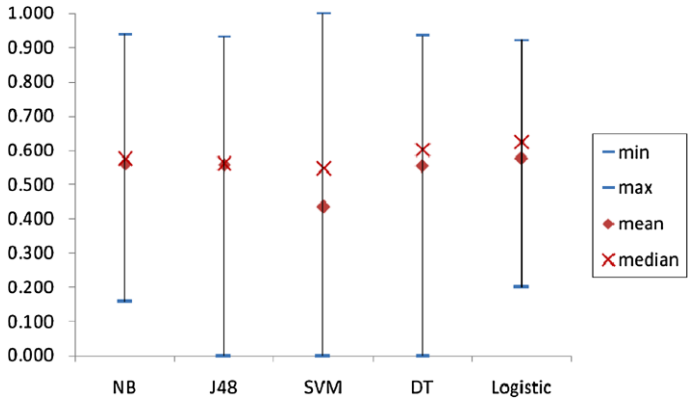

Fig. 10 Comparison of *Precision* values of the 5-folder cross-validation in scenario B

Table 4 Comparisons of predictions results in scenarios A, B and C

Scenario	Indicators	NB	J48	SVM	DT	Logistic
A	Mean (*F*)	0.591	0.620	0.577	0.627	0.625
	Mean (Recall)	0.668	0.718	0.833	0.735	0.743
	Mean (Precision)	0.545	0.562	0.469	0.560	0.553
B	Mean (*F*)	0.448	0.512	0.313	0.465	0.497
	Mean (Recall)	0.419	0.486	0.338	0.447	0.461
	Mean (Precision)	0.562	0.558	0.434	0.554	0.578
C	Mean (*F*)	0.440	0.496	0.335	0.495	0.470
	Mean (Recall)	0.444	0.585	0.456	0.622	0.530
	Mean (Precision)	0.548	0.512	0.390	0.502	0.511
Enhancement of indicators						
Scenario A vs. scenario B	Mean (*F*)	31.92%	21.09%	84.34%	34.84%	25.75%
	Mean (Recall)	59.43%	47.74%	46.45%	64.43%	61.17%
	Mean (Precision)	−3.02%	0.72%	8.06%	1.08%	−4.33%
Scenario A vs. scenario C	Mean (*F*)	34.32%	25.00%	72.24%	26.67%	32.98%
	Mean (Recall)	50.45%	22.74%	82.68%	18.17%	40.19%
	Mean (Precision)	−0.55%	9.77%	20.26%	11.55%	8.22%

(2) The scopes of the *mean (F)* values in scenario A are smaller than those in scenario B and C, indicating prediction results in scenario A are more stable.

In addition to the high level comparisons of mean values of performance indicators, we further conduct project-wise comparisons between predictions of scenarios A, B, and C. Table 5 illustrates whether performance indicators in scenario A increase when compared with those in scenario B and C. More specifically, we used

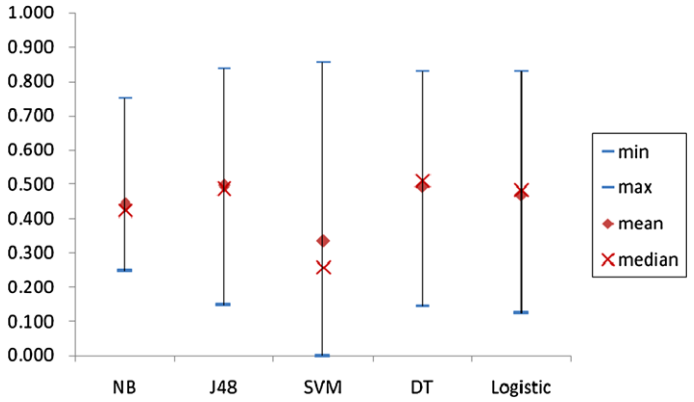

Fig. 11 Comparison of F values obtained from the most suitable training data in scenario C

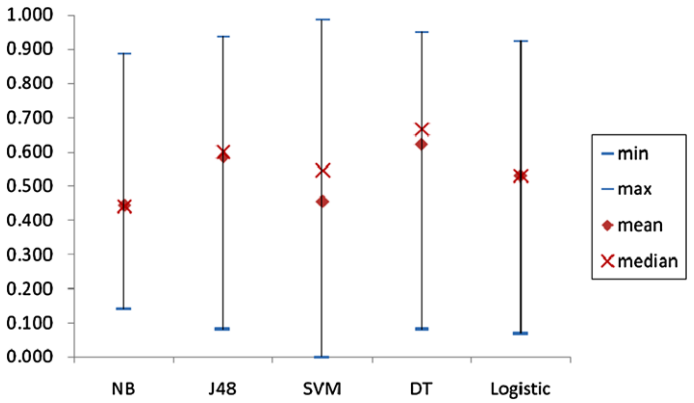

Fig. 12 Comparison of *Recall* obtained from the most suitable training data in scenario C

paired t-test to check whether prediction performance increase over the 5 predictors studied at the confidence level 0.95.

