Software Defect Detection with Rocus

Yuan Jiang (姜远), *Member, CCF*
Ming Li (黎铭), *Member, CCF, ACM, IEEE*
Zhi-Hua Zhou (周志华), *Senior Member, CCF, IEEE, Member, ACM*

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
E-mail: {jiangy, lim, zhouzh}@lamda.nju.edu.cn

Received May 15, 2009

Abstract
Software defect detection aims to automatically identify defective software modules for efficient software test in order to improve the quality of a software system. Although many machine learning methods have been successfully applied to the task, most of them fail to consider two practical yet important issues in software defect detection. First, it is rather difficult to collect a large amount of labeled training data to learn a well-performing model; second, a software system usually contains much fewer defective modules than defect-free modules, in which learning should be conducted over an imbalanced data set. In this paper, we address these two practical issues simultaneously by proposing a novel semi-supervised learning approach named Rocus. This method exploits the abundant unlabeled examples to improve the detection accuracy, as well as employs under-sampling to tackle the class-imbalance problem in the learning process. Experimental results on real-world software defect detection tasks show that Rocus is effective for software defect prediction. Its performance is better than a semi-supervised learning method that ignores the class-imbalance nature of the task and a class-imbalance learning method that does not make effective use of unlabeled data.

Keywords machine learning, data mining, semi-supervised learning, class-imbalance, software defect detection

1 Introduction

Enabled by technological advances in computer hardware, software systems have become increasingly powerful and versatile. However, the attendant increase in software complexity has made the timely development of reliable software systems extremely challenging. To make the software systems reliable, it is very important to identify as many defects as possible before releasing the software. However, due to the complexity of the software systems and the tight project schedule, it is almost impossible to extensively test every path of the software under all possible runtime environment. Thus, accurately predicting whether a software module contains defects can help to allocate the limited test resources effectively, and hence, improve the quality of the software systems. Such a process is usually referred to as *software defect detection*, which has already drawn much

This work was supported by National Science Foundation of China (60635030, 60903103), the Jiangsu Science Foundation (BK2008018).

attention in software engineering community.

Machine learning techniques have been successfully applied to build predictive models for software defect detection [1, 2, 3, 4, 5, 6, 7]. The static and dynamic code attributes or software metrics are extracted from each software module to form an example, which is then labeled as "defective" or "defect-free". Predictive models are then learned from a large number of examples, and is expected to accurately predict whether a given module is defective.

However, most of these studies have not considered two practical yet important issues in software defect detection. First, although it is relatively easy to automatically generate examples from software modules using some standard tools, determining whether a module contains defect through extensive test usually consumes too much time and resource, since program status grows exponentially as the complexity of software increases. With limited time and test resource, one can only obtain the labels for a small portion of modules. However, the predictive models learned from such a small labeled training set may not perform well. Second, the data in software defect detection are essentially imbalanced. The number of defective modules is usually much smaller than that of the defect-free modules. Ignoring the imbalance nature of the problem, a learner that minimizes the prediction error can often produce a useless predictive model that uniformly predicts *all* the modules as defect-free. Without taking these two issues into consideration, the effectiveness of software defect detection in many real-world tasks would be greatly reduced.

Some researchers have noticed the importance of these two issues in software defect detection and tried to tackle some of them based on machine learning techniques. For instance, Seliya and Khoshgoftaar [8] employed *semi-supervised learning* to improve the performance achieved on a small amount of labeled data by exploiting the abundant unlabeled data; contrarily, Pelayo and Dick [9] applied the resampling strategy to balance the skewed class distribution of the data set before learning the predictive model for software defect detection. Although attempting to tackle one issue may gain performance improvement to some extent, both methods suffer from the influence of other issue that they have not considered. If conventional semi-supervised learning is used, assuming that the learner can accurately assign labels to the unlabeled data, the learner may be easily biased by the overwhelming number of newly-labeled defect-free modules, and hence the refined model would be less sensitive to the defect modules. The sensitivity drops fast as the iterative semi-supervised learning proceeds. On the other hand, resampling methods would become less effective if provided with only a few labeled examples, where overfitting is inevitable no matter when replicating the small number of defective examples or reducing the number of overwhelming defect-free examples. Therefore, to achieve effective software defect detection, we need to consider these two important issues *simultaneously.* To the best of our knowledge, there is no previous work that has considered these two issues simultaneously in software defect detection.

In this paper, we address aforementioned two issues by proposing a novel semi-supervised learning method named ROCUS (RandOm Committee with Under-Sampling). This method incorporates recent advances in disagreement-based semi-supervised learning [10] with under-sampling strategy [11] for imbalanced data. The key idea is to keep the individual learner focusing on the minority-class during exploitation of the unlabeled data. Experiments on eight real-world software defect detection tasks show that ROCUS is effective for software defect detection. Its performance is better than both the semi-supervised learning method that ignores the class-imbalance nature of the tasks and the class-imbalance learning method that does not exploit unlabeled data.

The rest of the paper is organized as follows. Section 2 briefly reviews some related work. Section 3 presents the Rocus method. Section 4 reports the experiments over the software defect detection tasks. Finally, Section 5 concludes this paper.

2 Related Work

2.1 Software Defect Detection

Software defect detection, which aims to automatically identify the software module that contains certain defect, is essential to software quality insurance. Most of the software detection methods roughly fall into two categories. Methods in the first category leverage the execution information to identify suspicious program behaviors for defect detection [12, 13, 14], while methods in the second category elaborate to extract static code properties, which are usually represented by a set of *software metrics*, for each module in the software system [15, 16, 7]. Since it would be easier to measure the static code properties than the dynamic program behaviors, metric-based software defect detection has drawn much attention. Widely-used software metrics include LOC counts describing the module in term of size, Halstead attributes measuring the number of operators and operands in the module as the reading complexity [17] and McCabe complexity measures derived from the flow graph of the module [18].

In the past decade, machine learning has been widely applied to construct predictive models based on the extracted software metrics to detect defects in the software modules. Typical methods include linear or logistic regression [15, 7], classification and regression trees [3, 19], artificial neural networks [16, 1], memory-based methods [20, 21] and Baysian methods [22, 23]. In order to further increase the robustness to the outlier in the training data and improve the prediction performance, Guo et al. [2] applied ensemble learning to the software defect detection and achieved better performance compared to other commonly-used methods such as logistic regression and decision tree. Recently, Lessmann et al. [4] conducted an intensive empirical study, where they compared the predictive performance of 22 machine learning methods over the benchmark data sets of software defect detection.

Note that few previous studies have ever considered the characteristics of software defect detection (e.g., the defective module is difficult to collect). It has been showed that more accurate detection could be achieved even if only one of such characteristic is considered during learning [9, 8]. Since these characteristics are usually intertwined with each other, better performance could be expected if carefully considering them together during learning, which is what we have done in this paper.

2.2 Semi-supervised 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. Semi-supervised learning [24, 25] is a machine learning technique where the learner automatically exploits the large amount of unlabeled data in addition to few labeled data to help improving the learning performance.

