2018 IEEE International Conference on Information Reuse and Integration for Data Science

# Misclassification Cost-Sensitive Software Defect Prediction

Ling Xu[1,2], Bei Wang[1],Ling Liu[2],Mo Zhou[1],Shengping Liao[1],Meng Yan[3]
[1]School of Software Engineering, Chongqing University, Chongqing, China
[2]College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
[3]College of Computer Science and Technology, Zhejiang University, Zhejiang, China
xuling@cqu.edu.cn; 695887391@qq.com; ling.liu@cc.gatech.edu;
{357668386,513167943}@qq.com; mengy@zju.edu.cn

***Abstract*—Software defect prediction helps developers focus on defective modules for efficient software quality assurance. A common goal shared by existing software defect prediction methods is to attain low classification error rates. These proposals suffer from two practical problems: (i) Most of the prediction methods rely on a large number of labeled training data. However, collecting labeled data is a difficult and expensive task. It is hard to obtain classification labels over new software projects or existing projects without historical defect data. (ii) Software defect datasets are highly imbalanced. In many real-world applications, the misclassification cost of defective modules is generally several times higher than that of non-defective ones. In this paper, we present a misclassification Cost-sensitive approach to Software Defect Prediction (CSDP). The CSDP approach is novel in two aspects: First, CSDP addresses the problem of unlabeled software detect datasets by combining an unsupervised sampling method with a domain specific misclassification cost model. This preprocessing step selectively samples a small percentage of modules through estimating their classification labels. Second, CSDP builds a cost-sensitive support vector machine model to predict defect-proneness of the rest of modules with both overall classification error rate and domain specific misclassification cost as quality metrics. CSDP is evaluated on four NASA projects. Experimental results highlight three interesting observations: (1) CSDP achieves higher Normalized Expected Cost of Misclassification (NECM) compared with state-of-art supervised learning models under imbalanced training data with limited labeling. (2) CSDP outperforms state-of-art semi-supervised learning methods, which disregards classification costs, especially in recall rate. (3) CSDP enhanced through unsupervised sampling as a preprocessing step prior to training and prediction outperforms the baseline CSDP without the sampling process.**

***Keywords-software defect prediction; cost-sensitive; semi-supervised; unsupervised sampling***

## I. INTRODUCTION

Software defect prediction is the process of predicting whether software modules have defects or not, and it is used as an effective means for software quality assurance. High quality software typically contains less defect-prone software modules. However, as the size and complexity of the software system continue to increase, it becomes difficult if not impossible to manually test and track all the paths of possible failures under a tight project schedule. Thus, automatically and accurately predicting whether a software module contains defects may help to guide software developers focus quality on software defect-prone modules, which reduces the test time, and improves the quality of software systems.

Most of the existing work for software defect prediction adopt statistical and machine learning methods, such as association rule mining [1], support vector machine (SVM) [2], neural networks [3], and decision tree [3]. However, these studies have not considered two critical and yet software defect specific challenges. First, the training dataset used in these studies lacks of well-labeled data about software modules for a number of reasons. Collecting labeled software data is an expensive task given limited time and limited resources for software testing. It is also difficult to collect labeled data for new projects or projects with limited development history. The problem of lacking well-labeled software defect modules is considered a barrier of adopting defect prediction in industry [5]. Second, the datasets of software modules are highly imbalanced with respect to the defect v.s. non-defect classification. The number of defect modules is usually much less than the number of non-defect ones. At the same time, the misclassification cost of putting defect software into non-defect class is much higher than for the misclassification cost of putting non-defect software to defect class. Such imbalance and the sensitivity of software defect prediction to the misclassification cost have been observed in many real-world applications and software testing practice [12]. We argue that software defect prediction methods should use both the overall classification error rates and the domain specific misclassification costs as two equally important quality metrics to evaluate the effectiveness of software defect detection methods.

