Autom Softw Eng (2012) 19:201–230
DOI 10.1007/s10515-011-0092-1

Sample-based software defect prediction with active and semi-supervised learning

Ming Li · Hongyu Zhang · Rongxin Wu · Zhi-Hua Zhou

Received: 16 December 2010 / Accepted: 27 June 2011 / Published online: 29 July 2011

Abstract Software defect prediction can help us better understand and control software quality. Current defect prediction techniques are mainly based on a sufficient amount of historical project data. However, historical data is often not available for new projects and for many organizations. In this case, effective defect prediction is difficult to achieve. To address this problem, we propose sample-based methods for software defect prediction. For a large software system, we can select and test a small percentage of modules, and then build a defect prediction model to predict defect-proneness of the rest of the modules. In this paper, we describe three methods for selecting a sample: random sampling with conventional machine learners, random sampling with a semi-supervised learner and active sampling with active semi-supervised learner. To facilitate the active sampling, we propose a novel active semi-supervised learning method ACoForest which is able to sample the modules that are most helpful for learning a good prediction model. Our experiments on PROMISE datasets show that the proposed methods are effective and have potential to be applied to industrial practice.

M. Li · Z.-H. Zhou
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

M. Li
e-mail: lim@lamda.nju.edu.cn

Z.-H. Zhou
e-mail: zhouzh@lamda.nju.edu.cn

H. Zhang (✉) · R. Wu
MOE Key Laboratory for Information System Security, Tsinghua University, Beijing 100084, China
e-mail: hongyu@tsinghua.edu.cn

R. Wu
e-mail: wrx09@mails.tsinghua.edu.cn

Keywords Software defect prediction · Sampling · Quality assurance · Machine learning · Active semi-supervised learning

1 Introduction

Software quality assurance (SQA) is vital to the success of a software project. However, ensuring software quality is a resource and time-consuming activity, which may include manual code inspections, technical review meetings and intensive software testing. Recently, software defect prediction techniques have been proposed to help allocate limited SQA resources in a cost-effective manner by predicting defect-proneness of software modules. More resources and efforts could be spent on the modules that are likely to be defective (contains at least one defect).

Many defect prediction methods have been proposed in recent years. For example, Menzies et al. (2007) built defect prediction models based on 21 static code attributes. Nagappan et al. (2006) proposed to extract principle components from metrics and use these principles components for defect prediction. Kim et al. (2007) predicted defects by caching locations that are adjusted by software change history data. Zimmermann and Nagappan (2008) proposed models based on the structural measurement of dependency graphs. In our previous work, we also found that simple complexity metrics, such as Lines of Code, can be used to construct effective prediction models (Zhang et al. 2007; Zhang 2009). These methods elaborate to extract features from the defective and non-defective modules (e.g., files, methods or changes), apply machine learning or statistical methods to construct classification model, and then use the model to predict defect-proneness of a new module.

The current defect prediction models are all built using historical data from past projects, assuming that the new projects share the same product/process characteristics and their defects can be predicted based on previous data. This assumption may not be always valid. Firstly, historical data may not be available (and reliable) for some organizations, especially for those do not have well-defined software process. Secondly, even if some previous project data is available, whether it can construct effective prediction models for completely new projects remains to be verified. This is particularly of concern in a rapid changing environment, where the requirements, development personnel, organizational structure, or even process changes.

To address this problem, in this paper, we propose a sampling-based approach to software defect prediction. In our approach, we sample a small percentage of modules, examine the quality of sampled modules, construct a classification model based on the sample and then use it predict the un-sampled modules in the project. In this way, we can build a defect prediction model using data from the current project, independent from past projects.

A difficulty of the sampling approach is that, large software projects (e.g., Eclipse) often consist of hundreds or even thousands of source code files. How to select a sample and how to build an effective defect prediction models based on the sample are major research challenges.

In our approach, we propose to use two semi-supervised machine learning methods called CoForest and ACoForest to construct defect prediction models based on samples. CoForest (Li and Zhou 2007) is a disagreement-based semi-supervised learning algorithm that exploits the advantage of both semi-supervised learning and ensemble learning. It firstly learns an initial classifier from a small sample (labeled data) and refines the classifier by further exploiting a larger number of unsampled modules (unlabeled data). The ACoForest method extends CoForest by actively selecting and labeling some previously unlabeled data for training the classifiers. Our experiments on the public PROMISE datasets show that the prediction models constructed using CoForest and ACoForest can achieve better performance than those using conventional machine learning techniques (such as Logistic Regression, Decision Tree and Naive Bayes). Our results also show that a small sample can achieve similar performance as large samples do.

Unlike the previous studies, our approach does not require historical data. By applying our methods, a project team can randomly select a small sample of modules (e.g., 10%) from the current project, test them for their defect-proneness, and build a predictive model based on the sampled data in order to predict the defect proneness of the rest of modules of the current project. Furthermore, ACoForest also supports the adaptive selection of the modules—it can actively suggest the team which modules to be chosen for testing in order to increase the prediction performance. We believe our methods can be applied to open source and industrial projects to help improve software quality assurance practices.

The organization of the paper is as follows. In Sect. 2, we describe the proposed sample-based defect prediction approach. Section 3 describes our experimental design and results. We discuss the proposed approach in Sect. 4 and present the related work in Sect. 5. Section 6 concludes the paper.

2 Sample-based defect prediction

In this section, we describe the proposed sample-based defect prediction approach. Our approach can be classified into three methods—sampling with conventional machine learners, sampling with semi-supervised learning, and sampling with active semi-supervised learning.

2.1 Sampling with conventional machine learners

Software defect prediction, which aims to predict whether a particular software module contains any defects, can be cast into a classification problem in machine learning, where software metrics are extracted from each software module to form an example with manually assigned labels *defective* (having one or more defects) and *non-defective* (no defects). A classifier is then learned from these training examples in the purpose of predicting the defect-proneness of unknown software modules. In this paper, we propose a sample-based defect prediction approach which does not rely on the assumption that the current project has the same defect characteristics as the historical projects.

Given a newly finished project, unlike the previous studies that leverage the modules in historical projects for classifier learning, sample-based defect prediction manages to sample a small portion of modules for extensive testing in order to reliably label the sampled modules, while the defect-proneness of unsampled modules remains unknown. Then, a classifier is constructed based on the sample of software modules (the labeled data) and expected to provide accurate predictions for the unsampled modules (unlabeled data). Here, conventional machine learners (e.g., Logistic Regression, Decision Tree, Naive Bayes, etc.) can be applied to the classification.

In practice, modern software systems often consist of hundreds or even thousands of modules. An organization is usually not able to afford extensive testing for all modules especially when time and resources are limited. In this case, the organization can only manage to sample a small percentage of modules, test them for defect-proneness. Classifier would have to be learned from a small training set with the defect-proneness labels. Thus, the key for the sample-based defect prediction to be cost-effective is to learn a well-performing classifier while keeping the sample size small.

2.2 Sampling with semi-supervised learning—the CoForest method

To improve the performance of sample-based defect prediction, we propose to apply *semi-supervised learning* for classifier construction, which firstly learns an initial classifier from a small sample of labeled training set and refines it by further exploiting a larger number of available unlabeled data.

In semi-supervised learning, an effective paradigm is known as disagreement-based semi-supervised learning (Zhou and Li 2010), where multiple learners are trained for the same task and the disagreements among the learners are exploited during learning. In this paradigm, unlabeled data can be regarded as a special information exchange "platform". If one learner is much more confident on a disagreed unlabeled example than other learner(s), then this learner will teach other(s) with this example; if all learners are comparably confident on a disagreed unlabeled example, then this example may be selected for query. Many well-known disagreement-based semi-supervised learning methods (Blum and Mitchell 1998; Zhou and Li 2005, 2007; Li and Zhou 2007) have been developed.

In this study, we apply CoForest (Li and Zhou 2007) for defect prediction. It works based on a well-known ensemble learning algorithm named Random Forest (Breiman 2001) to tackle the problems of determining the most confident examples to label and producing the final hypothesis. The pseudo code of CoForest is presented in Table 1. Briefly, it works as follows. Let L denote the labeled data set and U denote the unlabeled data set. First, N random trees are initiated from the training sets bootstrap-sampled from the labeled data set L for creating a random forest. Then, in each learning iteration, each random tree is refined with the original labeled examples L and the newly labeled examples L' selected by its concomitant ensemble (i.e., the ensemble of the other random trees except for the current tree). The learning process iterates until certain stopping criterion is reached. Finally, the prediction is made based on the majority voting from the ensemble of random trees. Note that in this way, CoForest is able to exploit the advantage of both semi-supervised learning and ensemble learning simultaneously, as suggested in Zhou (2009).