As illustrated in Table 5, F values in scenario A are significantly greater than those in scenario B for 17 releases, while differences of F values of the other 17 releases are not significant. We get similar observations for the indicator *Recall*, 18 releases have significantly greater *Recall* values in scenario A than in scenario B. However, the *Precision* values turn out to be greater in scenario A than in scenario B for 4 releases and lower for 10 releases. Overall, half of the 34 releases get better prediction results in scenario A (cross-project) than in scenario B (intra-release). Table 5 also shows project-wise comparisons between scenario A and scenario C. 19 releases have significantly greater F values in scenario A and the other 15 releases are comparable. *Recall* values are greater in scenario A for 14 releases than in scenario C and lower for 1 release. *Precision* values are greater in scenario A for 11 releases and lower for 3 releases. Performance of prediction results have significant improvements in scenarios A (cross-project) when compared with scenario C (intra-project) for more

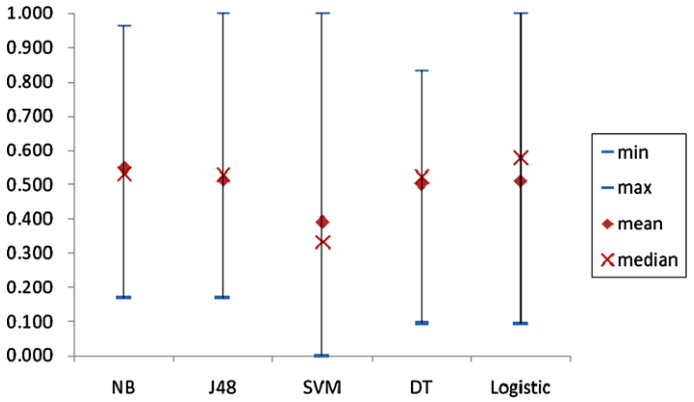

Fig. 13 Comparison of *Precision* obtained from the most suitable training data in scenario C

than half of the releases (19 out of 34). Observations from Table 5 are consistent with those from Table 4: predictions in scenario A get better results than those in scenarios B and C.

5.3 Discussions

Based on findings in Experiment 2, we answer Q2 as below:

A2: Training data from the same project doesn't always lead to better prediction results than training data from other projects. To the contrary, for more than half of the releases, training data from other projects (most suitable training sets in scenario A) may provide better prediction results.

The low precision prediction results in scenario C are unexpected. Different releases of the same projects are usually developed in similar contexts. Many studies reported that they predict each other well (Tosun et al. 2009; Ostrand et al. 2005; Weyuker et al. 2009; Weyuker and Ostrand 2008). However, in our experiments, we did not observe confirming evidences. One potential explanation is that the static code attributes used in this study are not informative enough for defect prediction, especially for prediction among releases in the same project. We found that some classes owning the same feature attribute values do not have the same class label (i.e., defect-prone or defect-free) in different releases. For example, in the project *poi*, the class *org.apache.poi.dev.RecordGenerator* has one defect in version 1.5 and has no defect in version 2.0RC1 while the values of feature attributes are identical. This kind of data is called noisy data, and it decreases prediction accuracy seriously (Khoshgoftaar et al. 2005; Hulse and Khoshgoftaar 2009). In scenario C, when using *poi-2.0* data to predict defects in release *poi-2.5* we got very poor prediction result with *Recall* equal to *0.089, Precision* equal to *0.688* and F equal to *0.157*. There are 314 classes in release *poi-2.0* and 385 classes in release *poi-2.5*, but 185 classes in these

Table 5 Project-wise comparisons between scenarios A, B, and C, where ↓/↑ means performance indicator in scenario A is lower/greater than that in scenario B or C, and—means the difference is not significant. The number in parentheses is significance level of each paired t-test

Release	Scenario A vs. scenario B			Scenario A vs. scenario C		
	F	Recall	Precision	F	Recall	Precision
ant-1.3	↑ (0.005)	↑ (0.050)	↑ (0.006)	– (0.213)	– (0.264)	– (0.400)
ant-1.4	↑ (0.035)	– (0.085)	– (0.161)	↑ (0.043)	↑ (0.048)	– (0.168)
ant-1.5	↑ (0.006)	– (0.063)	– (0.243)	– (0.383)	– (0.562)	– (0.918)
ant-1.6	– (0.124)	– (0.074)	– (0.322)	– (0.193)	– (0.154)	– (0.431)
ant-1.7	– (0.218)	– (0.071)	– (0.877)	– (0.262)	– (0.118)	– (0.892)
camel-1.0	↑ (0.004)	↑ (0.019)	↑ (0.008)	– (0.070)	– (0.182)	– (0.205)
camel-1.2	↑ (0.027)	↑ (0.001)	↓ (0.006)	↑ (0.003)	↑ (0.000)	↓ (0.005)
camel-1.4	↑ (0.013)	↑ (0.002)	– (0.193)	– (0.160)	↑ (0.012)	– (0.191)
camel-1.6	↑ (0.026)	↑ (0.006)	– (0.563)	– (0.063)	↑ (0.016)	– (0.107)
ivy-1.1	↑ (0.009)	↑ (0.013)	– (0.362)	↑ (0.000)	↑ (0.000)	– (0.983)
ivy-1.4	↑ (0.001)	↑ (0.004)	↑ (0.014)	↑ (0.003)	– (0.455)	↑ (0.006)
ivy-2.0	↑ (0.011)	↑ (0.029)	– (0.344)	↑ (0.009)	– (0.565)	↑ (0.027)
jedit-3.2	– (0.136)	– (0.054)	– (0.757)	↑ (0.031)	↑ (0.008)	– (0.486)
jedit-4.0	↑ (0.049)	↑ (0.032)	– (0.168)	– (0.162)	– (0.342)	↑ (0.044)
lucene-2.0	↑ (0.003)	↑ (0.029)	– (0.672)	↑ (0.013)	– (0.710)	– (0.576)
lucene-2.2	– (0.067)	↑ (0.048)	– (0.419)	↑ (0.046)	↑ (0.019)	– (0.146)
lucene-2.4	– (0.244)	↑ (0.022)	↓ (0.009)	– (0.099)	↑ (0.014)	↓ (0.032)
poi-1.5	– (0.215)	– (0.156)	– (0.299)	– (0.669)	– (0.225)	– (0.077)
poi-2.0	↑ (0.015)	↑ (0.012)	– (0.274)	↑ (0.004)	– (0.111)	↑ (0.011)
poi-2.5	– (0.216)	– (0.157)	↓ (0.038)	– (0.246)	– (0.168)	– (0.875)
poi-3.0	– (0.281)	– (0.309)	– (0.789)	– (0.130)	– (0.169)	– (0.525)
synapse-1.0	↑ (0.003)	– (0.068)	↑ (0.011)	↑ (0.003)	– (0.722)	↑ (0.000)
synapse-1.1	– (0.140)	– (0.143)	– (0.507)	– (0.062)	– (0.110)	↑ (0.037)
synapse-1.2	– (0.195)	– (0.099)	↓ (0.020)	↑ (0.038)	↑ (0.020)	– (0.212)
velocity-1.4	– (0.158)	– (0.578)	↓ (0.018)	↑ (0.005)	↑ (0.001)	↑ (0.042)
velocity-1.5	– (0.278)	– (0.078)	↓ (0.032)	– (0.291)	– (0.206)	↑ (0.001)
velocity-1.6	– (0.059)	↑ (0.021)	– (0.529)	↑ (0.003)	↓ (0.003)	↑ (0.002)
xalan-2.4	↑ (0.016)	↑ (0.016)	– (0.750)	↑ (0.031)	– (0.138)	↑ (0.034)
xalan-2.5	– (0.230)	↑ (0.027)	↓ (0.024)	↑ (0.019)	↑ (0.000)	↓ (0.006)
xalan-2.6	– (0.302)	– (0.065)	↓ (0.021)	↑ (0.029)	↑ (0.045)	– (0.720)
xerces-1.2	↑ (0.006)	↑ (0.003)	↓ (0.036)	– (0.129)	– (0.375)	– (0.938)
xerces-1.3	↑ (0.001)	↑ (0.002)	– (0.319)	↑ (0.032)	– (0.160)	– (0.051)
xerces-1.4	– (0.098)	– (0.789)	↓ (0.015)	↑ (0.001)	↑ (0.001)	– (0.194)
xerces-init	– (0.400)	– (0.080)	– (0.137)	↑ (0.000)	↑ (0.001)	↑ (0.017)
# ↑	17	18	4	19	14	11
# ↓	0	0	10	0	1	3
# –	17	16	20	15	19	20