Software defect prediction using machine learning (ML) algorithms has attracted much attentions in the last decade, ranging from semi-supervise learning [6], cross-project defect prediction [7], unsupervised learning [8], sampling [9], ensemble approaches [10], cost-sensitive [11], feature selection [12]. Most of these approaches improves prediction accuracy by focusing on addressing either the imbalance of labeled data and unlabeled data in the training set or the imbalance with respect to the misclassification cost between defect class and non-defect class. Few have ever addressed both problems under one unifying framework for software defect prediction. For instance, existing semi-supervised learning approaches focus on improving the ratio of labeled data in the training phase by developing class label predictor that assign labels to the unlabeled data using sampling methods. On one hand, sampling may lead to overfitting if provided with only a few labeled instances. Furthermore, most of class-label predictors are only effective when labeled data in the non-defective class is increasing and suffer from having imbalanced classes: a large number of newly labeled non-defective modules verse very few labeled defect modules. Even when more training data are labeled by non-defective class, the improvement of training data size is primarily on increasing the number of non-defective software modules. The fact that the actual number of software defect modules is significantly less than the number of non-defective software modules, makes the matter of imbalance of labeled data and unlabeled data even worse. As a result, even the total classification accuracy may be increased, the prediction model learner remains inefficient and suffering from the difficulty and low success rate in finding defective modules. Recently, a couple of researchers investigate the class imbalanced problem given the total set of labeled training data. A semi-supervise machine learning approach is proposed in [13], which promotes to employ under-sampling in the learning process, and [14] extends the proposal with an improved semi-supervised learning approach. Both employ random under-sampling technique to resample the original training set and updating training set in each round for co-train style algorithm. However, both methods consider that all classification errors have equal costs, regardless whether errors were misclassification of non-defect to defect or misclassification of defect to non-defect. They both aim to minimize the total expected costs.

In this paper, we highlight two observations: First, in software detect detection, the cost of misclassification of defect to non-defect in prediction is much higher and more destructive than that of misclassification of non-defect to defect. Thus, the misclassification cost for different classes has different effect on the quality of classification based prediction and simply using the total classification errors for all classes and the total number of misclassified records alone may not be effective measures. Second, the imbalance of defective samples v.s. non-defective samples in labeled training data may lead to overfitting, causing detrimental effect due to increased cost of misclassifying defective software into non-defective class. In order to achieve highly effective predictions, we need to consider different types of imbalance simultaneously and we need to evaluate the quality of software defect prediction classifiers using both the total classification error and the class-specific misclassification cost as the two core quality metrics of equal importance.

With these arguments in mind, we propose a Cost-sensitive Semi-supervised software Defect Prediction (CSDP) model. CSDP combine an unsupervised sampling method with a domain specific misclassification cost model. CSDP carries out the prediction task in two main phases. In the first phase, we adopt an unsupervised sampling method to estimate the label of a small percent of instances. The basic idea of unsupervised sampling is based on the observation that higher metric values tend to indicate more defect-proneness [16]. By obtaining a small percentage of labeled training data from the target dataset, these labeled training data may be used to construct a defect prediction model with a semi-supervised learning as a preprocessing step. In the second phase, CSDP builds the prediction model in a misclassification-cost sensitive manner by combining a cost-sensitive semi-supervised Support Vector Machine (CS4VM) [16] with software metrics for software defect prediction. We evaluate CSDP on four NASA projects. The empirical evaluation shows that CSDP is more effective compared with state-of-art supervised learning models, for software defect prediction, especially under the imbalanced data sets with limited labeling.

## II. MOTIVATION

In classic data mining and machine learning, classifiers usually focus on minimize the total errors of misclassification to achieve high classification accuracy. However, such an objective function is valid only when the misclassification cost of each class is equal. Unfortunately, there exist many real-world applications, in which different misclassifications are often associated with unequal costs. Table I gives two cases of misclassification cost. In Case 1, the cost of misclassifying a non-terrorist as terrorist is much lower than the cost of misclassifying an actual terrorist who may carry a bomb to a flight [25]. Case 2 is a multi-class problem. It is obviously that the cost of diagnosing a cancer patient as a healthy person is much more serious than erroneously diagnosing a healthy person as a cancer patient. The patients could lose their life because of a late diagnosis and treatment. Similarity, wrongly predicting healthy as cold or cold as a lung cancer, has a different cost than the other way around. In the context of software defect prediction, there are also two types of errors. Type I misclassification occurs when the classification model predicts a non-defective module as a defective one. Similarly, Type II misclassification may induce much worse prediction errors, when a defective module is wrongly classified as non-defective, because it may steer software defect undetected and lead to more serious damages. Thus, for software defect prediction, we argue that the cost of Type II misclassification is much higher than that of Type I misclassification.