In CoForest, the stopping criterion is essential to guarantee a good performance. Li and Zhou (2007) derived a stopping criterion based on the theoretical findings in Angluin and Laird (1988). By enforcing the worst case generalization error of a random tree in the current round to be less than that in the preceded round, they derived

Table 1 Pseudo code of CoForest (Li and Zhou 2007)

```
Algorithm: CoForest
Input:
  the labeled set $L$, the unlabeled set $U$,
  the confident threshold $\theta$, the number of random trees $N$,
Process:
  1. Construct a random forest with $N$ random trees $H = \{h_1, h_2, \dots, h_N\}$
  2. Repeat 3~9 until none of the random trees of $H$ changes
  3. Update the number of iteration, $t$ ($t = 1, 2, \dots$)
  4. For each random tree $h_i$ in $H$, do step 5 ~ 9
  5. Construct concomitant ensemble $H_{-i}$
  6. Use $H_{-i}$ to label all the unlabeled data, and estimate the labeling confidence
  7. Add the unlabeled data whose labeling confidences are above threshold $\theta$ to a newly
     labeled set $L'_{t,i}$
  8. Undersample $L'_{t,i}$ to make (1) holds. If it does not hold, skip step 9
  9. Update $h_i$ by learning a random tree using $L$ cup $L'_{t,i}$
Output:
  $H$, whose prediction is generated by the majority voting from all the component trees
```

that semi-supervised learning process will be beneficial if the following condition is satisfied

$$\frac{\hat{e}_{i,t}}{\hat{e}_{i,t-1}} < \frac{W_{i,t-1}}{W_{i,t}} < 1 \tag{1}$$

where $\hat{e}_{i,t}$ and $\hat{e}_{i,t-1}$ denote the estimated classification error of the i-th random tree in the t-th and $(t-1)$-th round, respectively, and $W_{i,t}$ and $W_{i,t-1}$ denote the total weights of its newly labeled sets $L'_{i,t}$ and $L_{i,t-1}$ in the t-th and $(t-1)$-th round, respectively, and $i \in \{1, 2, \dots, N\}$. For detailed information on the derivation, please refer to Li and Zhou (2007).

The CoForest has been successfully applied to the domain of computer-aided medical diagnosis (Li and Zhou 2007), where conducting a large amount of routine examinations places heavy burden on medical experts. The CoForest algorithm was applied to help learn hypothesis from diagnosed and undiagnosed samples in order to assist the medical experts in making diagnosis.

2.3 Sampling with active semi-supervised learning—the ACoForest method

Although a random sample can be used to approximate the properties of all the software modules in the current projects, a random sample is apparently not data-efficient since random sample neglects the "needs" of the learners for achieving good performance and hence may contain redundant information that the learner has already captured during the learning process. Intuitively, if a learner is trained using the data that the learner needs most for improving its performance, it may require less labeled data than the learners trained without caring its needs for learning; put it another way, if the same number of labeled data is used, the learner that is trained using the labeled data it needs most would achieve better performance than the learner that is trained without caring its needs for learning.

Active learning, which is another major approach for learning in presence of a large number of unlabeled data, aims to achieve good performance by learning with as few labeled data as possible. It assumes that the learner has some control over the data sampling process by allowing the learner to actively select and query the label of some informative unlabeled example which, if the labels are known, may contribute the most for improving the prediction accuracy. Since active learning and semi-supervised learning exploit the merit of unlabeled data from different perspective, they have been further combined to achieve better performance in image retrieval (Zhou et al. 2006), Email spam detection (Xu et al. 2009), etc. Recently, Wang and Zhou (2008) analytically showed that combining active learning and semi-supervised learning is beneficial in exploiting unlabeled data.

In this study, we extend CoForest to incorporate the idea of active learning into the sample-based defect prediction. We propose a novel *active semi-supervised learning* method called ACoForest, which leverages the advantages from both disagreement-based active learning and semi-supervised learning. In detail, let L and U denote the labeled set and unlabeled set, respectively. Similar to CoForest, ACoForest is firstly initiated by constructing a random forest with N random trees over L. Then, ACoForest iteratively exploits the unlabeled data via both active learning and semi-supervised learning. In each iteration, ACoForest firstly labels all the unlabeled examples and computes the degree of agreement of the ensemble on each unlabeled example. Then, it reversely ranks all the unlabeled data according to the degree of agreement, and selects the M top-most disagreed unlabeled data to query their labels from the user. These unlabeled data as well as their corresponding labels are then used to augment L. After that, ACoForest exploits the remaining unlabeled data just as CoForest does. The pseudo-code of ACoForest is shown in Table 2.

3 Experiments

3.1 Experimental settings

To evaluate the effectiveness of sample-based defect prediction methods, we perform experiments using datasets available at the PROMISE website.[1] We have collected the Eclipse, Lucene, and Xalan datasets.

The Eclipse datasets contain 198 attributes, including code complexity metrics (such as LOC, cyclomatic complexity, number of classes, etc.) and metrics about abstract syntax trees (such as number of blocks, number of if statements, method references, etc.) (Zimmermann et al. 2007). The Eclipse defect data was collected by mining Eclipse's bug databases and version archives (Zimmermann et al. 2007). In this study, we experiment with Eclipse 2.0 and 3.0. To show the generality of the results, we use the package-level data for Eclipse 3.0 and the file-level data for Eclipse 2.0. We also choose two Eclipse components: JDT.Core and SWT in Eclipse

[1] http://promisedata.org/.

Table 2 Pseudo code of ACoForest

```
Algorithm: ACoForest
Input:
  the labeled set $L$, the unlabeled set $U$,
  the confident threshold $\theta$, the number of random trees $N$,
  the number of examples to query in each iteration $M$
Process:
  1. Construct a random forest with $N$ random trees $H = \{h_1, h_2, \ldots, h_N\}$
  2. Repeat 3∼11 until none of the random trees of $H$ changes
  3. Update the number of iteration, $t$ $(t = 1, 2, \ldots)$
  4. Find $M$ unlabeled examples in $U$, on whose labeled the random trees in $H$ disagree
      the most.
  5. Query the labels of the selected $M$ unlabeled examples, and places them along with
    the labels into $L$.
  6. For each random tree $h_i$ in $H$, do step 7 ∼ 11
  7. Construct concomitant ensemble $H_{-i}$
  8. Use $H_{-i}$ to label all the unlabeled data, and estimate the labeling confidence
  9. Add the unlabeled data whose labeling confidences are above threshold $\theta$ to a newly
      labeled set $L'_{t,i}$
 10. Undersample $L'_{t,i}$ to make (1) holds. If it does not hold, skip step 11
 11. Update $h_i$ by learning a random tree using $L$ cup $L'_{t,i}$
Output:
  $H$, whose prediction is generated by the majority voting from all the component trees
```

Table 3 The summary of the datasets

Data sets	#attributes	LOC	#instances	#defective modules
Eclipse 3.0 (package)	198	1306K	661	415 (62.78%)
Eclipse 2.0	198	797K	6729	2611 (38.80%)
JDT.Core	198	181K	939	502 (53.46%)
SWT	198	194K	843	208 (24.67%)
Lucene	20	103K	340	203 (59.71%)
Xalan	20	57K	886	411 (46.44%)

3.0 to evaluate the defect prediction performance for smaller Eclipse projects. We only examine the pre-release defects, which are defects reported in the last six months before release. The data are summarized in Table 3.

The Lucene dataset we use contains metric and defect data for 340 source files in Apache Lucene v2.4. The Xalan dataset contains metric and defect data for 229 source files in Apache Xalan v2.6. Both datasets contains 20 attributes, including code complexity metrics (e.g., *Average Cyclomatic Complexity*), object-oriented metrics (e.g., *Depth of Inheitence Tree*) and program dependency metrics (e.g., *Number of Dependent Classes*). The data are also summarized in Table 3.

Having collected the data, we then apply the three methods described in Sect. 2 to construct defect prediction models from a small sample of modules and use them to predict defect-proneness of unsampled modules. We evaluate the performance of all the methods in terms of precision (P), recall (R), F-measure (F) and Balance-measure (B) (Menzies et al. 2007), which are defined as follows.

$$P = \frac{tp}{tp + fp} \tag{2}$$

$$R = \frac{tp}{tp + fn} \tag{3}$$

$$F = \frac{2PR}{P + R} \tag{4}$$

$$B = 1 - \sqrt{\frac{1}{2}\left(\left(\frac{fn}{tp + fn}\right)^2 + \left(\frac{fp}{tn + fp}\right)^2\right)} \tag{5}$$

where tp, fp, tn, fn are the number of defective modules that are predicted as *defective*, the number non-defective modules that are predicted as *defective*, the number non-defective modules that are predicted as *non-defective*, and the number defective module that are predicted as *non-defective*, respectively.

3.2 Method 1: Sampling with conventional learners

We first perform experiments to evaluate the effectiveness of the defect prediction models constructed using randomly selected samples and conventional machine learners. For each data set, we randomly sample a small portion of modules as the labeled training set according to a sampling rate (μ), while the remaining examples are used as the unlabeled and test sets. For example, given a project with 1000 modules, a random sample based on a sampling rate of 5% results in a set with 50 labeled modules and a set with 950 modules without labels. Since it is usually difficult to conduct costly extensive testing for many modules of a large projects (e.g., 50% modules) in practice and the key of sample-based approach is to achieve good performance while keeping the cost of extensive testing low, the sampling intensity should not be large. Here, we use ten different sampling rates $\{5\%, 10\%, 15\%, \ldots, 50\%\}$, in order to simulate different level of sampling intensity. The labeled data set is used to initiate the machine learners. We repeat the sampling process for 100 times, and the average performance of the compared methods is reported.

In this experiment, we adopt three widely-used machine learners, namely Logistic Regression, Naive Bayes, and Decision Tree. We compare the performance of all the methods in terms of precision, recall, F-measure and Balance-measure. The detailed comparisons in terms of precision, recall, F-measure and Balance-measure of the compared methods for the entire 60 experimental settings (i.e., 6 data sets $\times$ 10 sampling rates) are given in Tables 8 to 13 in the Appendix.