Generally, semi-supervised learning methods fall into four major categories, i.e., generative-model based methods [26, 27, 28], low density separation based methods [29, 30, 31], graph-based methods [32, 33, 34], and disagreement-based methods [35, 36, 37, 38, 39].

Disagreement-based methods use multiple learners and exploit the disagreements among the learners during the learning process. If majority learner(s) are much more confident on a disagreed unlabeled example than minority learner(s), then the majority will teach the minority on this example. Disagreement-based semi-supervised learning originates from the work of Blum and Mitchell [35], where classifiers learned from two sufficient and redundant views teach each other using some confidently predicted unlabeled examples. Later, Goldman and Zhou [36] proposed an algorithm which does not require two views but require two different learning algorithms. Zhou and Li [38] 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 [37] further extended the idea in [38] by collaborating more classifiers in training process. Besides classification, Zhou and Li [39] 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., [40]), information retrieval (e.g., [41, 42]), computer-aided diagnosis (e.g., [37]), etc.

Few research applied semi-supervised learning to software defect detection, where the labeled training examples are limited while the unlabeled examples are abundant. Recently, Seliya and Khoshgoftaar [8] applied a generative-model based semi-supervised learning method to software defect detection and achieved performance improvement. Note that [8] adopted a generative approach for exploiting unlabeled data while the proposed method adopts a discriminative approach. Thus, we did not include it in our empirical study in the purpose of fair comparison.

2.3 Learning from Imbalanced Data

In many real-world applications such as software defect detection, the class distribution of the data is imbalanced, that is, the examples from the minority class are (much) fewer than those from the other class. Since it is easy to achieve good performance by keeping the majority-class examples being classified correctly, the sensitivity of the classifiers to the minority class may be very low if directly learning from the imbalanced data. To achieve better sensitivity on the minority class, the class-imbalance problem should be explicitly tackled.

Popular class-imbalance learning techniques include sampling [43, 11, 44] and cost-sensitive learning [45, 46]. Since sampling technique is used in this paper, we introduce sampling in more details.

Sampling attempts to achieve a balanced class distribution by altering the data set. Under-sampling reduces the number of the majority-class examples while over-sampling increases the number of minority-class examples [11], both of which have been shown to be effective on class-imbalance problems. Sophisticated methods can be employed to balance the class distribution, such as adding synthetic minority-class examples generated from the interpolation of neighboring minority-class examples [43]; discarding the non-representative majority-class examples to balance the class distribution [44]; combining different sampling methods for further improvement [47]; using ensemble technique for exploratory under-sampling to avoid the removal of useful majority class examples [48]; etc.

The class-imbalance learning method are seldom used in software defect detection. Recently, Pelayo and Dick [9] studied the effectiveness of SMOTE [43] over the software defect detection, and found that balancing the skewed class distribution is beneficial for software defect detection.

3 The Proposed Approach

Let $L = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_{m_0}, y_{m_0})\}$ denote the set of labeled examples and let $U = \{\boldsymbol{x}_{m_0+1}, \boldsymbol{x}_{m_0+2}, \ldots, \boldsymbol{x}_N\}$ denote the set of unlabeled examples, where $\boldsymbol{x}_i$ is a d-dimensional feature vector, and $y_i \in \{-1, +1\}$ is the class label. Conventionally, we denote $+1$ as the minority class (e.g., "defective" in software defect detection). Thereinafter, we referred to class $+1$ as the *minority-class* and -1 as the *majority-class*. Both L and U are independently drawn from the same unknown distribution $\mathcal{D}$ whose marginal distributions satisfy $P_{\mathcal{D}}(y_i = +1) \ll P_{\mathcal{D}}(y_i = -1)$, and hence, L and U are imbalanced data sets in essence.

As mentioned in Section 1, directly applying semi-supervised learning to imbalanced data would be risky. Since L is imbalanced and usually small, very few examples of the minority-class would be used to initiate the semi-supervised learning process. The resulting model may have poor sensitivity on the minority-class and hence can hardly identify the examples of the minority-class from the unlabeled set. In this case, learner would have to use little information from the minority-class and overwhelming information of the majority-class for model refinement, and this leads to even poorer sensitivity to the minority-class. As the iterative semi-supervised learning proceeds, the learned model would be biased to predict every example to the majority-class.

In order to successfully conduct iterative semi-supervised learning on the imbalanced data, the learner should have the following two properties. First, the learner should have strong generalization ability, such that even if provided with a small labeled training set with imbalanced class distribution, the learner would not have zero sensitivity to the minority-class examples during the automatically labeling process; second, the influence of overwhelming number of the newly labeled majority-class examples should be further reduced in order to improve the sensitivity of the learner to the minority examples after its refinement in each learning iteration. Based on these two consideration, we propose the ROCUS method to exploit the imbalanced unlabeled examples.

To meet the first requirement, we train multiple classifiers and then combine them for prediction. The reason behind this specific choice of the *ensemble learning* paradigm is that an ensemble of classifiers can usually achieve better generalization performance than a single classifier. Such superiority is more obvious when the training set is small [37] and the class distribution is imbalanced [48]. Thus, by exploiting the generalization power, the ensemble trained from L is able to identify some minority-class examples from U effectively.

Since multiple classifiers are used, we employ the disagreement-based semi-supervised learning paradigm [10] to exploit the unlabeled examples in U. In detail, after the initial ensemble of classifiers $\{h_1, h_2, ..., h_C\}$ are constructed, some individual classifiers select some examples in U to label according to a disagreement level, and then teach the other classifiers with the newly labeled examples. Here, similar to [37], we adopt a simple case where the classifiers $H_{-i} = \{h_1, ..., h_{i-1}, h_{i+1}, ..., h_C\}$ are responsible for selecting confidently labeled unlabeled examples in U for a individual classifier h_i. Given an unlabeled example, we first label this example using the majority voting of $C-1$ individual classifiers, and then estimate the labeling confidence using the degree of agreement on the current labeling among these classifiers. If the labeling confidence is greater than a preset threshold θ, we feed h_i with this newly labeled example for its refinement. Inspired by [27], we associate a weight (between 0 and 1) to each unlabeled example according to its labeling confidence such that the contribution of those less confidently labeled examples will be reduced during the classifier refinement. To unify the representation, the weight of a labeled example is fix to 1.

Note that even if the ensemble of classifiers can provide accurate prediction for each selected example, the sensitivity of the current classifier h_i may not be improved after its refinement with these newly labeled examples. Since U itself is imbalanced, h_i would still lose its sensitivity on the minority-class after learning with many newly-labeled majority-class examples in U. Here, we employ under-sampling [11], an efficient strategy for class-imbalance learning, to tackle this problem. Specifically, let $\tilde{L}_{it}$ denote the newly labeled set in the t-th round, where the total weights of the minority-class examples in $\tilde{L}_{it}$ is p_{it}; let L'_{it} denote the corresponding under-sampled set, where the total weights of the minority-class examples is p_{it} and that of majority-class examples is p_{it}/γ. Here, γ specifies the expected ratio between the minority-class and the majority-class and is usually fixed to 1 to produce a balanced class distribution.