Table I. EXAMPLES OF MISCLASSIFICATION

| | **Misclassification** | **Cost** |
|---|---|---|
| Case 1 | Non-Terrorist -> Terrorist | Low |
| | Terrorist-> Non-Terrorist | High |
| Case 2 | Healthy -> Lung Cancer, Healthy -> Cold | Low |
| | Cold -> Healthy | higher |
| | Lung Cancer -> Healthy, Lung Cancer -> Cold | highest |

Cost-sensitive learning methods consider the varying costs of different misclassification types. There are two kinds of misclassification cost: class-dependent cost and example-dependent cost [11]. The former is the costs of misclassifying any example in class A into the B class are the same. The latter is the costs of classifying different examples in class A into B are different even when the error types are the same. In practice, it is not easy to get the cost for every training example. Comparatively, class-dependent cost is often used because it is easier to obtain, which is the focus in this paper.

## III. THE CSDP APPROACH

Software defect prediction is typically modeled as a binary classification problem. In this paper, we use the cost matrix in Table II for CSDP, where "0" denotes that the cost of correct prediction of an instance, "C(-1)" denotes the cost of misclassifying a non-defective module to a defective module, "C(+1)" denotes the cost of misclassifying a defective module to non-defective one.

Table II. COST MATRIX FOR CSDP

| | Actual defect-prone | Actual non-defect-prone |
|---|---|---|
| Predict defect-prone | 0 | C(-1) |
| Predict non-defect-prone | C(+1) | 0 |

Figure 1 shows the architecture of the proposed CSDP system for software defect prediction. It consists of three phases: data preprocessing, unsupervised sampling, and training and prediction. We will discuss each of these three phases in the next three subsections.

## A. Data Preprocessing

We extract software metrics by using static analysis tools and apply the log transformation to standardize all metrics. This step aims to find the optimal feature subset that is necessary and sufficient for defect prediction. In most of the software datasets, some metrics have missing values, and thus the training and validation data input vectors of metrics contain null values. For numbercial metric fields, we adopt mean substitution, a common method for the situation where the size of data set is huge. It applies the mean of metric values to replace the null value, followed by a normalization preprocess since the value ranges of the metrics vary widely. We compared several commonly used normalization algorithms, and our experimental results show that the logarithm filtering normalization returned better result. In our implementation of CSDP, we apply a logarithm method for minimum value to avoid taking the logarithm of zero. Our normalization approach is defined as follows:

$$f(x) = \begin{cases} \ln(x + 0.000001), x <= 0 \\ \ln(x), x > 0 \end{cases} \quad (1)$$

Where x is the value of metrics. For non-numerical metrics, we first removed the records that have one or more critical metric values unknown.

## B. Unsupervised Sampling

Data imbalance happens in many real-world applications, especially those for defect detection, such as in healthcare, manufacturing, and software assurance. One extreme of such data imbalance may cause no defect samples in some training datasets when the datasets are randomly split for training and testing [3]. This problem can be aggravated, especially in the semi-supervised learning scenarios. Such problem may lead to very specific classification rules or missing rules for the minority class without sufficient generalization capability for future prediction [26].