The results show that for all classifiers, increasing sample sizes does not significantly improve prediction results. For example, for JDT.Core, increasing sample size

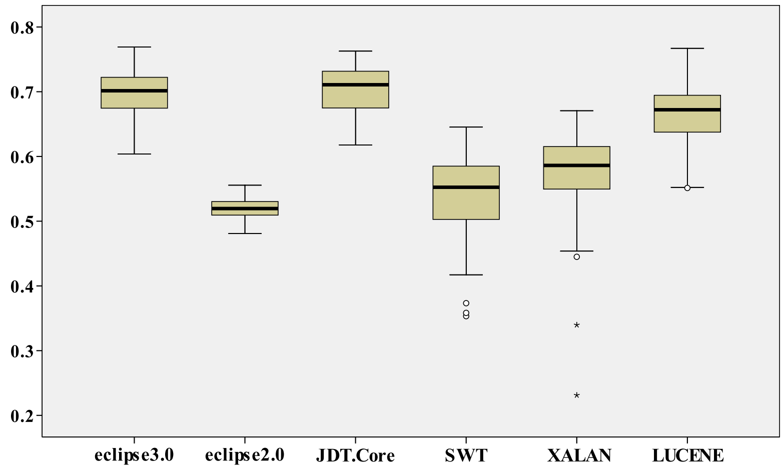

Fig. 1 The box-polts of the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using Decision Tree

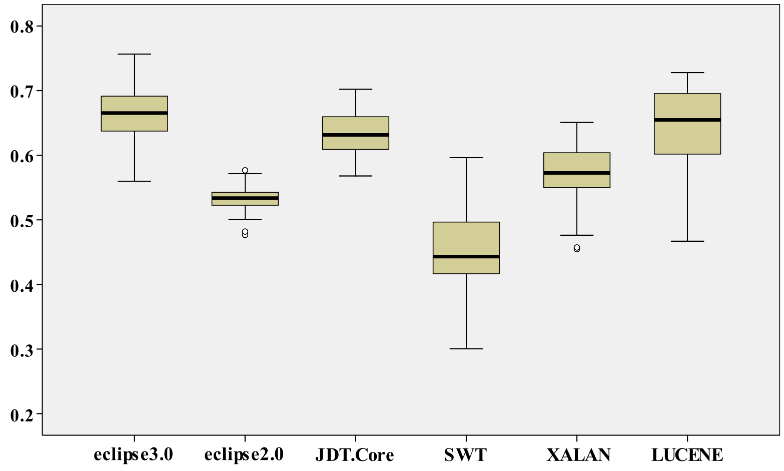

Fig. 2 The box-polts of the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using Logistic Regression

from 5% to 50% only increases the F-measure by 0.09 (from 0.63 to 0.72) using Logistic Regression, by 0.04 using Decision Tree (from 0.69 to 0.73). For Naive Bayes, the F-measure even decreases from 0.70 to 0.62 when sample size is increased.

Figures 1, 2 and 3 show the box-plots for the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using the three classifiers. For all projects, the interquartiles (ranges between upper quartile and lower quartile, i.e., the ranges between the 75th percentile point and the 25th percentile point) are all narrow (less than 0.1). The ranges between upper tails and lower tails of the box-plots are less than 0.25. There are only a few data points fall below the lower tail.

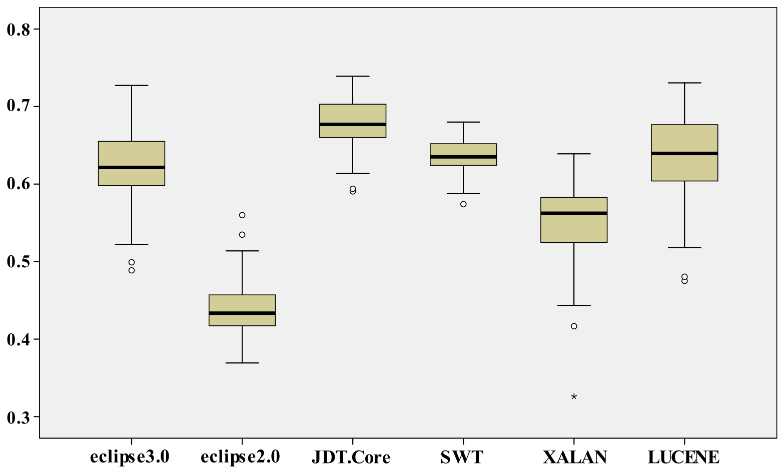

Fig. 3 The box-polts of the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using Naive Bayes

In summary, our results suggest that a smaller sample can achieve similar defect prediction performance as larger samples do. Building a defect prediction model with a small sample does not necessarily lead to significant performance improvement. However, the savings of cost resulted from data reduction is significant.

3.3 Method 2: Sampling with CoForest

We also perform experiments to evaluate the effectiveness of the defect prediction models constructed using random sampling and semi-supervised learning. Following the same experimental setting described in the previous section, for each data set, we use ten different sampling rates $\{5\%, 10\%, 15\%, \ldots, 50\%\}$. The labeled data set is used to initiate CoForest and the unlabeled data is leveraged for classifier refinement. For each of the 60 experimental settings (6 data sets $\times$ 10 sampling rates), we repeat the sampling process for 100 times and the average results are reported in Tables 8 to 13 in the Appendix. We also plot the performance of CoForest and the compared methods versus the sampling rate μ in Fig. 4.

We conduct Mann-Whitney U-test on each setting, and summarize the results in Table 4, where each table element (i, j) denote the "win/tie/loss" (win: the number of times that the i-th method performs significantly better than the j-th method; tie: the number of times that the performances of the i-th method the j-th method have no significantly difference; loss: the number of times that the i-th method performs significantly worse than the j-th method). We conduct sign test (Gibbons 1985) over the "win/tie/loss" values. The element (i, j) is boldfaced if the sign test over the corresponding "win/tie/loss" values suggests the i-th method is significantly better than the j-th method. The results show that CoForest is significantly better than the three conventional machine learning methods. Among the 60 different experimental settings, CoForest significantly outperforms Logistic Regression on 55 settings, Naive Bayes on 49 settings and Decision Tree on 59 settings.

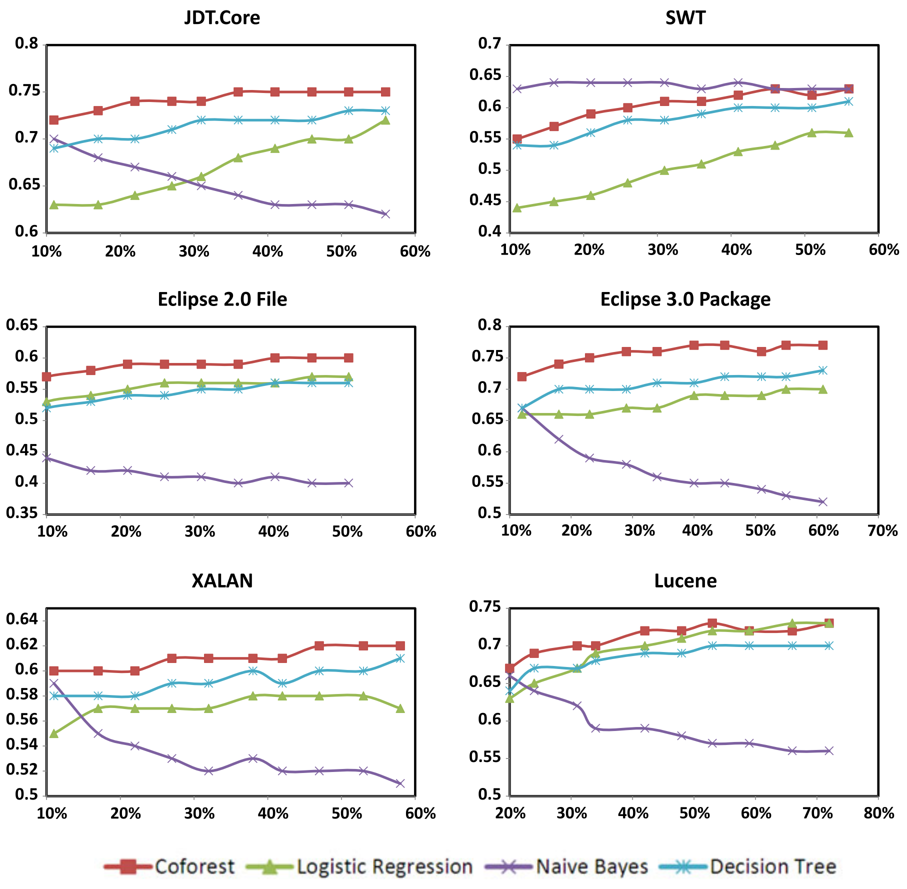

Fig. 4 The F-measures of the CoForest and the compared methods at different sampling rates

Table 4 The summary of Mann-Whitney U-test of compared methods over 60 different experimental settings (6 data sets × 10 sampling rates)

	Logistic Regression	Naive Bayes	Decision Tree
Logistic Regression	–	–	–
Naive Bayes	16/2/42	–	–
Decision Tree	**38/7/15**	**46/4/10**	–
CoForest	**55/5/0**	**49/3/8**	**59/1/0**

For better illustration, in Table 5, we summarize the performance in terms of F-measure and Balance-measure of CoForest as well as the three conventional machine learning methods over all the data set when only 10% modules are sampled, where the best performance in terms of F-measure and Balance-measure on each data set is boldfaced. It can be easily observed from the table that CoForest achieves the

Table 5 Performance of CoForest and the compared methods in predicting defects when only 10% modules are sampled

Dataset	CoForest		Logistic Regression		Naive Bayes		Decision Tree	
	F	B	F	B	F	B	F	B
JDT.Core	**.730**	**.704**	.630	.613	.680	.686	.700	.685
SWT	.570	.674	.450	.600	**.640**	**.759**	.540	.648
ECLIPSE 2.0	**.570**	**.639**	.530	.606	.440	.523	.520	.595
ECLIPSE 3.0	**.740**	**.591**	.660	.568	.620	.615	.700	.590
XALAN	**.600**	**.644**	.570	.624	.550	.599	.580	.628
LUCENE	**.690**	**.582**	.650	.587	.640	.634	.670	.586
Avg.	**.650**	**.639**	.582	.600	.595	.636	.618	.622