Table 6 Ratios of successful cross-project defect prediction in scenario A

Predictor	# total	# successful	Successful (%)
NB	160586	515	0.32%
J48	160586	2534	1.58%
SVM	160586	7495	4.67%
DT	160586	3965	2.47%
Logistic	160586	3399	2.12%

two releases are noisy. So it is easy to understand why *poi-2.0* data do not predict defects in *poi-2.5* well.[6]

6 Experiment 3. Are characteristics of data sets valuable for selecting suitable training data for cross-project defect prediction?

Although the results of scenario A show cross-project defect predictions work in some cases, the ratios of successful cross-project predictions are very low. Successful cross-project predictions here refer to predictions whose results meet our criteria, i.e., *Recall* greater than 70% and *Precision* greater than 50%. Table 6 shows the ratios of successful cross-project predictions in scenario A, where #total indicates the number of all cross-project defect predictions and #successful the number of successful predictions. For the 5 predictors, the ratios of successful cross-project predictions range from 0.32% to 4.67%. This finding is similar to what Zimmermann et al. have observed; they reported that only 3.4% cross-project defect predictions are successful (both *Recall* and *Precision* greater than 75%) (Zimmermann et al. 2009). Although the criteria used in this study and Zimmermann's are different, low ratios of successful predictions in these two studies indicate a same fact that we should select training data more carefully in the context of cross-project defect prediction.

We choose cross-project predictions based on *DT* algorithm in scenario A as the object of the following study, because the *DT* based predictor performs best in scenario A.

6.1 Design

6.1.1 Capture distributional characteristics of data set

One core challenge of selecting training data for cross-project defect prediction by considering data set characteristics is that how to capture these "characteristics" properly. Here in this paper, we see characteristics of a data set as the collection of distributional characteristics of each feature attributes. Figure 14 illustrates the proce-

[6]The reason why we still consider using data of the *poi* project is that noisy data do exist in some practical projects, and intentionally avoiding noisy data may reduce practicability of conclusions drawn from these data even though you can get better experimental results.