However, since under-sampling reduces the number of newly labeled examples, even a few misclassification of unlabeled examples can greatly increase the noise rate of the newly labeled sets. Learning on noisy data set may humble the performance of resulting classifier. Thus, we stop the iterative semi-supervised learning process if the data sets used for the classifier refinement become too noisy to improve the performance of the classifier. The relationship between the classifier's worse-case error and noise rate has been studied by [49], and has been applied to derive the stopping criterion for some disagreement-based semi-supervised learning methods [37, 41, 38]. Here, our derivation of stopping criterion is almost the same with these methods. For the self-containess of this paper, we include the derivation below.

We denote W_0 and $W_{i,t}$ as the total weights of examples in L and $L'_{i,t}$, respectively. Let $\hat{e}_{i,t}$ denote the estimated error rate of H_{-i}, the ensemble of classifiers other excluding h_i. Assume that the noise rate of original labeled set L is very small, and hence the noise rate in the augmented training set, i.e., $L \cup L'_{i,t}$ for refining h_i in t-th iteration can be estimated by

$$\eta_{i,t} = \frac{\hat{e}_{i,t} W_{i,t}}{W_0 + W_{i,t}} \tag{1}$$

We can define the utility function of the refinement of classifier h_i in the t-th round based on the relationship between h_i's worse-case error $\epsilon_{i,t}$, the (weighted) number of examples in the augmented training set $(W_0 + W_{i,t})$ and its noise rate $\eta_{i,t}$ as

$$u_{i,t} \equiv \frac{c}{\epsilon_{i,t}^2} = (W_0 + W_{i,t})(1 - 2\eta_{i,t})^2 \tag{2}$$

where c is a constant that makes the equality hold.

Since $u_{i,t}$ is inverse-proportional to the square of the worse case classification error $\epsilon_{i,t}$, by enforcing $u_{i,t} > u_{i,t-1}$ in the succeeded rounds, the performance of h_i will be improved after refinement in t-th round. By comparing the right hand side of Eq.(2) in the t-th round and $(t-1)$-th round, we have that $u_{i,t} > u_{i,t-1}$ holds if

$$0 < \frac{\hat{e}_{i,t}}{\hat{e}_{i,t-1}} < \frac{W_{i,t-1}}{W_{i,t}} < 1 \tag{3}$$

In some cases, Eq.(3) might not hold since $W_{i,t}$ may be much greater than $W_{i,t-1}$. To make Eq.(3) hold again, we randomly discard some examples of both the minority-class and majority-class according to γ.

Note that we do not apply undersampling to L. The rationality behind is that undersampling L will further decrease number of the original labeled examples which are more reliable than the automatically labeled examples. The benefit from balancing the training data might be counteracted by discarding many reliably labeled data. However, in this case, the augmented training set for classifier refinement may be slightly imbalanced. In order to compensate the effect caused by such imbalance, we rescale the output of each classifier h_i using the minority-majority ratio of current training examples r_i and then combine them for final prediction:

$$H^*(x) = \begin{cases} +1, & \frac{1}{C}\sum_i \frac{s(+1|h_i(x))}{s(+1|h_i(x))+r_i s(-1|h_i(x))} > 0.5 \\ -1, & o.w. \end{cases} \tag{4}$$

where $s\left(y|h(x)\right) \in [0, 1]$ gives a score for the prediction of x. If the score is greater than 0.5, x is predicted as "+1", and "-1" otherwise.

As pointed out by Li and Zhou [37], such a majority-teach-one style process may gradually reduce the diversity between individual classifiers. Although the performance of individual classifiers can be improved through the semi-supervised learning process, the performance of the ensemble may not be improved or even degrade due to the rapid decrease of diversity. The reason is that the "teachers" of two individual classifiers are quite similar, and thus the newly labeled set of these two classifiers would be similar. Refining over similar data sets makes these two classifiers more similar. Following the suggestion of Li and Zhou [37], we inject certain amount of randomness into the base learner such that even if newly labeled examples are similar, the learned classifiers can still be different. Here, we call the ensemble of randomized classifiers as *random committee*. If we use random tree inducer that generates a randomized decision tree as the base learner, the ensemble is equivalent to what used in [37]. In this paper, we adopt another approach for randomness injection. Specifically, we project the data onto a set of randomly generated unit vectors and construct a classifier in this new space. We repeat this process to achieve a number new classifiers. The dimensionality of the new spaces is usually smaller than the original one in order to achieve further diversity between different new spaces. Similar approach was used by Ho [50] for constructing ensemble, where the random unit vectors are enforced to be parallel to the basis of the original space.

Table 1 presents the pseudo code of ROCUS. We firstly construct C classifiers from L by using Bagging [51] with the base learning algorithm $\mathcal{A}$. Any learning algorithm that incorporates certain randomness may be used to instantiate $\mathcal{A}$. In this paper, $\mathcal{A}$ injects randomness by conducting random projection before learning a classifier. In each semi-supervised learning iteration, each classifier h_i is refined using the newly labeled examples selected by H_{-i}, the ensemble of classifiers other than h_i. Before the refinement, under-sampling is employed to tailor the newly labeled set such that its minority-majority ratio is roughly γ. We use the condition in Eq.(3) as the stopping criterion of the iterative learning process. As holding a separate validation set is infeasible in semi-supervised learning settings, the error rate $\hat{e}_{i,t}$ of H_{-i} is estimated on L under an assumption that the training data and test data have the same distribution.

Note that an alternative way to address the problems of "lack of sufficient labeled data" and "data imbalance" simultaneously by imposing a "class proportion" constraint over a special type of base learner, which can adjust the portion of labeling of unlabeled data according to the constraint, just as what TSVM [31] does. However, such a strategy may exclude many good candidate base learners that have good performance over some particular defect detection problem but fail to adjust their labeling according to the constraints. In contrast, by incorporating under-sampling, disagreement-based semi-supervised learning method can be easily adapted to the exploitation of unlabeled data while the data are imbalanced. Since the requirement of the base learner in ROCUS is no more than the ability of injecting randomness, which can be easily achieved, we may choose different base learners according to specific application scenario, and hence applicability of ROCUS will be better.