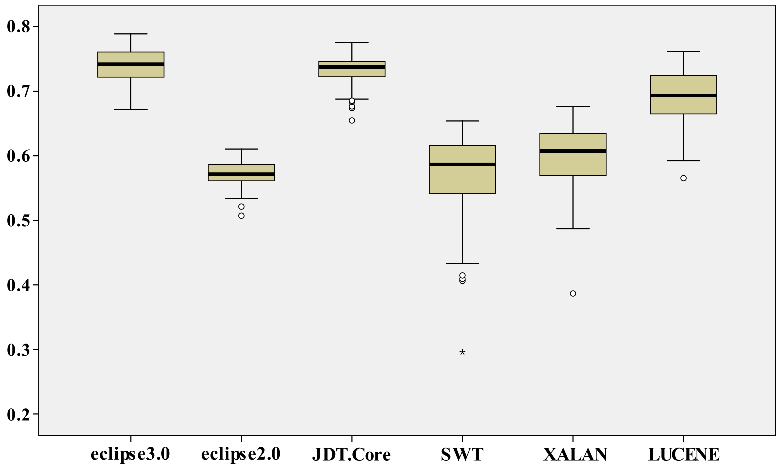

Fig. 5 The box-polts of the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using CoForest

best performance among the compared methods except that on *SWT* NaiveBayes performs the best. On average, CoForest also achieves the best average performance. For example, the average F-measure of CoForest is 0.650, which improves the average F-measure of Logistic Regression (0.582) by 12.1%, Naive Bayes (0.595) by 9.2%, Decision Tree (0.618) by 5.2%. Similar trends are also observed in terms of Balance-measure. These improvements are achieved because CoForest is able to utilize the extra unlabeled data to improve prediction performance while the conventional machine learning methods cannot.

It can be observed from Fig. 4 that for CoForest increasing sample size may only result in marginal performance improvement. For example, for *JDT.Core* when increasing sample size from 5% to 20%, the F-measure is only improved by 2%. When the sample size is further increased to 50%, the F-measure is essentially not changed.

Figure 5 shows the box-plots for the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using CoForest. For all projects, the interquartiles are all narrow (less than 0.07). The ranges between upper tails and lower tails of the box-plots are also narrow (less than 0.20). There are only a few data points fall below the lower tail. These results show that for large samples, the CoForest learner can achieve good defect prediction accuracy with high confidence. For the *SWT* project, the variations among different trails are larger (as reflected by the wider ranges in box-plots). However, the lower quartile is 0.54, therefore among the 100 trails, 75 trails have F-measures higher than 0.54.

In summary, our results confirm that the proposed semi-supervised learner CoForest can improve the performance of the sample-based defect prediction with high confidence.

3.4 Method 3: Sampling with ACoForest

Following the same experimental design described in the previous two sections, we also evaluate the sample-based defect prediction method with active learning techniques. We use ACoForest to learn a classifier for defect prediction. We use ten different sampling rates $\{5\%, 10\%, 15\%, \ldots, 50\%\}$ to generate initial labeled training data. The average precision, recall, F-measure and Balance-measure at different sampling rates are tabulated in Table 14 to 19 in the Appendix. Note that, since ACoForest can actively select a number of informative unlabeled data for extensive software testing, it actually uses more labeled data than the original sample during learning process. To achieve fair comparisons, for other methods which cannot actively select modules for extensive testing, we feed them with the same number of labeled data (i.e., the number of initially sampled plus the number of actively queried) used for training ACoForest. To explicitly show how many labeled data are used during the training process, we tabulated the average percentage of labeled data in the braces w.r.t. the corresponding sampling rate. Note that the only difference of the labeled data used by ACoForest and other methods is that besides the initially sampled modules, ACoForest actively select some informative modules for extensive testing while other methods randomly select the same number of modules for testing. Similarly, we plot the performance of ACoForest and the compared methods versus the sampling rate μ in Fig. 6 for better illustration.

We also conduct Mann-Whitney U-test on each setting, and summarize the results in Table 6, where each table element (i, j) denotes the number of "win/tie/loss". We conduct sign test (Gibbons 1985) over the "win/tie/loss" values. The element (i, j) is boldfaced if the sign test over the corresponding "win/tie/loss" values suggests the i-th method is significantly better than the j-th method. The results show that ACoForest is significantly better than the three conventional machine learning methods. Among the 60 different experimental settings (6 data sets $\times$ 10 sampling rates), ACoForest significantly outperforms Logistic Regression on all 60 settings, Naive Bayes on 59 settings and Decision Tree on 60 settings. Compared to CoForest, ACoForest achieves significantly better performance on 52 settings, and comparable performance on 7 settings, and significantly worse performance on only 1 setting.

Table 7 further summarizes the performance in terms of F-measure and Balance-measure of ACoForest, CoForest and three conventional machine learning methods over all the data set when only 10% modules are initially sampled. The best performance in terms of F-measure and Balance-measure on each data set is boldfaced.

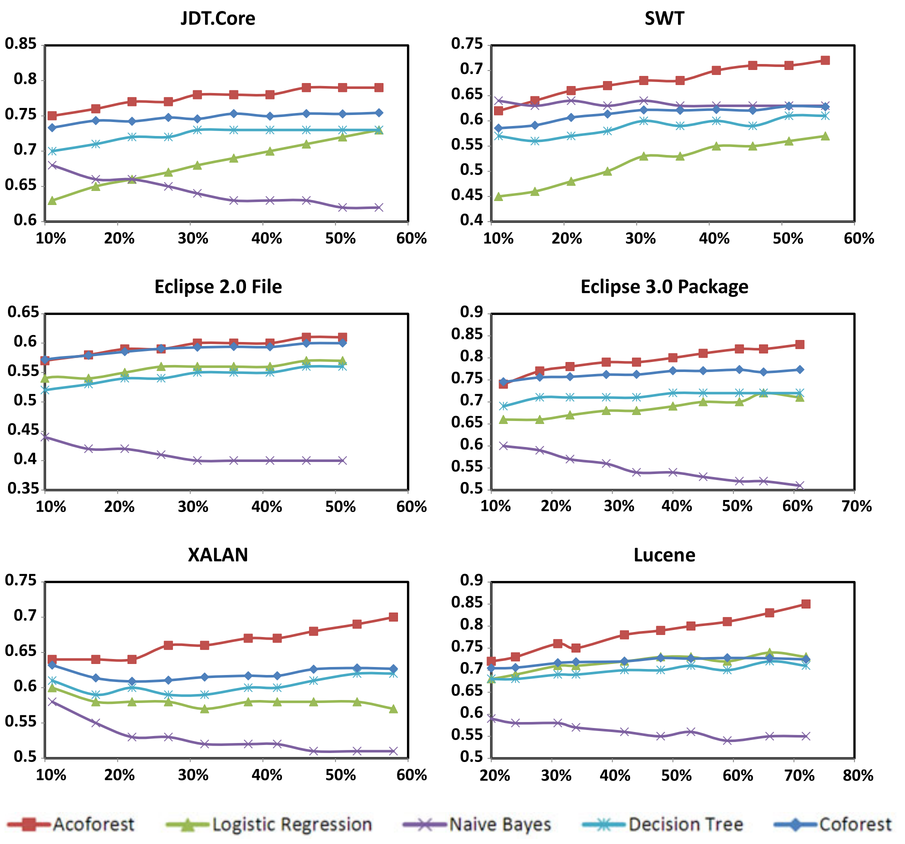

Fig. 6 The F-measures of the ACoForest and the compared methods at different sampling rates

Table 6 The summary of Mann-Whitney U-test of compared methods over 60 experimental settings (6 data sets × 10 sampling rates)

	Logistic Regression	Naive Bayes	Decision Tree	CoForest
Logistic Regression	–	–	–	–
Naive Bayes	12/1/47	–	–	–
Decision Tree	**37/5/18**	**50/0/10**	–	–
CoForest	**50/10/0**	**50/4/6**	**56/4/0**	–
ACoForest	**60/0/0**	**59/1/0**	**60/0/0**	**52/7/1**

It can be easily observed from the table that ACoForest almost always achieves the best performance among the compared methods except that on *ECLIPSE 2.0* CoForest performs the best in terms of F-measure. ACoForest also achieves the best average performance. For example, the average F-measure of ACoForest is 0.685, which improves the average F-measure of CoForest (0.664) by 3.2%, Logistic Regression (0.597) by 14.7%, Naive Bayes (0.595) by 19.1%, Decision Tree (0.628) by

Table 7 Performance of ACoForest and the compared methods in predicting defects when only 10% modules are initially sampled

Dataset	ACoForest		CoForest		Logistic Regression		Naive Bayes		Decision Tree	
	F	B	F	B	F	B	F	B	F	B
JDT.Core	**.760**	**.744**	.743	.717	.650	.631	.660	.670	.710	.690
SWT	**.640**	**.717**	.591	.687	.460	.607	.630	.741	.560	.653
ECLIPSE 2.0	.570	**.641**	**.572**	.639	.540	.608	.440	.522	.520	.596
ECLIPSE 3.0	**.770**	**.624**	.756	.598	.660	.572	.590	.596	.710	.597
XALAN	**.640**	**.679**	.614	.658	.580	.635	.550	.591	.590	.644
LUCENE	**.730**	**.633**	.706	.606	.690	.633	.580	.603	.680	.602
Avg.	**.685**	**.673**	.664	.651	.597	.614	.575	.620	.628	.630