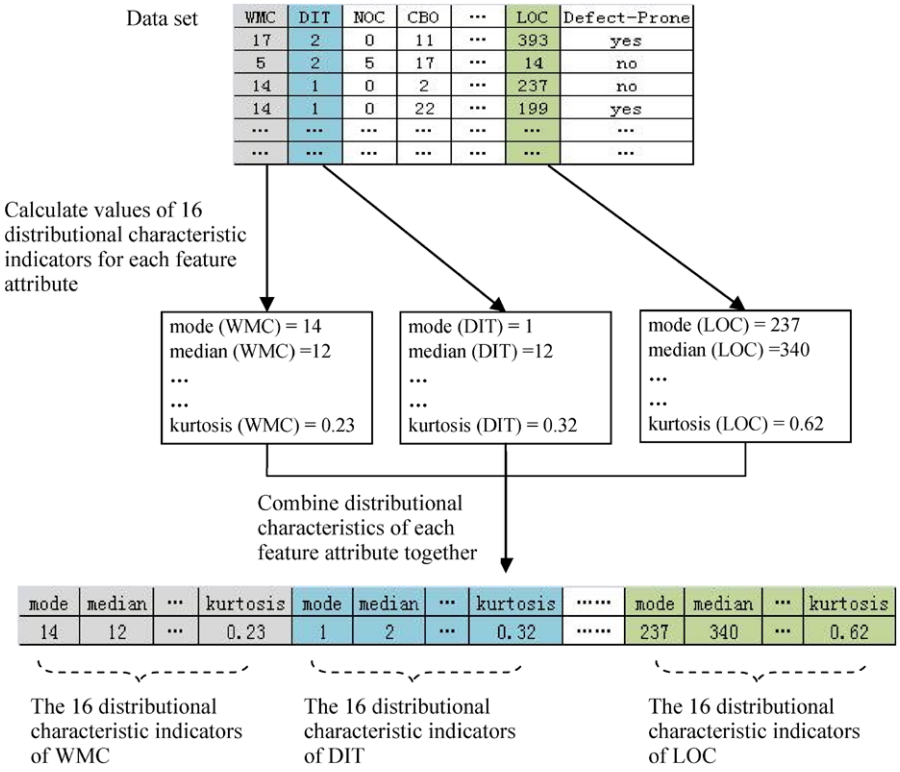

Fig. 14 Capturing distributional characteristics of a data set

dure of capturing distributional characteristics of a given data set. First, for each feature attribute in a training/test set, we calculate 16 indicators (listed in Table 7) to describe its distribution. Then we combine these indicators together, which makes a set of 320 indicators (i.e., 20 feature attributes multiplying 16 indicators). These 320 indicators are used to describe the distributional characteristics of a given training/test set.

The indicators *Mode*, *Median*, *Mean*, and *Harmonic Mean* describe the central tendency of attribute values, and the indicators *Range*, *Variation Ratio*, *Interquartile Range*, *Variance*, *Standard Deviation*, and *Coefficient of Variation* describe the dispersion of attribute values.

6.1.2 Verify the relationship between distributional characteristics of data sets and prediction results

Previous work of Zimmermann et al. (2009) and Watanabe et al. (2008) show that data characteristics are potential determinants for successful cross-project defect prediction. So, to answer research question Q3, we investigate the relationship between the results of cross-project defect predictions and distributional characteristics of training sets and test sets.

Table 7 Descriptions of indicators used to describe the distributional characteristics of a data set

Indicator	Description
Mode	The value that occurs most frequently in a population
Median	The numeric value separating the higher half of a population from the lower half
Mean	The average value of samples in a population; specifically, it refers to arithmetic mean in this paper
Harmonic Mean	The reciprocal of the arithmetic mean of the reciprocals
Minimum	The greatest value in a population
Maximum	The least value in a population
Range	The numeric area of values in a population; equals to the deviation of the Minimum to the Maximum
Variation Ratio	The proportion of cases that are not the mode
First Quartile	The value cutting off 25% lowest cases in a population
Third Quartile	The value cutting off 75% lowest cases in a population
Interquartile Range	The deviation of the First Quartile to the Third Quartile
Variance	The arithmetic mean of the squared deviation of the Mean to values of cases in a population
Standard Deviation	The square root of the Variance
Coefficient of Variation	The ratio of the Standard Deviation to the Mean
Skewness	A measure of the asymmetry of a population $Skewness = \frac{\sum(X-\overline{X})^3}{\sigma^3 N}$
Kurtosis	A measure of the peakedness of a population $Kurtosis = \frac{\sum(X-\overline{X})^4}{\sigma^4 N}$

For each cross-project prediction in scenario A, we generate an instance which is called a train-test-result instance in this paper. A train-test-result instance consists of three parts: distributional characteristics of the training set, distributional characteristics of the test set, and the prediction result. Different from Experiments 1 and 2, the prediction results here are discretized to binary values: prediction results meeting our criteria are labeled to "*yes*" while the others are labeled to "*no*". For each cross-project prediction in scenario A, we get a 641-attribute instance: 320 attributes describing distributional characteristics of the training set, 320 attributes describing distributional characteristics of the test set, and 1 attribute indicating whether the prediction is successful (see Fig. 15).

We follow the procedure of Zimmermann et al. (2009), using a decision tree to investigate whether prediction results of cross-project defect predictions are related to distributional characteristics of training sets and test sets. We construct a C4.5 decision tree on the *train-test-result* instances generated. If the decision tree can identify those successful cross-project defect predictions precisely, it means that distributional characteristics of data set are related to prediction results in the cross-project context. We conduct 5-folder cross-validation to verify how precisely the C4.5 decision tree can identify successful cross-project predictions.

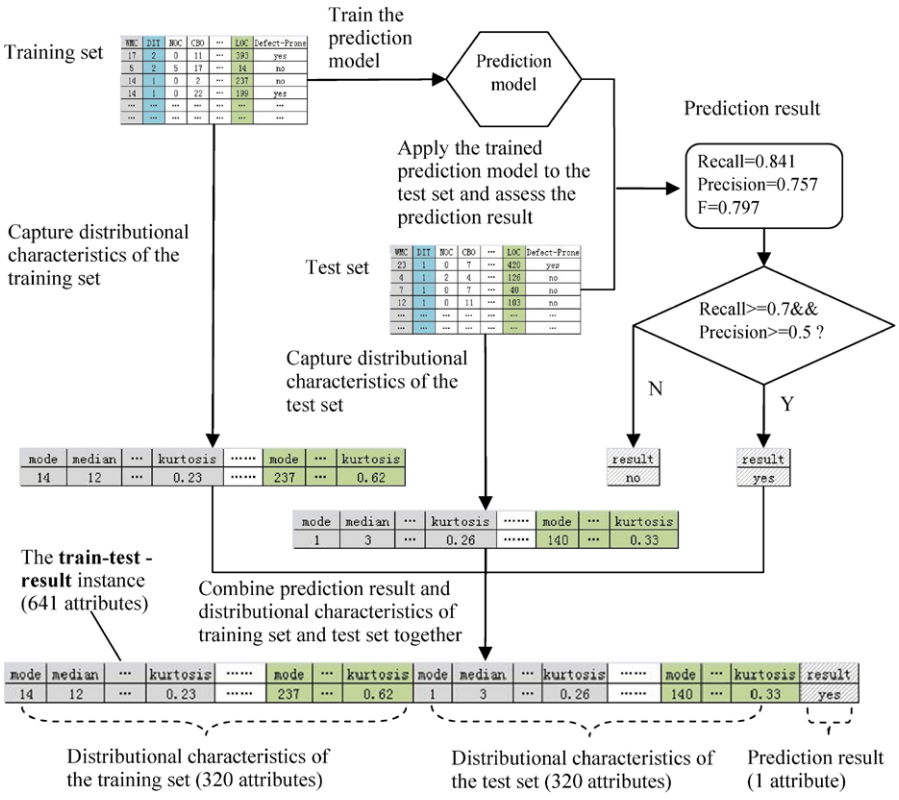

Fig. 15 Procedure of generating a *train-test-result* instance from a cross-project defect prediction