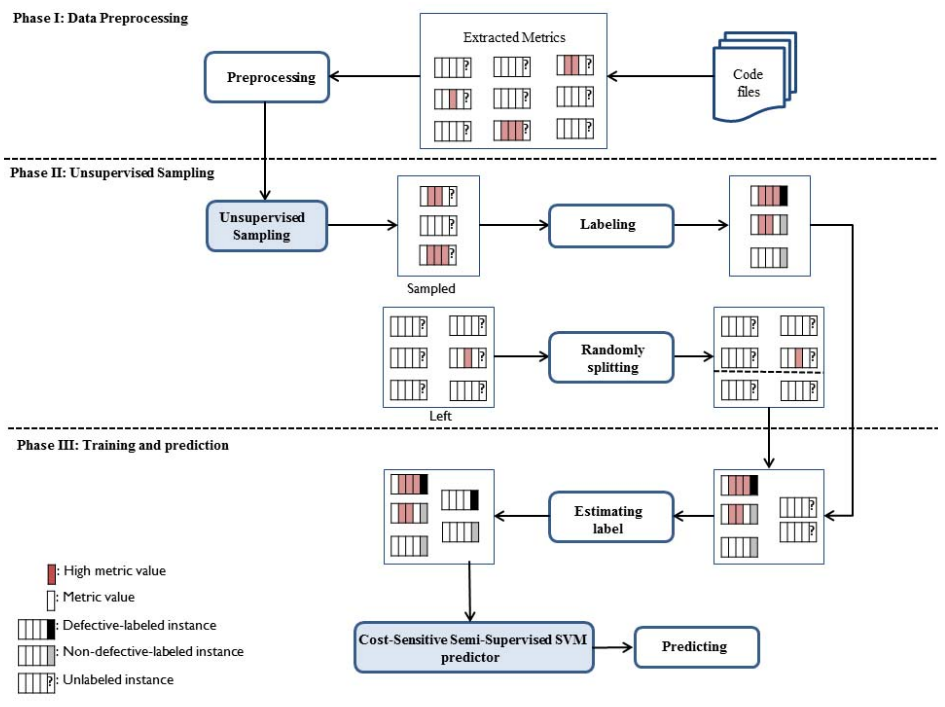

Phase I: Data Preprocessing
Preprocessing
Extracted Metrics
Code files
Phase II: Unsupervised Sampling
Unsupervised Sampling
Sampled
Labeling
Randomly splitting
Left
Phase III: Training and prediction
Estimating label
Cost-Sensitive Semi-Supervised SVM predictor
Predicting
: High metric value
: Metric value
: Defective-labeled instance
: Non-defective-labeled instance
: Unlabeled instance

Figure 1. The procedure of CSDP defect predictor

To address the problem of lacking labeled software instances, especially for new software projects, we propose to use a novel unsupervised sampling method to sample a few parts of fault modules that have high degree of defect indicators as labeled instances, and choose some non-defective instances randomly from the software repository. By putting the small amount of labeled data together with the set of unlabeled data examples together to form the training dataset, it is expected that the training data does closely resemble the original data distribution and may help addressing the unbalance distribution of software defect data.

Generally, the defective instances have higher metric values than non-defective ones [7]. Based on this observation, we propose an unsupervised sampling method to obtain the labeled data for training by using the magnitude of metric values. The use of the magnitude of metric values as the threshold to determine the sampling procedure is inspired by [15]. Figure 2 shows an example of this unsupervised sampling approach. Assume that we have six metrics represented by X1, …, X6 without label content for the seven software instances, denoted by I1, …, I7. We set the median value as the specific cutoff threshold for each of the six metrics. We can determine the samples to select using the following steps:

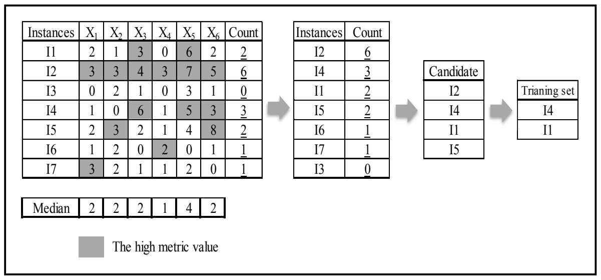
Step 1: Identifying the higher metric values that are greater than the median value. For example, the median of the metric X1 is 2 from {2, 3, 0, 1, 2, 1, 3}. The median of X5 is 4 from {6, 7, 3, 5, 4, 0, 2}. Thus, we identify the higher metric values of X1 are those two cells with value {3, 3}, which are higher than 2 and highlighted. Similarly, the higher metric values of X5 are those three cells with value {6, 7, 5}. This step is repeated until all other metrics are examined and computed.