Table 1. Pseudo-code of the Rocus algorithm

```
Algorithm: Rocus
Input:
    the labeled set $L$, the unlabeled set $U$,
    the confidence threshold $\theta$, the minority-majority ratio $\gamma$
    the number of individual classifiers $C$,
    the base learning algorithm $\mathcal{A}$ of the random committee
Process:
    1. Learn a random committee $\{h_1, \ldots, h_C\}$ from $L$ using Bagging and the base learning algorithm $\mathcal{A}$
    2. Repeat Step 3∼9 until none of the classifier in the random committee changes
    3.   Set $t$ ($t \in \mathbb{N}$) as the current iteration number
    4.   For each $i \in 1, \cdot, C$, do Step 4∼9
    5.     Estimate error $e_{i,t}$ of $H_{-i}$ on $L$
    6.     Label all the unlabeled examples with $H_{-i}$
    7.     Add the unlabeled examples whose labeling confidence exceeds threshold $\theta$ to a newly labeled set $L'_{i,t}$
    8.     Undersample $L'_{i,t}$ such that the ratio of minority class over the majority class is no less than $\gamma$
    9.     If Eq.(3) holds, retrain $h_i$ from $L \cup L'_{i,t}$ using the learning algorithm $\mathcal{A}$
Output: Output $H^*(x)$ according to Equation (4)
```

4 Empirical Studies

We evaluate the effectiveness of Rocus on eight software defect detection benchmark tasks. Each data set corresponds to different software projects in NASA Metrics Data Program [52]. Some of these software projects are developed for satellite flight control, while others are used for the ground-system. All the software projects are written in C/C++. Each project consists of a number of software modules, each of which is manually labeled as "defective" if one or more defects were reported during the test phase and "defect-free" otherwise. Typical software metrics such as *LOC counts*, *McCabe complexity measures* derived from the pathway of modules, *Halstead attributes* measuring the number of operators and operands in the module as the reading complexity, etc., are extracted from each software module using some standard code analysis tools [1] . The detailed information of all the software metrics used in the current study can be found in [5]. The detailed information of the experimental data sets are tabulated in Table 2, where Ratio denotes the inverse of the minority-majority ratio of the data set. It is obvious from the table that the data sets are imbalanced and the number of defective modules are smaller than that of the modules without any defect.

For each data set, we randomly select 75% examples for training and keep the the remaining examples aside for test. Since all the examples in the training set are labeled, in order to simulate the case where only a small portion of training data are labeled, we randomly partition the training set into labeled and unlabeled sets according to a *labeled rate* μ. For example, if a training set consisting of 1000 examples and $\mu = 10\%$, 100 examples are put into the labeled set with their labels, while the remaining 900 examples are put into the unlabeled set without their labels. In the experiment, we use four different labeled rates: 10%, 20%, 30% and 40%.

In the experiments, the randomized base learner of Rocus is instantiated as an Ad-

[1]Please refer to http://www.locmetrics.com/alternatives.html for some freely-accessed tools

Table 2. Software defect detection data sets

data	#attr.	#inst.	#min/#maj	Ratio	Description
jm1	21	10885	2106/8779	4.2	a real-time predictive ground system
kc1	21	2109	326/1783	5.5	a storage management system for receiving and processing ground data
kc2	21	522	107/415	3.9	another part of *kc1* project for science data processing
mw1	37	403	31/372	12.0	a zero gravity experiment system related to combustion
pc1	21	1109	77/1032	13.4	flight software for an earth orbiting satellite
pc3	37	1563	160/1403	8.8	flight software for an earth orbiting satellite
pc4	37	1458	178/1280	7.2	flight software for an earth orbiting satellite
pc5	38	17186	516/16670	32.3	a safety enhancement of a cockpit upgrade system

aBoost [53] preceded by a random projector. The random projector first randomly generates $2d/3$ random unit vectors and project all the examples onto these random vectors. Here, d is the number of features of the data set. Following the suggestions of [37], we fix the size of the random committee to $C = 6$, and the confidence threshold θ is set to 0.75, which indicates an unlabeled example is regarded to be confidently labeled if more than 3/4 individual classifiers of the random committee agree on its labeling. We set the minority-majority ratio $\gamma = 1$ to enforce the newly labeled example set to be balanced.

We compare the performance of Rocus with the following methods:

- Roca, standing for RandOm Committee using All labeled and unlabeled examples, is a disagreement-based semi-supervised learning method. Roca is almost the same as Rocus except it ignores the imbalanced class distribution. In each semi-supervised learning iteration, Roca directly uses all the automatically labeled examples for classifier refinement.

- Labeled is a random committee learned only from the labeled set. It can be regarded as the initial status of Roca and Rocus before exploiting any unlabeled examples.

- UnderSampling is a class-imbalance learning method. It works in supervised settings. Although it does not exploit the unlabeled data to improve its performance on defect detection, it attemps to reduce the influence of the imbalanced labeled set using under-sampling. The minority-majority ratio after under-sampling (γ) is set to 1.

- Smote [43] is another supervised class-imbalance learning method. Unlike UnderSampling reducing the number of the majority-class examples, Smote balanced the labeled data set by generating many virtual minority-class examples in the neighborhood of a minority-class example.

Note that we use AdaBoost as the base learning algorithm for the 4 compared methods, just like what is used in Rocus. Additionally, we use an AdaBoost learned only on the labeled set as the baseline for comparison. Since AdaBoost involves the training of multiple classifiers, to make the learning process of AdaBoost fast, we instantiate the base learner of AdaBoost as decision stumps which make decisions based on the value of only one feature. All the methods used in the experiment are implemented using Weka [54].

To compare performance of different methods on defect detection, we use two widely-used evaluation measures, namely F1-measure and AUC [55]. F1-measure summarizes the precision and recall of the detection. High precision and recall result in high F1-measure. AUC measures the area under the ROC cure, which indicates how well the test examples are ordered according to the real-value output of the classifiers. AUC is large if the test examples of the minority class are placed to the top of the ordering. Note that, unlike F1-measure directly reveals how well the prediction over the test examples are, AUC only shows the potential of a classifier to produce good classifications over the test examples. A classifier with high AUC can still produce bad classification if the learned threshold over the real-valued output is bad for the classification.

For each data set under different labeled rates, we repeat the random partition of labeled/unlabeled/test sets for 50 times, the performances of all the compared methods are averaged over the 50 runs. The average F1-measures are tabulated in Table 3∼6 and the average AUCs are tabulated in Table 7∼10. In each table, the best performance among all the compared methods is boldfaced on each experimental data set. The columns of "Roca imprv." and "Rocus imprv." report the performance improvement of Roca and Rocus, respectively, after exploiting the unlabeled data. Let a denote the initial performance before exploiting any unlabeled data and b denote the final performance after the semi-supervised learning. The improvement is computed as $(b-a)/a$. The last rows of the tables report the average performances over all the experimental data sets.

Table 3∼6 show that Rocus always outperforms the other compared methods for software defect detection tasks. The average F1-measure of Rocus is always the highest under different label rates. By comparing Rocus with its initial status Labeled, we can find that, after carefully exploiting the unlabeled examples, the Rocus is able to dramatically improve its initial performance. Even if there is only 10% training examples are labeled, the average performance improvement in terms of F1-measure still reaches 70.6%. As μ increases, such performance improvement can be even larger. Pairwise t-tests at 95% significance level indicate that the performance improvement on each data set is of statistical significance. This fact suggests that the unlabeled examples is beneficial for constructing better predictive model in software defect detection.