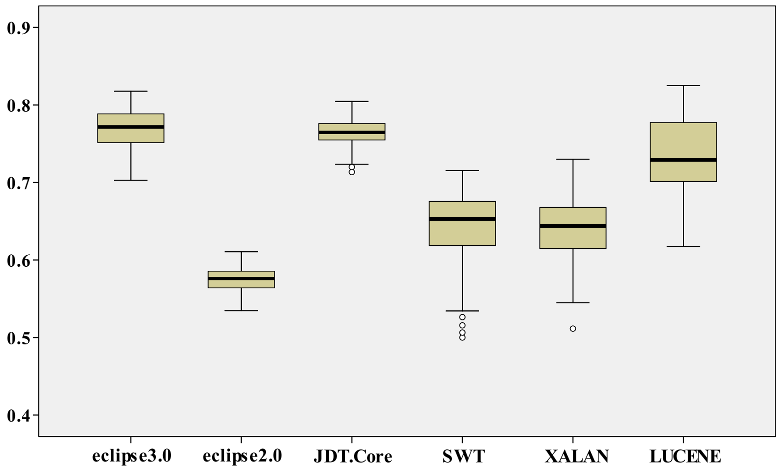

Fig. 7 The box-polts of the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using ACoForest

9.1%. Similar trends are also observed in terms of Balance-measure. The superiority of ACoForest over CoForest and larger improvement of ACoForest than that of CoForst over conventional machine learning methods suggest that besides exploiting extra unlabeled data, actively sampling modules for testing according to the learner's needs can further improve the prediction performance.

Figure 7 shows the box-plots for the F-measures obtained from 100 trails with sampling rate $\mu = 10\%$ using ACoForest. For all projects, the interquartiles are all narrow (less than 0.08). The ranges between upper tails and lower tails of the box-plots are less than 0.20. There are only a few data points fall below the lower tail.

In summary, our results confirm that the proposed active semi-supervised learning method ACoForest can achieve better defect prediction performance with high confidence.

4 Discussions

4.1 Applying the proposed methods

Our experiments show that a smaller sample can achieve similar defect prediction performance as larger samples do. The sample can serve as an initial labeled training set that represent the underlying data distribution of the entire dataset. Thus if there is no sufficient historical datasets for building an effective defect prediction model for a new project, we can randomly sample a small percentage of modules to test, obtain their defect status (defective or non-defective), and then use the collected sample to build a defect prediction for this project.

Our experiments also show that in general, sampling with semi-supervised learning and active learning can achieve better prediction performance than sampling with conventional machine learning techniques. A sample may contain much information that a conventional machine learner has already learned well but may contain little information that the learner needs for improving the current prediction accuracy. The proposed CoForest and ACoForest learners take the needs for learning into account and obtain information needed for improving performance from the un-sampled modules.

Both CoForest and ACoForest methods work well for sample-based defect prediction. ACoForest also supports the active selection of the modules—it can actively suggest the QA team which modules to be chosen in order to increase the prediction performance. Thus in order to apply ACoForest, interactions with test engineers are required. If such interactions is allowed (which implies that more time and efforts are allowed), we can apply the ACoForest method. If such interaction is not allowed due to limited time and resources, we can apply the CoForest method.

4.2 Threats to validity

In our approach, we draw a random sample from the population of modules. To ensure proper statistical inference and to ensure the cost-effectiveness of the proposed method, the population size should be large enough. Therefore, the proposed method is suitable for large-scale software systems.

The simple random sampling method requires that each individual in a sample to be collected entirely by chance with the same probability. Selection bias may be introduced if the module sample is collected simply by convenience, or from a single developer/team. The selection bias can lead to non-sampling errors (errors caused by human rather than sampling) and should be avoided.

The defect data for a sample can be collected through quality assurance activities such as software testing, static program checking and code inspection. As the sample will be used for prediction, these activities should be carefully carried out so that most of defects can be discovered. Incorrect sample data may lead to incorrect estimates of the population.

In our experiments, we used the public defect dataset available at the PROMISE dataset. Although this dataset has been used by many other studies (Koru and Liu 2005; Menzies et al. 2007; Zhang et al. 2010; Zimmermann et al. 2007, 2009), our results may be under threat if the dataset is seriously flawed (e.g., there were major problems in bug data collection and recording). Also, all the data used are collected from open source projects. It is desirable to replicate the experiments on industrial, in-house developed projects to further evaluate their validity. This will be our important future work.

5 Related work

5.1 Software defect prediction

The ability to predict software quality is important for software quality improvement and project management. Over the years, many prediction models have been proposed to predict defect-proneness of software modules (Menzies et al. 2007; Zhang 2009; Zhang and Wu 2010; Zimmermann et al. 2007, 2009). For example, Menzies et al. (2007) performed defect predictions for five NASA projects using static code metrics. In their work, probability of detection (recall) and probability of false alarm (pf) are used to measure the accuracy of a defect prediction model. Their models generate the average results of recall = 71% and pf = 25%, using a Naive Bayes classifier. Zhang and Zhang (2007) pointed out that Menzies et al.'s results were not satisfactory when precision was considered. They found that high recall and low pf do not necessarily lead to high precision. Kim et al. (2007) predicted defects by caching locations that were adjusted by software change history data. Their method could predict 73%–95% faults based on 10% files. Zimmermann and Nagappan (2008) proposed models based on the structural measurement of dependency graphs. Hassan (2009) found that the more complex the changes to a file, the higher the chance the file would contain faults. Lessmann et al. (2008) reported an extensive study on the statistical differences between 19 data miners commonly used for defect prediction and they found that the learners' performance was remarkably similar. In our prior work, we also proposed code complexity-based defect-prediction models (Zhang et al. 2007; Zhang 2009). Recently, Jiang et al. (2011) proposed ROCUS algorithm that is able to exploit unlabeled data when the distribution is imbalanced. They also applied it to defect prediction.

If no good historical data is available, can we use data from other projects or even other companies to build effective prediction models? Recently Zimmermann et al. (2009) ran 622 cross-project predictions for 12 real-world applications, their results show that cross-project prediction is a serious challenge, i.e., simply using models from projects in the same domain or with the same process does not lead to accurate predictions. Turhan et al. (2009) also investigated the applicability of cross-company (CC) data for building localized defect predictors. They used data from NASA to predict defect-proneness of modules developed by a Turkish software company. They applied nearest neighbor (NN) filtering method to CC data, and observed that defect predictors learned from WC (within-company) data outperform the ones learned from CC data. They also found the CC predictions resulted in increased number of false alarms. All these results show that data from other projects or companies cannot be easily used for local projects without advanced research. Therefore, if no good historical data is available, effective defect prediction would be difficult. In this paper, we propose sample-based defect prediction methods to address the problem. Our idea of building sample-based defect prediction models was initially presented in Zhang and Wu (2010).

In this study we show that a small sample from a local project can build cost-effective defect prediction models. Compared to the related work, our method has the following advantages:

– Does not require historical or other projects' data
– Requires only a small sample of modules
– Can achieved improved prediction performance by learning from unsampled modules.

The proposed sample-based defect prediction approach is independent of the attributes used for model construction. In this study, we only use static code attributes to build the model. As suggested in Hassan (2009), Zimmermann and Nagappan (2008), using more types of attributes, such as changes and dependency metrics, could improve the accuracy of the prediction. We will investigate more types of attributes in our future work.

5.2 Semi-supervised learning and active learning

In many practical applications, many unlabeled data can be easily collected, while only a few labeled data can be obtained since much human effort and expertise is required for labeling. Thus, methods have been developed for exploiting the available unlabeled data for performance improvement. *Semi-supervised learning and* active learning are two major techniques.

Semi-supervised learning (Chapelle et al. 2006; Zhu 2006) aims to construct a learner that automatically exploits the large amount of unlabeled data in addition to the labeled data in order to help improve the learning performance. Semi-supervised learning methods are usually divided into generative-model based methods (Miller and Uyar 1997; Nigam et al. 2000; Shahshahani and Landgrebe 1994), low density separation based methods (Chapelle and Zien 2005; Grandvalet and Bengio 2005; Joachims 1999), graph-based methods (Belkin et al. 2006; Zhou et al. 2004; Zhu et al. 2003), and disagreement-based methods (Zhou and Li 2010).

Disagreement-based semi-supervised learning, which uses multiple learners and exploits the disagreements among the learners during the learning process, originates from co-training proposed by Blum and Mitchell (1998), where classifiers learned from two sufficient and redundant views teach each other using some confidently predicted unlabeled examples. Later, Goldman and Zhou (2000) proposed an extension of co-training which does not require two views but two different special learning algorithms. Zhou and Li (2005) proposed to use three classifiers to exploit unlabeled data, where an unlabeled example is labeled and used to teach one classifier if the other two classifiers agree on its labeling. Later, Li and Zhou (2007) further extended the idea in Zhou and Li (2005) by collaborating more classifiers in training process. Besides classification, Zhou and Li (2007) also adapted the disagreement-based paradigm to semi-supervised regression. Disagreement-based semi-supervised learning paradigm has been widely applied to natural language processing (e.g., Steedman et al. 2003), information retrieval (e.g., Zhou et al. 2006; Li et al. 2009), computer-aided diagnosis (e.g., Li and Zhou 2007), etc.

Active learning deals with methods that assume the learner has some control over the input space. In exploiting unlabeled data, it can query the ground-truth labels of specific examples from an oracle (e.g., a human expert). Here the key is to select appropriate unlabeled examples for query such that the learning performance can be improved with a minimum number of queries. There are two major schemes, i.e. uncertainty sampling and committee-based sampling. Approaches of the former train a single learner and then query the unlabeled examples on which the learner is least confident (Lewis and Catlett 1994; Lewis and Gale 1994; Tong and Koller 2000; Balcan et al. 2007); while approaches of the latter generate a committee of multiple learners and select the unlabeled examples on which the committee members disagree the most (Seung et al. 1992; Dagan and Engelson 1994; Freund et al. 1997).