6.1.3 Automatically select training data from other projects

To further investigate the value of distributional characteristics of data sets for selecting training data, we propose a three-step approach to automatically select training data for projects without historical data. Our approach can be described as below:

Step I: Run cross-project defect predictions on available data in the same way of scenario A and record the prediction results as well as the distributional characteristics of the training sets and test sets. For example, if selecting training data for the release *ant-1.7*, we conduct cross-project defect predictions on the 29 data sets of the other 9 projects respectively, and then generate train-test-result instances for each cross-project prediction.

Step II: Construct a C4.5 decision tree on the train-test-result instances generated in Step I.

Step III: Input the distributional characteristics of the data set to be predicted and those of a candidate training set into the decision tree constructed in Step II. If the output is the "yes", the candidate training set is suitable for the data set to be predicted; otherwise, the candidate training set is not suitable for the data set to be predicted. For data sets that do not find any suitable train-

ing sets, they may contain special defect patterns that cannot be learned from available data, and we suggest other defect prediction approaches that are independent of empirical data.

In Experiment 3, we design another scenario of defect prediction that implements the above steps:

Scenario D: using the data set selected by the approach we proposed as the training data. Using the approach above, we select training data for each studied data sets. We then investigate the effectiveness of our approach by analyzing the prediction results provided by the selected training data.

6.2 Results

We generate 160586 train-test-result instances from predictions in scenario A. The best result of 5-folder cross-validation we observed is that *Recall* equal to *68.3%* and *Precision* equal to *73.9%*. Considering that only 2.47% cross-project predictions are successful (see Table 6), we think the performance of this decision tree is fairly high. The high performance of the decision tree indicates that the distributional characteristics of training set and test set are related to results of cross-project defect predictions.

The decision tree learned from train-test-result instances contains 642 leaf nodes, among which 270 nodes are labeled as "*yes*", indicating the decision tree learned 270 different rules for detecting successful cross-project defect predictions. The top 5 rules drawn from the decision tree to detect successful cross-project defect predictions are listed in Table 8, where #support indicates the number of predictions satisfying the rule. It is hard to get general patterns of relationships between prediction results and particular distributional characteristic, because different distributional characteristics usually exhibit interaction effects when classifying instances. However, we can simply rank the 640 distributional characteristics by their information gain. Information gain is a concept derived from information entropy which presents the amount of information a particular distributional characteristic can provide for classification. Information gain indicates how important the distributional characteristic is for judging whether a cross-project defect prediction is successful. Figure 16 illustrates the information gain of each distributional characteristic after ranking. One interesting observation is that distributional characteristics of test sets generally have greater information gain than those of training sets. Intuitively, this observation point out that distributional characteristics of test sets are more informative.

Table 9 shows the results of scenario D, where #training sets indicates the number of training sets selected by our approach for the given data set; the *mean* (*Recall*), *mean* (*Precision*), and *mean* (F) represent the mean values of *Recall*, *Precision*, and F of prediction results provided by these selected training sets, respectively.

Our major observations include:

(1) 24 releases out of 34 get training data from other projects using our approach while the other 10 do not.
(2) 14 releases are predicted with *Recall* greater than *0.7*, and 9 releases are predicted with *Precision* greater than *0.5*, and 5 releases are well predicted with both *Recall* greater than *0.7* and *Precision* greater than *0.5*.

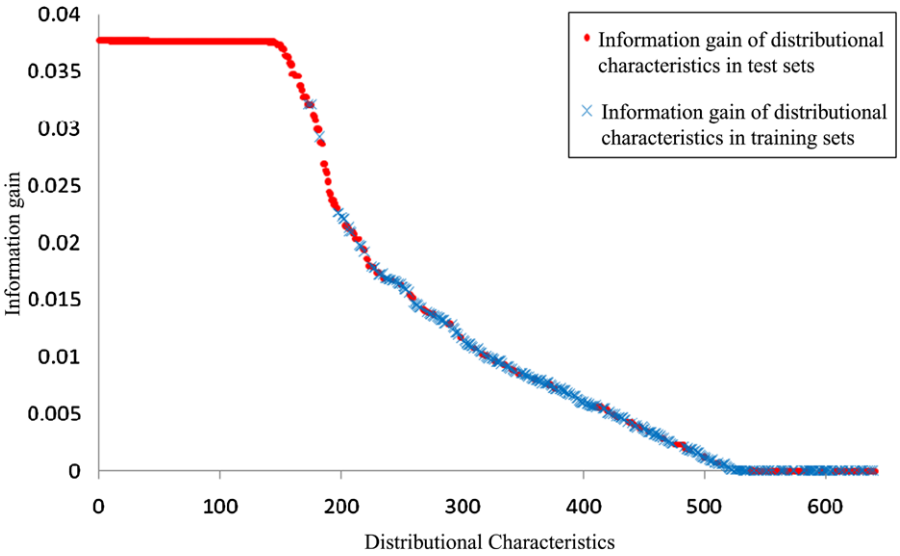

Fig. 16 Information gain of each distributional characteristic of training sets and test sets

(3) The *mean (Recall)* values are greater than those in scenarios B and C in Experiment 2 (constructing predictors using data in the same release or project), indicating more defect-prone classes are detected by the training data selected by our approach. However, the mean *Precision* values are lower, while the mean *F* values are comparable with those in scenarios B and C.