We first compare Rocus with another disagreement-based semi-supervised learning method Roca. Although the labeling strategy, the confidence estimation and the stopping criterion are exactly the same in both semi-supervised learning methods, the performance of the learned predictive models are quite different. In contrast to Rocus which is able to improve the performance using the unlabeled data, the performance of the predictive model learned by Roca degrades as the semi-supervised learning proceeds. It can be observed from the tables that, the final performance of Roca is worse than Labeled, which is the initial status shared by both semi-supervised learning methods, on almost all the data sets under different labeled rates. The only exception is on *kc2* when $\mu = 0.4$, where Roca appears slightly better than Labeled. Pairwise t-tests at 95% significance level show that the performance degradation is significant. In fact, the performance of Roca is the worst among all the compared methods, which is even worse than a simple AdaBoost directly applied only on the labeled set. This fact verifies our claim in previous sections that semi-supervised learning may not be effective on imbalance data sets, because the overwhelming majority-class examples in the newly-labeled set drive the focus of the predictive model away from the minority-class. Note that the only difference between these two semi-supervised learning algorithms is that Rocus explicitly tackles the class-imbalance problem using the

Table 3. The F1-measures of the compared methods when $\mu = 0.1$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.097	.150	.369	**.416**	.109	-27.3%	.413	174.8%
kc1	.216	.268	.342	.392	.185	-31.0%	**.419**	56.3%
kc2	.484	.444	.482	.492	.433	-2.6%	**.536**	20.7%
mw1	.197	.146	.225	.232	.111	-24.5%	**.238**	62.8%
pc1	.133	.167	.213	.191	.081	-51.2%	**.249**	49.6%
pc3	.155	.154	.244	**.280**	.038	-75.3%	.267	74.0%
pc4	.316	.238	**.434**	.417	.050	-78.8%	.398	67.2%
pc5	.304	.306	.449	.327	.158	-48.4%	**.487**	59.3%
avg.	.238	.234	.345	.343	.146	-42.4%	**.376**	70.6%

Table 4. The F1-measures of the compared methods when $\mu = 0.2$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.059	.126	.387	.426	.098	-22.1%	**.428**	240.0%
kc1	.206	.241	.366	.408	.189	-21.7%	**.425**	76.4%
kc2	.471	.463	.481	.502	.456	-1.4%	**.529**	14.3%
mw1	.206	.220	.226	.221	.147	-33.2%	**.266**	21.0%
pc1	.081	.156	.215	.199	.053	-66.0%	**.300**	92.1%
pc3	.127	.133	.272	.305	.032	-75.7%	**.326**	144.8%
pc4	.248	.293	**.505**	.481	.081	-72.4%	.475	61.9%
pc5	.228	.317	**.478**	.357	.230	-27.4%	.452	42.5%
avg.	.203	.244	.366	.362	.161	-4.0%	**.400**	86.6%

Table 5. The F1-measures of the compared methods when $\mu = 0.3$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.045	.087	.400	**.429**	.064	-26.4%	.426	388.9%
kc1	.159	.265	.387	.417	.219	-17.2%	**.441**	66.3%
kc2	.468	.504	.487	.535	.489	-3.0%	**.556**	10.3%
mw1	.219	.153	.219	.225	.099	-35.1%	**.254**	66.0%
pc1	.103	.159	.212	.228	.042	-73.6%	**.347**	117.9%
pc3	.053	.117	.279	.315	.021	-82.1%	**.367**	213.2%
pc4	.288	.316	**.536**	.504	.144	-54.5%	.497	57.5%
pc5	.226	.317	**.470**	.375	.266	-16.1%	.434	36.9%
avg.	.195	.240	.374	.379	.168	-38.5%	**.415**	119.6%

under-sampling technique during the exploitation of the unlabeled examples, while Roca completely ignores the class-imbalance problem. Therefore, it can be concluded that under-sampling is essential for the effectiveness of semi-supervised learning on imbalanced data sets.

Second, we compare Rocus with the two supervised learning methods Smote and UnderSample, which is able to tackle the class-imbalance problem in learning. It can be observed from the tables that although Smote or UnderSample may achieve the best performance on one or two data sets under certain label rate, the average F1-measures are always much less than that of Rocus. For example, when $\mu = 0.1$, the average F1-measure of Rocus is 0.376 while Smote and UnderSample

Table 6. The F1-measures of the compared methods when $\mu = 0.4$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.042	.087	.396	**.432**	.074	-14.9%	.425	390.3%
kc1	.193	.265	.395	.412	.214	-19.4%	**.443**	67.1%
kc2	.473	.480	.524	.520	.496	3.2%	**.562**	17.1%
mw1	.195	.175	.246	.202	.090	-48.6%	**.288**	64.8%
pc1	.071	.136	.224	.248	.077	-43.4%	**.353**	159.5%
pc3	.040	.098	.275	.328	.024	-75.5%	**.383**	290.2%
pc4	.272	.320	**.542**	.511	.141	-56.0%	.504	57.4%
pc5	.228	.330	**.480**	.378	.291	-11.9%	.411	24.5%
avg.	.189	.236	.385	.379	.176	-33.3%	**.421**	133.9%

Table 7. The AUCs of the compared methods when $\mu = 0.1$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.680	.697	.694	.687	**.698**	0.1%	**.698**	0.1%
kc1	.756	.773	.757	.747	.773	0.1%	**.780**	0.9%
kc2	.774	.777	.759	.759	.790	1.7%	**.806**	3.7%
mw1	.641	.680	.635	.604	.661	-2.7%	**.708**	4.1%
pc1	.716	**.741**	.715	.666	.727	-1.9%	.740	-0.1%
pc3	**.735**	.725	.732	.712	.716	-1.2%	.734	1.3%
pc4	.871	.806	**.878**	.825	.807	0.2%	.816	1.3%
pc5	.947	.955	.949	.943	.956	0.1%	**.957**	0.2%
avg.	.765	.769	.765	.743	.766	-0.5%	**.780**	1.4%

Table 8. The AUCs of the compared methods when $\mu = 0.2$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.694	.709	.706	.697	**.710**	0.2%	.709	0.1%
kc1	.766	.788	.771	.763	**.793**	0.6%	**.793**	0.6%
kc2	.781	.792	.760	.772	.787	-0.7%	**.806**	1.8%
mw1	.694	**.705**	.686	.648	.686	-2.6%	.704	-0.1%
pc1	.758	.798	.770	.705	.794	-0.5%	**.805**	0.8%
pc3	.764	.765	.770	.746	.766	.2%	**.774**	1.2%
pc4	.886	.849	**.906**	.878	.847	-0.3%	.851	0.3%
pc5	.950	.963	.955	.953	**.963**	0.0%	.962	-0.1%
avg.	.787	.796	.791	.770	.793	-0.4%	**.801**	0.6%

are 0.345 and 0.343, respectively. Such superiority can be more obvious when μ becomes larger. Moreover, Table 3∼6 also show that Smote, UnderSample and Rocus performs better than AdaBoost and Labeled, both of which fail to consider the imbalanced class distribution in the learning process. Therefore, it can be concluded that explicitly tackling the class-imbalance problem is helpful for software defect detection, and the performance of class-imbalance learning can be further improved if the unlabeled data are exploited in appropriate way.