Since semi-supervised learning and active learning exploit the unlabeled data in different way, they are further combined to achieve better performance (Muslea et al. 2002; Zhou et al. 2006; Xu et al. 2009). Recently, Wang and Zhou (2008) analytically showed that the sample complexity can be exponentially reduced if multi-view active learning and semi-supervised learning are combined.

6 Conclusion

Current techniques for estimating quality, especially for defect prediction, are mainly based on a sufficient amount of historical project data. However, if no historical data is available, effective defect prediction would be difficult. In this paper, we propose to use a small sample of modules to construct cost-effective defect prediction models for large scale systems. We apply a semi-supervised learning method called CoForest to build a classification model based on a sample and the remaining un-sampled modules. Such a model is then used to predict defect-proneness of an un-sampled module in the project. We also propose a novel active semi-supervised learning method called ACoForest, which can actively select several informative un-sampled modules for testing while automatically exploiting the remaining un-sampled modules for better performance. Our experimental results on PROMISE datasets show that the proposed methods can achieve better performance than conventional machine learners, after exploiting the un-sampled modules. We also notice that sample size does not affect the prediction results significantly.

In the future, we plan to continue to apply our sample-based quality estimation methods to industrial practices and evaluate their effectiveness. We will also develop an integrated environment that can automate the entire defect prediction process.

Acknowledgements This research is supported by National Science Foundation of China (60903103, 61073006, 61021062) and the Tsinghua University research project 2010THZ0. The authors would like to thank anonymous reviewers for their suggestions and Wei-Wei Tu for proof-reading.

Appendix

Table 8 Performance of CoForest and the compared methods in predicting defects on *JDT.Core*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.745	.701	.720	.678	.630	.635	.630	.603	.679	.730	.700	.692	.701	.696	.690	.665
10%	.743	.727	.730	.704	.623	.648	.630	.613	.608	.774	.680	.686	.696	.717	.700	.685
15%	.735	.742	.740	.715	.621	.660	.640	.623	.583	.789	.670	.677	.699	.715	.700	.684
20%	.747	.741	.740	.718	.637	.673	.650	.636	.563	.809	.660	.672	.714	.717	.710	.689
25%	.743	.749	.740	.723	.644	.682	.660	.644	.541	.820	.650	.660	.708	.729	.720	.697
30%	.751	.751	.750	.726	.665	.690	.680	.655	.527	.822	.640	.652	.722	.727	.720	.698
35%	.749	.750	.750	.726	.683	.695	.690	.664	.514	.830	.630	.645	.716	.724	.720	.695
40%	.748	.756	.750	.730	.687	.706	.700	.673	.506	.836	.630	.641	.719	.732	.720	.702
45%	.752	.751	.750	.728	.703	.709	.700	.681	.504	.838	.630	.640	.727	.728	.730	.703
50%	.752	.750	.750	.727	.715	.719	.720	.693	.497	.844	.620	.636	.726	.730	.730	.704

*Mann-Whitney U-test shows that CoForest performs better than other methods, at significant level 0.05

Table 9 Performance of CoForest and the compared methods in predicting defects on *SWT*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.525	.602	.550	.655	.447	.452	.440	.590	.720	.565	.630	.767	.527	.582	.540	.654
10%	.555	.612	.570	.674	.471	.444	.450	.600	.696	.595	.640	.759	.518	.590	.540	.648
15%	.572	.626	.590	.687	.482	.451	.460	.608	.690	.606	.640	.755	.535	.607	.560	.660
20%	.584	.634	.600	.694	.489	.479	.480	.612	.667	.617	.640	.740	.546	.621	.580	.668
25%	.588	.639	.610	.697	.512	.496	.500	.628	.666	.620	.640	.743	.540	.629	.580	.665
30%	.593	.645	.610	.700	.518	.507	.510	.635	.644	.625	.630	.728	.546	.647	.590	.669
35%	.598	.654	.620	.703	.539	.527	.530	.651	.651	.632	.640	.732	.557	.645	.600	.676
40%	.606	.654	.630	.709	.554	.533	.540	.659	.644	.626	.630	.729	.569	.642	.600	.684
45%	.594	.655	.620	.699	.560	.554	.560	.663	.632	.632	.630	.720	.551	.652	.600	.671
50%	.610	.651	.630	.711	.577	.550	.560	.676	.640	.627	.630	.726	.569	.654	.610	.685

*Mann-Whitney U-test shows that CoForest performs better than other methods (except for Naive Bayes), at significant level 0.05

Table 10 Performance of CoForest and the compared methods in predicting defects on *ECLIPSE 2.0*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.553	.556	.550	.624	.504	.491	.500	.576	.362	.604	.450	.536	.485	.525	.500	.583
10%	.579	.569	.570	.639	.504	.567	.530	.606	.340	.619	.440	.523	.509	.532	.520	.595
15%	.576	.587	.580	.644	.496	.602	.540	.612	.315	.628	.420	.508	.515	.544	.530	.601
20%	.579	.595	.590	.648	.496	.620	.550	.615	.310	.631	.420	.505	.522	.550	.540	.606
25%	.585	.598	.590	.651	.496	.633	.560	.618	.305	.636	.410	.501	.522	.554	.540	.606
30%	.583	.606	.590	.653	.495	.637	.560	.618	.297	.638	.410	.496	.533	.562	.550	.614
35%	.578	.615	.590	.653	.495	.645	.560	.619	.293	.643	.400	.494	.536	.568	.550	.617
40%	.587	.615	.600	.657	.500	.649	.560	.623	.296	.643	.410	.496	.543	.569	.560	.620
45%	.578	.620	.600	.655	.502	.651	.570	.624	.295	.643	.400	.495	.546	.573	.560	.622
50%	.573	.626	.600	.654	.502	.655	.570	.625	.292	.642	.400	.493	.547	.577	.560	.624

*Mann-Whitney U-test shows that CoForest performs better than other methods, at significant level 0.05

Table 11 Performance of CoForest and the compared methods in predicting defects on *ECLIPSE 3.0 - PACKAGE*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.761	.695	.720	.559	.658	.680	.660	.556	.624	.733	.670	.614	.672	.686	.670	.562
10%	.776	.712	.740	.591	.646	.687	.660	.568	.529	.769	.620	.615	.696	.705	.700	.590
15%	.794	.720	.750	.599	.632	.685	.660	.566	.479	.788	.590	.600	.697	.707	.700	.593
20%	.806	.722	.760	.602	.647	.691	.670	.574	.454	.801	.580	.590	.697	.710	.700	.596
25%	.806	.727	.760	.611	.651	.703	.670	.589	.431	.805	.560	.578	.707	.715	.710	.604
30%	.805	.731	.770	.620	.665	.710	.690	.599	.414	.814	.550	.570	.708	.715	.710	.604
35%	.818	.732	.770	.618	.662	.713	.690	.601	.411	.819	.550	.569	.721	.720	.720	.610
40%	.804	.732	.760	.627	.679	.713	.690	.607	.403	.817	.540	.564	.725	.714	.720	.607
45%	.808	.734	.770	.631	.686	.719	.700	.615	.394	.825	.530	.560	.722	.721	.720	.618
50%	.808	.735	.770	.631	.683	.723	.700	.618	.382	.824	.520	.552	.731	.722	.730	.618

*Mann-Whitney U-test shows that CoForest performs better than other methods, at significant level 0.05

Table 12 Performance of CoForest and the compared methods in predicting defects on *XALAN*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.617	.610	.600	.642	.561	.554	.550	.594	.501	.730	.590	.630	.575	.600	.580	.622
10%	.592	.618	.600	.644	.548	.610	.570	.624	.447	.740	.550	.599	.559	.618	.580	.628
15%	.590	.623	.600	.649	.531	.630	.570	.626	.425	.748	.540	.585	.557	.616	.580	.631
20%	.593	.631	.610	.652	.515	.645	.570	.622	.411	.757	.530	.577	.558	.632	.590	.637
25%	.591	.628	.610	.651	.515	.640	.570	.621	.405	.748	.520	.572	.557	.632	.590	.638
30%	.603	.630	.610	.656	.519	.656	.580	.627	.409	.756	.530	.576	.578	.629	.600	.644
35%	.593	.638	.610	.655	.519	.656	.580	.627	.399	.757	.520	.569	.561	.639	.590	.640
40%	.599	.638	.620	.658	.513	.669	.580	.627	.395	.766	.520	.567	.576	.646	.600	.650
45%	.606	.635	.620	.659	.512	.673	.580	.627	.393	.762	.520	.565	.578	.641	.600	.649
50%	.605	.638	.620	.660	.505	.666	.570	.621	.386	.774	.510	.561	.588	.641	.610	.653

*Mann-Whitney *U*-test shows that CoForest performs better than other methods, at significant level 0.05

Table 13 Performance of CoForest and the compared methods in predicting defects on *LUCENE*