6.3 Discussions

Menzies et al. (2007a) suggested that defect models with high *Recall* but low *Precision* are still useful in many cases. Therefore, the finding that the values of *mean (Recall)* values of prediction results in scenario D are high shows that the distributional characteristics are valuable for the selection of training data and our approach for training data selection is beneficial for projects without historical data.

Based on the results, here is our answer to Q3:

A3: The distributional characteristics of data sets are informative for training data selection. In addition, by constructing a decision tree on train-test-result instances generated from available historical data using the approach proposed above, we may find out cross-project training data for projects with prediction results comparable with the prediction results provided by data in the same project.

Our findings in scenario D are similar to what Turhan et al. (2009) observed (Cross-company data increase the probability of defect detection at the cost of increasing false positive rate). We need to further verify whether it is a common phenomenon in the context of defect prediction.

Table 8 Rules learned by C4.5 decision tree to detect successful cross-project defect predictions

Rule	#support
IF (Coefficient_of_Variation (IC-test)[a] ≤ 0.973368 and Coefficient_of_Variation (Avg_CC-train) > 1.037086 and Kurtosis (Avg_CC-train) ≤ 61.206369 and Coefficient_of_Variation (WMC-train) ≤ 2.058551 and Variance (CBO-train) ≤ 255.060496 and (Ce-train) ≤ 6.302122 and (WMC-train) ≤ 10.900778 and Kurtosis (MOA-train) ≤ 70.909677 and Variance (Max_CC-train) > 61.029769 and Coefficient_of_Variation (NPM-train) ≤ 1.460476) THEN result = "yes"	588
IF (Coefficient_of_Variation (IC-test) > 0.973368 and Coefficient_of_Variation (Avg_CC-train) > 1.060936 and Coefficient_of_Variation (Ca-test) ≤ 1.95824 and (DAM-train) ≤ 0.428485 and Kurtosis (MOA-train) ≤ 93.542587 and Standard_Deviation (Ce-train) ≤ 8.294951 and Max (LCOM-train) ≤ 11469 and Skewness (Max_CC-train) ≤ 13.2318 and Mode (CBO-test) > 3 and Third_Quartile (Max_CC-train) > 2 and Coefficient_of_Variation (AMC-train) ≤ 3.162814 and Kurtosis (MFA-train) > 1.237675 and Variation_Ratio (NOC-train) > 0.092997 and Quartile_Deviation (RFC-train) ≤ 24 and Skewness (MOA-train) ≤ 7.136721 and Coefficient_of_Variation (DAM-train) > 1.09936 and Variance (WMC-train) > 108.460206) THEN result = "yes"	508
IF (Coefficient_of_Variation (IC-test) ≤ 0.973368 and Coefficient_of_Variation (Avg_CC-train) ≤ 1.037086 and Skewness (Ca-train) ≤ 4.842674 and Mean (CBO-train) > 7.287356 and Variance (LCOM-train) ≤ 652861.833171 and Variance (DAM-train) > 0.184748 and Variance (CAM-train) ≤ 0.062436 and Mean (Ce-train) > 4.022549 and Variance (WMC-train) > 86.643885 and Skewness (NPM-train) ≤ 6.804858 and Median (LCOM-train) > 2.5 and Variance (CAM-train) > 0.047754 and Kurtosis (MOA-train) ≤ 83.628131 and Coefficient_of_Variation (AMC-train) > 1.053063 and Mean (RFC-train) ≤ 25.332732) THEN result = "yes"	120
IF (Coefficient_of_Variation (IC-test) > 0.973368 and Coefficient_of_Variation (Avg_CC-train) > 1.060936 and Coefficient_of_Variation (Ca-test) ≤ 1.95824 and Mean (DAM-train) ≤ 0.428485 and Kurtosis (MOA-train) ≤ 93.542587 and Standard_Deviation (Ce-train) ≤ 8.294951 and Max (LCOM-train) ≤ 11469 and Skewness (Max_CC-train) ≤ 13.2318 and Mode (CBO-test) > 3 and Third_Quartile (Max_CC-train) > 2 and Coefficient_of_Variation (AMC-train) ≤ 3.162814 and Kurtosis (MFA-train) ≤ 1.237675 and Max (MOA-train) ≤ 34 and Max (WMC-train) > 123 and Median (WMC-test) > 6 and Third_Quartile (AMC-train) > 27.1 and Skewness (Ce-train) ≤ 3.435897) THEN result = "yes"	89
IF (Coefficient_of_Variation (IC-test) > 0.973368 and Coefficient_of_Variation (Ca-test) ≤ 1.95824 and Mean (DAM-train) ≤ 0.428485 and Kurtosis (MOA-train) ≤ 93.542587 and Standard_Deviation (Ce-train) ≤ 8.294951 and Max (LCOM-train) ≤ 11469 and Skewness (Max_CC-train) ≤ 13.2318 and Mode (CBO-test) > 3 and Third_Quartile (Max_CC-train) > 2 and Coefficient_of_Variation (AMC-train) ≤ 3.162814 and Kurtosis (MFA-train) > 1.237675 and Variation_Ratio (NOC-train) > 0.092997 and Quartile_Deviation (RFC-train) ≤ 24 and Skewness (MOA-train) > 7.136721 and Quartile_Deviation (CBO-train) > 5 and Variance (LCOM-train) ≤ 96756.608908 and Mean (LOC-train) ≤ 296.985004 and Skewness (MFA-train) ≤ 0.566226 and Kurtosis (Ce-train) > 13.008719) THEN result = "yes"	84

[a]Coefficient_of_Variation (IC-test) means the coefficient of variation of feature attribute IC in the test set

One drawback of our training data selection method is that the efficiency is not sufficiently high,because the number of candidate training sets is huge. It exponentially increases over the number of available data sets. Testing all candidates is time consuming; we need to find more efficient methods for training data selection.