Table 7∼10 tabulate the AUC values of all the compared methods under 4 different label rates. The trends in the 4 tables are similar to that of F1-measure shown in Table 3∼6. Specifically, Rocus always achieves the best

Table 9. The AUCs of the compared methods when $\mu = 0.3$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.698	.707	**.713**	.696	.706	-0.2%	.705	-0.3%
kc1	.769	**.800**	.780	.773	**.800**	0.0%	**.800**	-0.1%
kc2	.788	.798	.777	.788	.807	1.1%	**.811**	1.6%
mw1	.691	.693	.698	.678	.682	-1.6%	**.704**	1.7%
pc1	.774	.837	.796	.747	.834	-0.3%	**.850**	1.6%
pc3	.768	.781	.782	.763	.789	1.0%	**.791**	1.2%
pc4	.894	.867	**.915**	.893	.870	0.3%	.873	0.7%
pc5	.954	.964	.960	.956	**.965**	0.1%	.964	0.0%
avg.	.792	.806	.803	.787	.807	0.1%	**.812**	0.8%

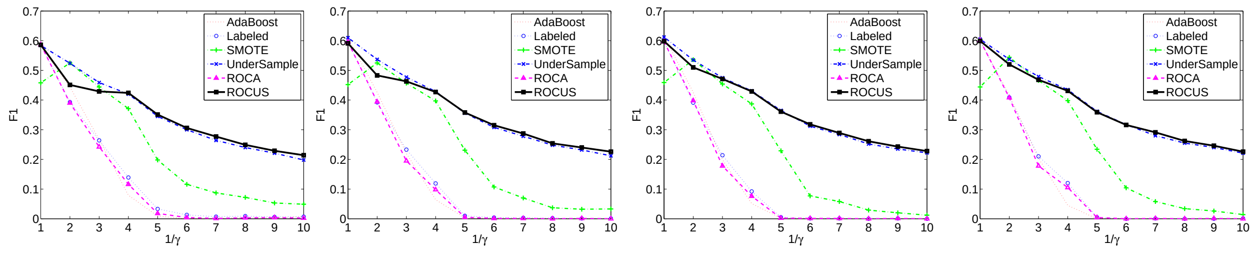
Table 10. The AUCs of the compared methods when $\mu = 0.4$

dataset	AdaBoost	Labeled	Smote	UnderSample	Roca	Roca imprv.	Rocus	Rocus imprv.
jm1	.702	.709	**.716**	.701	.709	0.0%	.709	0.0%
kc1	.769	**.802**	.782	.774	.800	-0.2%	.800	-0.1%
kc2	.800	.812	.785	.780	.813	0.2%	**.818**	0.8%
mw1	.699	**.726**	.716	.676	.717	-1.2%	.717	-1.2%
pc1	.788	.839	.812	.781	.845	0.6%	**.854**	1.7%
pc3	.770	.794	.789	.776	**.795**	0.2%	**.795**	0.1%
pc4	.899	.879	**.921**	.899	.876	-0.3%	.882	0.3%
pc5	.954	.965	.960	.956	**.966**	0.0%	.965	0.0%
avg.	.798	.816	.810	.793	.815	-0.1%	**.818**	0.2%

performance in terms of average AUC among the compared methods. The exploitation of unlabeled data in Roca can hardly lead to any improvement of the average AUC values and it even causes performance degeneration when μ is small. In contrast, after tackling the class-imbalance problem explicitly, unlabeled data become beneficial for Rocus. Moreover, the performance of Smote is comparable to the methods ignoring the class-imbalance problem (i.e., AdaBoost and Labeled), and the performance of UnderSample is even worse than these two methods. In contrast, by exploiting available unlabeled examples, Rocus performs better.

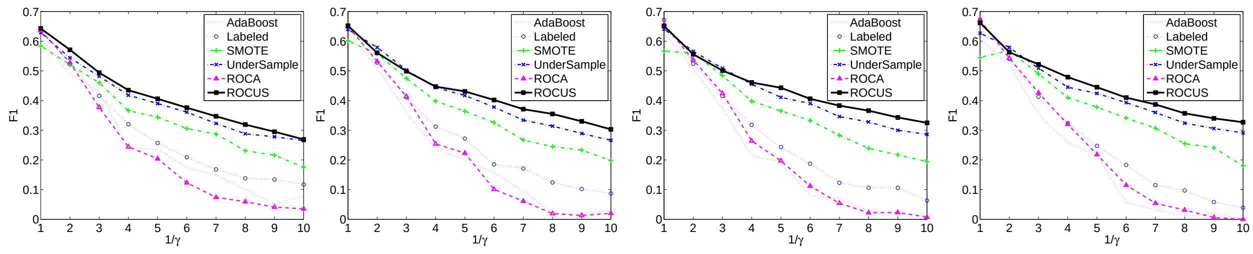
Although Table 3∼10 suggest that, in software defect detection, Rocus can effectively exploit unlabeled examples to achieve better performance even if the class distribution is imbalanced, its performance may vibrate under different degree of imbalance. In order to study the influence of the imbalance degree on Rocus, we conduct additional experiment, where we alter the class distribution of the data set. In detail, a data set is tailored such that the number of the defective examples over the number of the non-defective examples is roughly γ. In the experiment, $1/\gamma \in \{1, 2, \ldots, 10\}$. If the original ratio is larger than γ, we randomly discard some defective examples; otherwise, we randomly discard some defect-free examples. We repeat the experiment on each tailored data set for 50 times, where the labeled/unlabeled/test data sets are generated according to the 4 label rates (i.e., 10%, 20%, 30%, 40%). Since F1-measure directly reports how well the defect detection is, we only illustrate performances of the compared methods in terms of the F1-measure.

We plot the average F1-measure versus the inverse of the minority-majority ratio ($1/\gamma$) under different label rates on the experimental



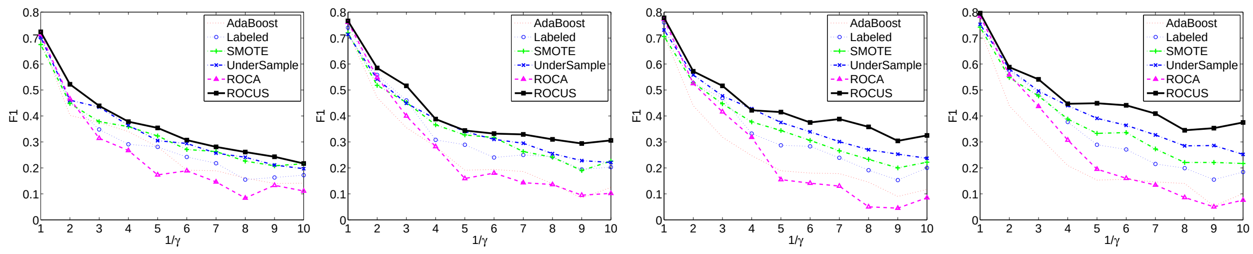
(a) $\mu = 10\%$ (b) $\mu = 20\%$ (c) $\mu = 30\%$ (d) $\mu = 40\%$

Fig. 1. F1-measures of the compared methods on *jm1* over different minority-majority rate



(a) $\mu = 10\%$ (b) $\mu = 20\%$ (c) $\mu = 30\%$ (d) $\mu = 40\%$

Fig. 2. F1-measures of the compared methods on *kc1* over different minority-majority rate



(a) $\mu = 10\%$ (b) $\mu = 20\%$ (c) $\mu = 30\%$ (d) $\mu = 40\%$

Fig. 3. F1-measures of the compared methods on *pc1* over different minority-majority rate

data sets. Due to the space limitation, we only report the results on data sets of small size (*pc1*), median size (*kc1*), and large size (*jm1*), as shown in Figure 1 to 3. Similar trends are observed on the other data sets.