μ	CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%	.682	.675	.670	.579	.616	.673	.630	.578	.641	.712	.660	.615	.644	.666	.640	.568
10%	.718	.675	.690	.582	.627	.675	.650	.587	.555	.764	.640	.634	.666	.677	.670	.586
15%	.731	.689	.700	.601	.663	.686	.670	.603	.512	.793	.620	.627	.679	.682	.670	.593
20%	.729	.688	.700	.605	.680	.702	.690	.623	.481	.803	.590	.611	.677	.690	.680	.605
25%	.754	.694	.720	.616	.704	.710	.700	.637	.478	.810	.590	.612	.703	.690	.690	.610
30%	.743	.697	.720	.621	.710	.722	.710	.651	.450	.822	.580	.598	.683	.697	.690	.620
35%	.756	.705	.730	.628	.713	.728	.720	.656	.445	.831	.570	.596	.705	.700	.700	.622
40%	.744	.701	.720	.628	.721	.727	.720	.659	.438	.826	.570	.591	.706	.701	.700	.628
45%	.739	.707	.720	.633	.720	.735	.730	.666	.422	.835	.560	.582	.691	.707	.700	.630
50%	.744	.715	.730	.642	.716	.740	.730	.671	.422	.842	.560	.583	.697	.713	.700	.638

*Mann-Whitney *U*-test shows that CoForest performs better than other methods, at significant level 0.05

Table 14 Performance of ACoForest and the compared methods in predicting defects on *JDT.CORE*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(11%)	.769	.735	.750	.714	.742	.730	.733	.707	.619	.649	.630	.615	.606	.776	.680	.685	.699	.704	.700	.675
10%(17%)	.767	.766	.760	.744	.754	.737	.743	.717	.634	.665	.650	.631	.565	.797	.660	.670	.717	.714	.710	.690
15%(22%)	.769	.775	.770	.751	.737	.751	.742	.723	.640	.677	.660	.640	.553	.810	.660	.666	.716	.723	.720	.696
20%(27%)	.774	.774	.770	.753	.749	.749	.748	.725	.655	.685	.670	.650	.540	.825	.650	.661	.719	.724	.720	.697
25%(31%)	.776	.781	.780	.758	.743	.751	.746	.724	.667	.694	.680	.659	.526	.831	.640	.653	.720	.734	.730	.705
30%(36%)	.781	.785	.780	.763	.755	.753	.753	.730	.685	.698	.690	.668	.516	.828	.630	.646	.726	.728	.730	.701
35%(41%)	.779	.790	.780	.766	.748	.753	.749	.728	.697	.706	.700	.676	.505	.839	.630	.640	.723	.729	.730	.701
40%(46%)	.785	.790	.790	.768	.751	.757	.753	.732	.698	.715	.710	.684	.500	.844	.630	.638	.729	.729	.730	.702
45%(51%)	.788	.790	.790	.770	.754	.753	.753	.730	.711	.721	.720	.693	.493	.842	.620	.633	.724	.732	.730	.704
50%(56%)	.795	.795	.790	.776	.753	.757	.754	.734	.719	.735	.730	.705	.490	.849	.620	.632	.727	.739	.730	.708

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

Table 15 Performance of ACoForest and the compared methods in predicting defects on *SWT*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(11%)	.580	.687	.620	.696	.566	.623	.585	.684	.464	.448	.450	.600	.696	.599	.640	.762	.537	.615	.570	.664
10%(16%)	.610	.693	.640	.717	.573	.623	.591	.687	.476	.458	.460	.607	.660	.617	.630	.741	.521	.617	.560	.653
15%(21%)	.631	.704	.660	.731	.587	.636	.607	.697	.492	.475	.480	.618	.657	.621	.640	.739	.530	.621	.570	.659
20%(26%)	.636	.708	.670	.734	.593	.644	.614	.701	.511	.499	.500	.632	.639	.626	.630	.728	.546	.632	.580	.669
25%(31%)	.650	.717	.680	.745	.604	.647	.622	.707	.534	.521	.530	.647	.645	.631	.640	.731	.561	.641	.600	.679
30%(36%)	.650	.729	.680	.744	.595	.657	.621	.702	.537	.527	.530	.649	.628	.637	.630	.721	.539	.648	.590	.665
35%(41%)	.664	.738	.700	.754	.605	.646	.623	.707	.563	.540	.550	.667	.633	.636	.630	.723	.562	.651	.600	.680
40%(46%)	.677	.741	.710	.764	.602	.646	.621	.705	.563	.546	.550	.667	.629	.634	.630	.720	.554	.642	.590	.674
45%(51%)	.672	.757	.710	.759	.601	.667	.629	.706	.565	.567	.560	.670	.620	.639	.630	.715	.568	.665	.610	.684
50%(56%)	.691	.759	.720	.774	.614	.648	.628	.712	.587	.554	.570	.681	.627	.631	.630	.718	.568	.654	.610	.684

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

Table 16 Performance of ACoForest and the compared methods in predicting defects on *ECLIPSE 2.0*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(6%)	.559	.560	.560	.627	.554	.557	.552	.625	.503	.500	.500	.580	.356	.607	.450	.532	.495	.524	.510	.588
10%(10%)	.582	.570	.570	.641	.577	.570	.572	.639	.504	.571	.540	.608	.338	.620	.440	.522	.513	.531	.520	.596
15%(16%)	.580	.590	.580	.647	.575	.587	.579	.643	.497	.605	.540	.613	.313	.629	.420	.506	.517	.543	.530	.602
20%(21%)	.584	.597	.590	.651	.579	.596	.585	.648	.496	.621	.550	.616	.310	.633	.420	.505	.526	.551	.540	.608
25%(26%)	.587	.603	.590	.653	.585	.599	.590	.651	.496	.633	.560	.618	.304	.637	.410	.501	.525	.556	.540	.608
30%(31%)	.590	.610	.600	.657	.583	.606	.593	.653	.495	.637	.560	.618	.297	.639	.400	.496	.533	.562	.550	.614
35%(36%)	.582	.619	.600	.657	.577	.615	.594	.653	.495	.646	.560	.619	.293	.644	.400	.494	.534	.569	.550	.616
40%(41%)	.584	.625	.600	.659	.571	.620	.593	.651	.500	.649	.560	.623	.295	.643	.400	.495	.539	.571	.550	.619
45%(46%)	.586	.632	.610	.663	.583	.619	.600	.657	.501	.652	.570	.624	.294	.643	.400	.494	.546	.571	.560	.622
50%(51%)	.589	.637	.610	.665	.579	.625	.600	.656	.502	.655	.570	.625	.290	.643	.400	.492	.548	.580	.560	.625

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

Table 17 Performance of ACoForest and the compared methods in predicting defects on *ECLIPSE 3.0—PACKAGE*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(12%)	.796	.711	.740	.581	.783	.717	.745	.595	.629	.690	.660	.572	.496	.787	.600	.607	.680	.702	.690	.584
10%(18%)	.809	.735	.770	.624	.801	.719	.756	.598	.643	.690	.660	.572	.468	.794	.590	.596	.705	.711	.710	.597
15%(23%)	.826	.744	.780	.634	.798	.723	.757	.606	.645	.697	.670	.581	.446	.802	.570	.586	.700	.715	.710	.605
20%(29%)	.835	.752	.790	.649	.804	.727	.762	.614	.657	.700	.680	.586	.427	.807	.560	.577	.709	.717	.710	.609
25%(34%)	.838	.757	.790	.656	.797	.732	.762	.623	.664	.709	.680	.598	.410	.813	.540	.568	.714	.717	.710	.608
30%(40%)	.839	.768	.800	.675	.812	.734	.770	.625	.677	.715	.690	.605	.401	.817	.540	.563	.717	.719	.720	.611
35%(45%)	.853	.774	.810	.683	.809	.738	.770	.629	.674	.721	.700	.612	.393	.823	.530	.559	.720	.722	.720	.613
40%(51%)	.855	.782	.820	.701	.812	.739	.773	.640	.689	.722	.700	.621	.387	.822	.520	.555	.725	.719	.720	.616
45%(55%)	.856	.785	.820	.705	.798	.742	.767	.640	.704	.732	.720	.629	.379	.830	.520	.551	.717	.722	.720	.616
50%(61%)	.868	.800	.830	.727	.811	.741	.773	.640	.699	.732	.710	.631	.368	.829	.510	.544	.725	.721	.720	.617

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

Table 18 Performance of ACoForest and the compared methods in predicting defects on *XALAN*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(11%)	.642	.656	.640	.676	.632	.641	.632	.668	.580	.641	.600	.648	.464	.769	.580	.612	.596	.633	.610	.650
10%(17%)	.621	.668	.640	.679	.596	.643	.614	.658	.539	.649	.580	.635	.431	.765	.550	.591	.561	.649	.590	.644
15%(22%)	.621	.678	.640	.684	.592	.633	.609	.654	.527	.653	.580	.631	.414	.755	.530	.579	.573	.643	.600	.648
20%(27%)	.631	.688	.660	.693	.590	.639	.611	.655	.515	.663	.580	.627	.403	.769	.530	.572	.563	.641	.590	.643
25%(32%)	.636	.685	.660	.696	.600	.636	.615	.659	.515	.657	.570	.626	.395	.755	.520	.566	.567	.639	.590	.645
30%(38%)	.652	.695	.670	.706	.602	.637	.617	.659	.521	.667	.580	.631	.398	.766	.520	.569	.578	.639	.600	.648
35%(42%)	.651	.701	.670	.708	.600	.639	.617	.659	.519	.666	.580	.629	.395	.765	.520	.567	.576	.645	.600	.651
40%(47%)	.664	.703	.680	.714	.611	.646	.626	.667	.515	.676	.580	.630	.388	.768	.510	.562	.583	.644	.610	.653
45%(53%)	.673	.713	.690	.722	.613	.647	.628	.667	.511	.684	.580	.629	.386	.764	.510	.561	.587	.652	.620	.658
50%(58%)	.675	.723	.700	.727	.612	.646	.627	.665	.503	.674	.570	.622	.383	.775	.510	.559	.591	.652	.620	.658