Table 9 Prediction results provided by training data selected by our approach in scenario D

Release	# training sets	Mean (Recall)	Mean (Precision)	Mean (F)
ant-1.3	98	0.901	0.231	0.365
ant-1.4	0	–	–	–
ant-1.5	0	–	–	–
ant-1.6	0	–	–	–
ant-1.7	0	–	–	–
camel-1.0	0	–	–	–
camel-1.2	0	–	–	–
camel-1.4	0	–	–	–
camel-1.6	0	–	–	–
ivy-1.1	131	0.815	0.621	0.695
ivy-1.4	17	0.721	0.112	0.194
ivy-2.0	17	0.835	0.214	0.340
jedit-3.2	22	0.672	0.343	0.451
jedit-4.0	22	0.727	0.276	0.397
lucene-2.0	92	0.748	0.509	0.599
lucene-2.2	72	0.715	0.607	0.649
lucene-2.4	0	–	–	–
poi-1.5	4	0.707	0.698	0.698
poi-2.0	4	0.770	0.151	0.251
poi-2.5	4	0.717	0.742	0.725
poi-3.0	12	0.621	0.844	0.704
synapse-1.0	39	0.925	0.127	0.223
synapse-1.1	37	0.807	0.332	0.468
synapse-1.2	37	0.839	0.429	0.564
velocity-1.4	0	–	–	–
velocity-1.5	0	–	–	–
velocity-1.6	0	–	–	–
xalan-2.4	20	0.815	0.234	0.363
xalan-2.5	20	0.568	0.551	0.557
xalan-2.6	20	0.548	0.512	0.526
xerces-init	59	0.569	0.491	0.516
xerces-1.2	167	0.440	0.176	0.243
xerces-1.3	167	0.623	0.242	0.338
xerces-1.4	21	0.498	0.898	0.636
Average	–	0.708	0.425	0.477

7 Threats to validity

The first threat to our results is the definition of acceptable prediction results in this paper. We use *Recall* greater than 70% and *Precision* greater than 50% as the criteria of judging whether prediction results are acceptable. Our choice is based on some

previous studies and our own research experience, while others may consider different criteria for acceptance. Some of our observations and conclusions may change if adopting different criteria.

The second threat is the static code metrics used in our experiment. Empirical data of releases in the same project showed limited predictive power in scenario B and C, while some other researchers argued that releases in the same projects predict each other well (Tosun et al. 2009; Ostrand et al. 2005; Weyuker et al. 2009; Weyuker and Ostrand 2008). We think it is because static code features we used are not informative enough for defect prediction. Actually the effectiveness of static code metrics has been debated widely (Menzies et al. 2007b; Turhan et al. 2009; Fenton and Ohlsson 2000; Shepped and Ince 1994). So we limit our conclusions at the context of using static code metrics, more specifically, the 20 static code metrics used in our experiments.

Another threat comes from data collection. According to the statement of Jureczko and Madeyski (2010), Jureczko and Spinellis (2010), defects were identified by comments in the source code version control systems using regular expression matching technology, and then were assigned to classes according to date of fixing. Jureczko et al. admit that there are maybe mistakes in defect identification and assignment.

What's more, the projects we studied are all open source projects; we cannot assure conclusions drawn from these projects are generally applicable for commercial proprietary closed software.

8 Conclusions and future work

Cross-project defect prediction is important for projects without historical data. It extends the application field of defect prediction models, e.g., predicting defects in the first release of a new project. However, empirical evidences show that cross-project defect predictions only work in a few cases. Training data is very important for machine learning based defect prediction. To product a successful cross-project prediction, we need to select the training data from other projects carefully.

In this study, we reported results from three experiments to investigate on the usability of cross-project defect prediction with regard to training data selection. The three large-scale experiments were conducted on 34 data sets collected from 10 open source projects. Each experiment focuses on one research question we presented. Our findings partly support what Zimmermann et al. (2009) and Turhan et al. (2009) reported. Cross-project defect prediction only works in few cases, while training data from other projects is still valuable for constructing defect prediction models for project without historical data if selected carefully. More specifically, conclusions from our study are as follows:

(1) In the best cases, cross-project data may provide acceptable prediction results.
(2) Training data from the same project does not always lead to better prediction results than training data from other projects. To the contrary, carefully selected cross-project training data may provide better prediction results than training data from the same project.

(3) Data distributional characteristics are informative for training data selection. Based on this finding, we proposed a training data selection method. Prediction results provided by cross-project training data selected using our method are comparable with those provided by training data from the same project.

Although results of our experiments show that cross-project defect prediction works by selecting training data carefully, many problems still need to be solved in this context. Our future work includes:

(1) Testing the performance of more metrics in the context of cross-project defect prediction, e.g., code change history, process metrics, project characteristics, and some other static code metrics (Chidamber and Kemerer 1994; Ohlsson and Alberg 1996; Nagappan and Ball 2005; Moser et al. 2008).
(2) Repeating our experiments on more projects. Although it is difficult to draw general conclusions from empirical studies, results of larger scale experiments are more convincing.
(3) Improving the efficiency and effectiveness of training data selection. The efficiency of our selection method is influenced by the number of candidate training sets. We plan to analyze more potential determinants for successful cross-project defect prediction and develop more efficient and more precise training data selection methods.

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant Nos. 60873072, 61073044, and 60903050; the National Science and Technology Major Project; the National Basic Research Program under Grant No. 2007CB310802; CAS Innovation Program. We thank Marian Jureczko and Lech Madeyski for sharing defect data sets used in this paper. We also thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper.