As expected, F1-measures of all the compared methods decrease as the data set becomes more imbalanced, but the influence of the increase of class imbalance on Rocus is the smallest among the compared methods. It can be observed from the figures that Rocus almost always perform better than the other methods, and such superiority is more obvious when the class distribution becomes more imbalanced. We first consider the two semi-supervised learning methods. When the class distribution is balanced, Roca and Rocus appear to be comparable. As $1/\gamma$ increases, the performance of Roca degrades rather fast, especially when label rate is small. For example, when only 10% of the training data from *kc1* are labeled, the curve of Roca even drops below that of AdaBoost after $1/\gamma$ grows larger than 3. This fact suggests that the degree of imbalance has great influence on the semi-supervised learning method if it does not tackle the class-imbalance problem explicitly. Then, we compare Rocus, Smote and UnderSam-

PLE, each of which explicitly considers the imbalanced class distribution in learning. The figures show that there exist a gap between the curves of ROCUS and the other two methods, respectively. Such gap increases as $1/\gamma$ becomes larger, which indicates that exploiting unlabeled data can help to improve the performance of class-imbalance learning.

5 Conclusion

Detection of defects from software modules is important for improving the quality of a software system. However, many real-world data for software defect detection are imbalanced and only a small portion of examples are labeled as "defective" or "defect-free' in advance. In this paper, we address these two practical yet important issues simultaneously for software defect detection. We propose a disgreement-based semi-supervised learning method ROCUS to exploit the abundant unlabeled examples for better detection. ROCUS employs under-sampling to tailor the newly-labeled set, which effectively reduces the chance of the refined predictive model being less sensitive to the defective examples during the iterative semi-supervised learning process.Experimental results on real-world software defect detection tasks show that ROCUS can achieve better detection performance than a semi-supervised learning method that ignores the class-imbalance nature of the tasks and a class-imbalance learning method that is not able to exploit unlabeled data.

Note that ROCUS maintains the diversity of the ensemble between individual classifiers by injecting randomness into the base learner. The diversity can also be achieved by introducing *selective ensemble* [56]. Exploiting unlabeled data for improving the performance of individual classifiers and the diversity between them simultaneously in disagreement-based semi-supervised learning is an interesting future work.

In software defect detection, it would be more useful to predict the number of defects that may be contained in certain software module. Such a problem can be formalized as an regression problem, just as done in [57]. Recently, Zhou and Li [39] proposed a disagreement-based semi-supervised regression method, whose idea may be useful to extend ROCUS to predict the number of defects in each module in future. Moreover, since misclassifying a defective module as defect-free may lead to worse consequence than misclassifying a defect-free module as defective, extending ROCUS to cost-sensitive scenario, where the overall cost rather than misclassification error is minimized during learning, would be an another interesting future work. Additionally, in software defect detection settings, each module is represented as a single feature vector based on the metrics extracted from the module. It would be interesting to find appropriate metrics to represent each module with a set of feature vectors such that the defect detection problem may be solved within multi-instance learning [58] framework, which will be investigated in future.

Since the problems of "lack of sufficient labeled data" and "data imbalance" may be tangled together, exploiting the interaction between these two important issues in influencing the problems similar to software defect detection is expected to achieve better results, which is another interesting work to be done in future. Besides, it would be another interesting work to adapt ROCUS to other software engineering tasks or even the tasks beyond software engineering, where the data distributions are essentially imbalanced and the labels for examples are difficult to obtain.

Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve this paper, and thank Dr. Hongyu Zhang for his comments and Wei-Wei Tu for the proof-reading.

References

[1] Y.-S. Dai, M. Xie, Q. Long, and S.-H. Ng. Uncertainty analysis in software reliability modeling by Bayesian approach with maximum-entropy principle. *IEEE Transactions on Software Engineering*, 33(11):781–795, 2007.

[2] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *Proceedings of the 15nd International Symposium on Software Reliability Engineering*, 2004.

[3] T.M. Khoshgoftaar, E.B. Allen, W.D. Jones, and J.P. Hudepohl. Classification-tree models of software-quality over multiple releases. *IEEE Transactions on Reliability*, 49(1):4–11, 2000.

[4] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[5] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[6] Hongyu Zhang and Xiuzheng Zhang. Comments on "Data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(9):635–637, 2007.

[7] Y. Zhou and H. Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering*, 32(10):771–789, 2006.

[8] N. Seliya and T.M. Khoshgoftaar. Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Journal*, 15:327–344, 2007.

[9] L. Pelayo and S. Dick. Applying novel resampling strategies to software defect prediction. In *Proceedings of the 2007 Annual Meeting of the North American Fuzzy Information Processing Society*, pages 69–72, San Diego, CA, 2007.

[10] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knoledge and Information Systems*, 2009.

[11] C. Drummond and R. C. Holte. C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling. In *Working Notes of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, Washington, DC, 2003.

[12] Alice X. Zheng, Michael I. Jordan, Ben Liblit, Mayur Naik, and Alex Aiken. Statistical debugging: simultaneous identification of multiple bugs. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 1105–1112, Pittsburgh, PA, 2006.

[13] David Andrzejewski, Anne Mulhern, Ben Liblit, and Xiaojin Zhu. Statistical debugging using latent topic models. In *Proceedings of the 18th European Conference on Machine Learning*, pages 6–17, 2007.

[14] Trishul M. Chilimbi, Ben Liblit, Krishna K. Mehra, Aditya V. Nori, and Kapil

Vaswani. Holmes: Effective statistical debugging via efficient path profiling. In *Proceedings of the 31st International Conference on Software Engineering*, pages 34–44, Vancouver, Canada, 2009.

[15] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[16] T.M. Khoshgoftaar and E.B. Allen. Neural networks for software quality prediction. In W. Pedrycz and J. f. Peters, editors, *Computational Intelligence in Software Engineering*, pages 33–63. World Scientific, Singapore, 1998.

[17] M.H. Halstead. *Elements of Software Science*. Elsevier, 1977.

[18] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[19] Tibor Gyimóthy, Rudolf Ferenc, and István Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions Software Engineering*, 31(10):897–910, 2005.

[20] K. Ganesan, Taghi M. Khoshgoftaar, and Edward B. Allen. Verifying requirements through mathematical modelling and animation. *International Journal of Software Engineering and Knowledge Engineering*, 10(2):139–152, 2000.

[21] Taghi M. Khoshgoftaar and Naeem Seliya. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering*, 8(3):255–283, 2003.

[22] Norman E. Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions Software Engineering*, 25(5):675–689, 1999.