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

Table 19 Performance of ACoForest and the compared methods in predicting defects on *LUCENE*

μ	ACoForest				CoForest				Logistic Regression				Naive Bayes				Decision Tree			
	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B	R	P	F	B
5%(20%)	.736	.716	.720	.632	.710	.707	.704	.622	.661	.713	.680	.629	.484	.802	.590	.611	.677	.697	.680	.607
10%(24%)	.763	.715	.730	.633	.731	.689	.706	.606	.685	.711	.690	.633	.461	.818	.580	.603	.678	.687	.680	.602
15%(31%)	.788	.739	.760	.669	.740	.700	.716	.621	.711	.721	.710	.647	.461	.812	.580	.602	.706	.689	.690	.604
20%(34%)	.784	.729	.750	.659	.743	.701	.718	.624	.709	.720	.710	.649	.448	.822	.570	.596	.699	.697	.690	.618
25%(42%)	.811	.751	.780	.690	.755	.692	.720	.620	.720	.720	.720	.654	.427	.825	.560	.584	.706	.702	.700	.631
30%(48%)	.820	.772	.790	.718	.744	.717	.728	.644	.722	.745	.730	.674	.415	.843	.550	.578	.688	.718	.700	.640
35%(53%)	.829	.778	.800	.724	.750	.708	.726	.635	.724	.741	.730	.674	.425	.847	.560	.585	.705	.712	.710	.637
40%(59%)	.837	.792	.810	.746	.748	.713	.728	.644	.714	.740	.720	.674	.408	.836	.540	.573	.702	.707	.700	.635
45%(66%)	.852	.814	.830	.772	.740	.720	.727	.649	.723	.753	.740	.684	.412	.842	.550	.576	.715	.725	.720	.651
50%(72%)	.860	.836	.850	.799	.732	.722	.724	.650	.720	.754	.730	.687	.407	.848	.550	.574	.704	.721	.710	.647

*Mann-Whitney U-test shows that ACoForest performs better than other methods, at significant level 0.05

References

Angluin, D., Laird, P.: Learning from noisy examples. Mach. Learn. **2**(4), 343–370 (1988)

Balcan, M.F., Broder, A.Z., Zhang, T.: Margin based active learning. In: Proceedings of the 20th Annual Conference on Learning Theory, San Diego, CA, pp. 35–50 (2007)

Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res. **7**(11), 2399–2434 (2006)

Blum, A., Mitchell, T.: Combining labeled and unlabeled datawith co-training. In: Proceedings of the 11th Annual Conf. on Computational Learning Theory, pp. 92–100 (1998)

Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)

Chapelle, O., Zien, A.: Semi-supervised learning by low density separation. In: Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics, Savannah Hotel, Barbados, pp. 57–64 (2005)

Chapelle, O., Schölkopf, B., Zien, A. (eds.): Semi-Supervised Learning. MIT Press, Cambridge (2006)

Dagan, I., Engelson, S.P.: Committee-based sampling for training probabilistic classifiers. In: Proceedings of the 12th International Conference on Machine Learning, Tahoe City, CA, pp. 150–157 (1994)

Freund, Y.H.S., Seung, E.S., Tishby, N.: Selective sampling using the query by committee algorithm. Mach. Learn. **28**(2–3), 133–168 (1997)

Gibbons, J.D.: Nonparametric Statistical Inference. Marcel Dekker, New York (1985)

Goldman, S., Zhou, Y.: Enhancing supervised learning with unlabeled data. In: Proceedings of the 17th International Conference on Machine Learning, San Francisco, CA, pp. 327–334 (2000)

Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, vol. 17, pp. 529–536. MIT Press, Cambridge (2005)

Hassan, E.: Predicting faults using the complexity of code changes. In: Proceedings of the 31th International Conference on Software Engineering, Vancouver, Canada, pp. 78–88 (2009)

Jiang, Y., Li, M., Zhou, Z.-H.: Software defect detection with ROCUS. J. Comput. Sci. Technol. **26**(2), 328–342 (2011)

Joachims, T.: Transductive inference for text classification using support vector machines. In: Proceedings of the 16th International Conference on Machine Learning, Bled, Slovenia, pp. 200–209 (1999)

Kim, S., Zimmermann, T., Whitehead, E., Zeller, J.A.: Predicting faults from cached history. In: Proceedings of ICSE'07, Minneapolis, USA, pp. 489–498 (2007)

Koru, L.H., 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)

Lewis, D., Gale, W.: A sequential algorithm for training text classifiers. In: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, pp. 3–12 (1994)

Lewis, D.D., Catlett, J.: Heterogeneous uncertainty sampling for supervised learning. In: Proceedings of the 11th International Conference on Machine Learning, New Brunswick, NJ, pp. 148–156 (1994)

Li, M., Zhou, Z.H.: Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. **37**(6), 1088–1098 (2007)

Li, M., Li, H., Zhou, Z.H.: Semi-supervised document retrieval. Inf. Process. Manag. **45**(3), 341–355 (2009)

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

Miller, D.J., Uyar, H.S.: A mixture of experts classifier with learning based on both labelled and unlabelled data. In: Mozer, M., Jordan, M.I., Petsche, T. (eds.) Advances in Neural Information Processing Systems, vol. 9, pp. 571–577. MIT Press, Cambridge (1997)

Muslea, I., Minton, S., Knoblock, C.A.: Active + semi-supervised learning = robust multi-view learning. In: Proceedings of the 19th International Conference on Machine Learning, Sydney, Australia, pp. 435–442 (2002)

Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of ICSE'06, Shanghai, China, pp. 452–461 (2006)

Nigam, K., McCallum, A.K., Thrun, S., Mitchell, T.: Text classification from labeled and unlabeled documents using EM. Mach. Learn. **39**(2–3), 103–134 (2000)

Seung, H., Opper, M., Sompolinsky, H.: Query by committee. In: Proceedings of the 5th ACM Workshop on Computational Learning Theory, Pittsburgh, PA, pp. 287–294 (1992)

Shahshahani, B., Landgrebe, D.: The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon. IEEE Trans. Geosci. Remote Sens. **32**(5), 1087–1095 (1994)

Steedman, M., Osborne, M., Sarkar, A., Clark, S., Hwa, R., Hockenmaier, J., Ruhlen, P., Baker, S., Crim, J.: Bootstrapping statistical parsers from small data sets. In: Proceedings of the 11th Conference on the European Chapter of the Association for Computational Linguistics, Budapest, Hungary, pp. 331–338 (2003)

Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. In: Proceedings of the 17th International Conference on Machine Learning, Stanford, CA, pp. 999–1006 (2000)

Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empir. Softw. Eng. **14**, 540–578 (2009). doi:10.1007/s10515-011-0092-1. See http://portal.acm.org/citation.cfm?id=1612763.1612782

Wang, W., Zhou, Z.H.: On multi-view active learning and the combination with semi-supervised learning. In: Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, pp. 1152–1159 (2008)

Xu, J.M., Fumera, G., Roli, F., Zhou, Z.H.: Training spam assassin with active semi-supervised learning. In: Proceedings of the 6th Conference on Email and Anti-Spam, Mountain View, CA (2009)

Zhang, H.: An investigation of the relationships between lines of code and defects. In: Proceedings of 25th IEEE International Conference on Software Maintenance, Edmonton, Canada, pp. 274–283 (2009)

Zhang, H., Wu, R.: Sampling program quality. In: Proceedings of 26th IEEE International Conference on Software Maintenance, Timisoara, Romania, pp. 1–10 (2010)

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

Zhang, H., Zhang, X., Gu, M.: Predicting defective software components from code complexity measures. In: Proceedings of 13th IEEE Pacific Rim International Symposium on Dependable Computing, Australia, pp. 93–96 (2007)

Zhang, H., Nelson, A., Menzies, T.: On the value of learning from defect dense components for software defect prediction. In: Proceedings of International Conference on Predictor Models in Software Engineering, Timisoara, Romania, p. 14 (2010)

Zhou, D., Bousquet, O., Lal, T.N., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems 16. MIT Press, Cambridge (2004)

Zhou, Z.-H.: When semi-supervised learning meets ensemble learning. In: Proceedings of 8th International Workshop on Multiple Classifier Systems, Reykjavik, Iceland, pp. 529–538 (2009)

Zhou, Z.-H., Li, M.: Tri-training: Exploiting unlabeled data using three classifiers. IEEE Trans. Knowl. Data Eng. **17**(11), 1529–1541 (2005)

Zhou, Z.-H., Li, M.: Semi-supervised regression with co-training style algorithms. IEEE Trans. Knowl. Data Eng. **19**(11), 1479–1493 (2007)

Zhou, Z.-H., Li, M.: Semi-supervised learning by disagreement. Knowl. Inf. Syst. **24**(3), 415–439 (2010)

Zhou, Z.-H., Chen, K.J., Dai, H.B.: Enhancing relevance feedback in image retrieval using unlabeled data. ACM Trans. Inf. Sys. **24**(2), 219–244 (2006)

Zhu, X.: Semi-supervised learning literature survey. Tech. Rep. 1530, Department of Computer Sciences, University of Wisconsin at Madison, Madison, WI (2006). http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf

Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: Proceedings of the 20th International Conference on Machine Learning, Washington, DC, pp. 912–919 (2003)

Zimmermann, T., Nagappan, N.: Predicting defects using network analysis on dependency graphs. In: Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, pp. 531–540 (2008)

Zimmermann, T., Premraj, R., Zeller, A.: Predicting defects for eclipse. In: Proceedings of International Conference on Predictor Models in Software Engineering, Minneapolis, USA (2007)

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B.: Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proceedings of ESEC/FSE 2009, Amsterdam, The Netherlands, pp. 91–100 (2009)