References

Boetticher, G., Menzies, T., Ostrand, T.J.: PROMISE repository of empirical software engineering data. http://promisedata.org/repository (2007). Accessed 12 December 2010

Carvalho, A.B., Pozo, A., Vergilio, S.R.: A symbolic fault-prediction model based on multiobjective particle swarm optimization. J. Syst. Softw. **83**(5), 7346–7354 (2010)

Catal, C., Diri, B.: A systematic review of software fault prediction studies. Expert Syst. Appl. **36**(4), 7346–7354 (2009)

Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)

D'Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, pp. 31–41 (2010)

Fenton, N., Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Softw. Eng. **26**(8), 797–814 (2000)

Hassan, A.E., Holt, R.C.: The top ten list: dynamic fault prediction. In: Proceedings of the 21st IEEE International Conference on Software Maintenance, pp. 263–272 (2005)

Hulse, J.V., Khoshgoftaar, T.: Knowledge discovery from imbalanced and noisy data. Data Knowl. Eng. **68**(12), 1513–1542 (2009)

Jiang, Y., Cukic, B., Ma, Y.: Techniques for evaluating fault prediction models. Empir. Softw. Eng. **13**(15), 561–595 (2008)

Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp. 1–10 (2010)

Jureczko, M., Spinellis, D.: Using object-oriented design metrics to predict software defects. In: Proceedings of the 5th International Conference on Dependability of Computer Systems, pp. 69–81 (2010)

Khoshgoftaar, T.M., Seliya, N., Drown, D.J.: Evolutionary data analysis for the class imbalance problem. Intell. Data Anal. **14**(1), 69–88 (2010)

Khoshgoftaar, T.M., Zhong, S., Joshi, V.: Enhancing software quality estimation using ensemble-classifier based noise filtering. Intell. Data Anal. **9**(1), 3–27 (2005)

Kocaqueneli, E., Gay, G., Menzies, T., Yang, Y., Keung, J.W.: When to use data from other projects for effort estimation. In: Proceedings of the 25th International Conference on Automated Software Engineering, pp. 321–324 (2010)

Koru, A.G., Liu, H.: Building effective defect-prediction models in practice. IEEE Softw. **22**(6), 23–29 (2005)

Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. **34**(4), 485–496 (2008)

Li, Q., Yang, Y., Li, M., Wang, Q., Boehm, B.W., Hu, C.: Improving software testing process: feature prioritization to make winners of success-critical stakeholders. J. Softw. Maint. Evol.: Res. Pract. (2010, published online)

Menzies, T., Dekhtyar, A., Distefano, J., Greenwald, J.: Problems with precision: a response to "Comments on 'Data mining static code attributes to learn defect predictors' ". IEEE Trans. Softw. Eng. **33**(9), 637–640 (2007a)

Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. **33**(1), 2–13 (2007b)

Menzies, T., Turhan, B., Benser, A., Gay, G., Cukic, B., Jiang, Y.: Implications of ceiling effects in defect predictors. In: Proceedings of the 4th International Conference on Predictive Models in Software Engineering, pp. 47–54 (2008)

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A.: Defect prediction from static code features: current results, limitations, new approaches. Autom. Softw. Eng. **17**(4), 375–407 (2010)

Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th International Conference on Software Engineering, pp. 181–190 (2008)

Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th International Conference on Software Engineering, pp. 284–292 (2005)

Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th International Conference on Software Engineering, pp. 452–461 (2006)

Ohlsson, N., Alberg, H.: Predicting fault-prone software modules in telephone switches. IEEE Trans. Softw. Eng. **22**(12), 886–894 (1996)

Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. **31**(4), 340–355 (2005)

Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo (1993)

Shepped, M., Ince, D.: A critique of three metrics. J. Syst. Softw. **26**(3), 197–210 (1994)

Tosun, A., Turhan, B., Bener, A.: Practical considerations in deploying AI for defect prediction: a case study within the Turkish telecommunication industry. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering, pp. 1–9 (2009)

Tosun, A., Bener, A., Kale, R.: AI-based software defect predictors: applications and benefits in a case study. In: Proceedings of the 22th Innovative Applications of Artificial Intelligence Conference, pp. 1748–1755 (2010)

Turhan, B., Menzies, T., Bener, A.: On the relative value of cross-company and within_company data for defect prediction. Empir. Softw. Eng. **14**(5), 540–578 (2009)

Turhan, B., Bener, A., Menzies, T.: Regularities in learning defect predictors. In: The 11th International Conference on Product Focused Software Development and Process Improvement, pp. 116–130 (2010)

Vapnik, V.: Statistical Learning Theory. Wiley-Interscience, New York (1998)

Wahyudin, D., Ramler, D., Biffl, S.: A framework for defect prediction in specific software project contexts. In: The 3rd IFIP Central and East European Conference on Software Engineering Techniques (2008)

Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the International Workshop on Predictive Models in Software Engineering, pp. 19–24 (2008)

Weyuker, E.J., Ostrand, T.J.: What can fault prediction do for you? Lect. Notes Comput. Sci. **4966**, 18–29 (2008)

Weyuker, E.J., Ostrand, T.J., Bell, R.M.: Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. Empir. Softw. Eng. **13**(5), 539–559 (2008)

Weyuker, E.J., Ostrand, T.J., Bell, R.M.: Comparing the effectiveness of several modeling methods for fault prediction. Empir. Softw. Eng. **15**(3), 277–295 (2009)

Zhang, H., Zhang, X.: Comments on "Data mining static code attributes to learn defect predictors". IEEE Trans. Softw. Eng. **33**(9), 635–637 (2007)

Zimmermann, T., Nagappan, N., Gall, H.: 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, pp. 91–100 (2009)