[23] Elena Pérez-Miñana and Jean-Jacques Gras. Improving fault prediction using bayesian networks for the development of embedded software applications. *Software Testing, Verification Reliability*, 16(3):157–174, 2006.

[24] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[25] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Department of Computer Sciences, University of Wisconsin at Madison, Madison, WI, 2006. http://www.cs.wisc.edu/∼jerryzhu/pub/ssl_survey.pdf.

[26] D. J. Miller and H. S. Uyar. A mixture of experts classifier with learning based on both labelled and unlabelled data. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 571–577. MIT Press, Cambridge, MA, 1997.

[27] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2-3):103–134, 2000.

[28] B. Shahshahani and D. Landgrebe. The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. *IEEE Transactions on Geoscience and Remote Sensing*, 32(5):1087–1095, 1994.

[29] O. Chapelle and A. Zien. Semi-supervised learning by low density separation. In *proceedings of the 10th International Work-*

shop on Artificial Intelligence and Statistics, pages 57–64. Savannah Hotel, Barbados, 2005.

[30] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 529–536. MIT Press, Cambridge, MA, 2005.

[31] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the 16th International Conference on Machine Learning*, pages 200–209, Bled, Slovenia, 1999.

[32] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(11):2399–2434, 2006.

[33] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[34] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, pages 912–919, Washington, DC, 2003.

[35] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, Madison, WI, 1998.

[36] S. Goldman and Y. Zhou. Enhancing supervised learning with unlabeled data. In *Proceedings of the 17th International Conference on Machine Learning*, pages 327–334, San Francisco, CA, 2000.

[37] M. Li and Z.-H. Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 37(6):1088–1098, 2007.

[38] Z.-H. Zhou and M. Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, 2005.

[39] Z.-H. Zhou and M. Li. Semi-supervised regression with co-training style algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1479–1493, 2007.

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

[41] M. Li and Z.-H. Zhou. Semi-supervised document retrieval. *Information Processing & Management*, 45(3):341–355, 2009.

[42] Z.-H. Zhou, K.-J. Chen, and H.-B. Dai. Enhancing relevance feedback in image retrieval using unlabeled data. *ACM Transactions on Information Systems*, 24(2):219–244, 2006.

[43] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique.

Journal of Artificial Intelligence Research, 16:321–357, 2002.

[44] M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *Proceedings of the 14th International Conference on Machine Learning*, pages 179–186, Nashville, TN, 1997.

[45] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, 1999.

[46] C. Elkan. The foundations of cost-senstive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 973–978, Seattle, WA, 2001.

[47] G. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations*, 6(1):20–29, 2004.

[48] X.-Y. Liu, J.-X. Wu, and Z.-H. Zhou. Exploratory under-sampling for class-imbalance learning. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 39(2):539–550, 2009.

[49] D Angluin and P Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[50] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[51] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[52] M. Chapman, P. Callis, and W. Jackson. Metrics data program. NASA IV and V Facility, [http://mdp.ivv.nasa.gov/], 2004.

[53] R. E. Schapire. A brief introduction to Boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1401–1406, Stockholm, Sweden, 1999.

[54] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.

[55] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(6):1145–1159, 1997.

[56] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[57] Taghi M. Khoshgoftaar and Naeem Seliya. Tree-based software quality estimation models for fault prediction. In *Proceedings of 8th IEEE International Conference Metrics Symposium*, pages 203–214, Ottawa, Canada, 2002.

[58] Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artifical Intelligence*, 89(1-2):31–71, 1997.

Yuan Jiang received the Ph.D degree in computer science from Nanjing University, China, in 2004. Now she is an associate professor at the Department of Computer Science and Technology, Nanjing University. Her research interests include machine learning, information retrieval and data mining. In these areas, she has published more than 30 technical papers in refereed journals or conferences. Dr. JIANG served as the publication chair of PAKDD'07, also served as program committee members for many conferences such as CCTA'07, BIBE'07, ICNC'08, ICNC'09, CCDM'09, NCIIP'09, CCML'10, etc. She is now a committee member of Machine Learning Society of Chinese Association of Artificial Intelligence (CAAI), and is a committee member of Artificial Intelligence Society of Jiangsu Computer Association. She is a member of Chinese Computer Federation(CCF). She was selected in the Program for New Century Excellent talents in University, Ministry of Education, in 2009.

Ming Li received the BSc and PhD degrees in computer science from Nanjing University, China, in 2003 and 2008 respectively. He is currently an assistant professor with LAMDA Group, the Department of Computer Sciences and Technology, Nanjing University. His major research interests include machine learning, data mining and information retrieval, especially on learning with labeled and unlabeled data. He has been granted various awards including the CCF Outstanding Doctoral Dissertation Award (2009), Microsoft Fellowship Award (2005), etc. He has served on the program committee of a number of important international conferences including KDD'10, ACML'10, ACML'09, ACM CKIM'09, IEEE ICME'10, AI'10, etc, and served as reviewers for a number of journals including IEEE Trans. KDE, IEEE Trans. NN, IEEE Trans. SMCC, ACM Trans. IST, Pattern Recognition, Knowledge and Information Systems, Journal of Computer Science and Technology, etc. He is a committee member of CAAI machine learning society, member of ACM, IEEE, IEEE computer society, CCF and CAAI.

Zhi-Hua Zhou received the BSc, MSc and PhD degrees in computer science from Nanjing University, China, in 1996, 1998 and 2000, respectively, all with the highest honors.

He joined the Department of Computer Science and Technology at Nanjing University as an assistant professor in 2001, and is currently Cheung Kong Professor and Director of the LAMDA group. His research interests are in artificial intelligence, machine learning, data mining, pattern recognition, information retrieval, evolutionary computation and neural computation. In these areas he has published over 70 papers in leading international journals or conference proceedings.

Dr. Zhou has won various awards/honors including the National Science and Technology Award for Young Scholars of China (2006), the Award of National Science Fund for Distinguished Young Scholars of China (2003), the National Excellent Doctoral Dissertation Award of China (2003), the Microsoft Young Professorship Award (2006), etc. He is an Associate Editor of IEEE Transactions on Knowledge and Data Engineering, Associate Editor-in-Chief of Chinese Science Bulletin, and on the editorial boards of Artificial Intelligence in Medicine, Intelligent Data Analysis, Science in China, etc. He is the founder of ACML, Steering Committee member of PAKDD and PRICAI, Program Committee Chair/Co-Chair of PAKDD' 07, PRICAI' 08 and ACML' 09, vice Chair or area Chair of conferences including IEEE ICDM' 06, IEEE ICDM' 08, SIAM DM' 09, ACM CIKM' 09, etc., and General Chair/Co-Chair or Program Committee Chair/Co-Chair of a dozen of native conferences. He is the chair of the Machine Learning

Society of the Chinese Association of Artificial Intelligence (CAAI), vice chair of the Artificial Intelligence and Pattern Recognition Society of the China Computer Federation (CCF), and chair of the IEEE Computer Society Nanjing Chapter. He is a fellow of IET, a member of AAAI and ACM, and a senior member of IEEE, IEEE Computer Society and IEEE Computational Intelligence Society, CCF.