Step 2：Counting the number of higher metric values (those highlighted in grey) for each software instance. For example, the number of higher metric values is 2 for instance I1 and 6 for instance I2. This step repeats until the count is computed for all instances.

Step 3: Sorting the instances by their total number (count) of higher metric values.

Step 4: Choosing $2*N$ previous instances as the candidates when assuming the size of labeled data as $N$. The goal is to get an appropriate range to randomly sample instances for training. For *N=2*, 4 instances will be chosen.

Step 5：Randomly sampling *N* instances to be labeled as the training data set from the candidate set.



| Instances | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | Count |
|---|---|---|---|---|---|---|---|
| I1 | 2 | 1 | 3 | 0 | 6 | 2 | 2 |
| I2 | 3 | 3 | 4 | 3 | 7 | 5 | 6 |
| I3 | 0 | 2 | 1 | 0 | 3 | 1 | 0 |
| I4 | 1 | 0 | 6 | 1 | 5 | 3 | 3 |
| I5 | 2 | 3 | 2 | 1 | 4 | 8 | 2 |
| I6 | 1 | 2 | 0 | 2 | 0 | 1 | 1 |
| I7 | 3 | 2 | 1 | 1 | 2 | 0 | 1 |

| Median | 2 | 2 | 2 | 1 | 4 | 2 |
|---|---|---|---|---|---|---|

| Instances | Count |
|---|---|
| I2 | 6 |
| I4 | 3 |
| I1 | 2 |
| I5 | 2 |
| I6 | 1 |
| I7 | 1 |
| I3 | 0 |

| Candidate |
|---|
| I2 |
| I4 |
| I1 |
| I5 |

| Trianing set |
|---|
| I4 |
| I1 |

Figure 2. Illustration of the phase in unsupervised approach

The goal of computing a candidate set and then randomly choose *N* instances from the candidate set is to select the samples from the top of a few candidates as the training set since they have higher metric values. As sampling rate affects the prediction performance, we set different sampling rates in the experiments. Then, we randomly sample unlabeled data from the whole dataset. The size of unlabeled sets is two times as large as labeled sets. Finally, the rest of datasets without labels are chosen as the testing data sets.

## *C. Training and Prediction*

We use the training dataset to build the cost-sensitive semi-supervised SVM predictor, and then predict the defect-prone instances from the rest unlabeled instances. Since semi-supervised learning uses a few labeled data for training, the size of the labeled set is an important factor in prediction performance. Thus, we use varying sampling rates to model the different proportions of labeled sets and apply different sampling rates in our study. Now we define our cost-sensitive software detect based SVM classifier.

Conventional SVM method is not cost sensitive and thus not suitable for the software defect classification. We build our CSDP model by employing a cost-sensitive semi-supervised learning algorithm based on low-density separation, such as CS4VM [16]. Let $\{(x_l, y_l), \ldots (x_l, y_l)\}$ be a set of labeled data, and $\{x_{l+1}, \ldots, x_{l+u}\}$ be a set of unlabeled data, where l and u are the total number of labeled samples and total number of unlabeled samples respectively, and $y \in \{\pm 1\}$. We denote $I_l=\{1, \ldots, l\}$ and $I_u=\{l+1, \ldots, l+u\}$ as the two indices for the labeled and unlabeled instances respectively. The labels $\hat{y} = [\hat{y}_i; i \in \mathcal{I}_u]$ of the unlabeled samples are unknown, and are optimized using the following objective function:

$$\begin{aligned} &\min_f \frac{1}{2}\|f\|_H^2 + C_1 \sum_{i\in\mathcal{I}_l} l(\mathrm{y}_i, f(x_i)) + C_2 \sum_{i\in\mathcal{I}_u} l(\hat{\mathrm{y}}_i, f(x_i)) \\ &\mathrm{s.t.} \sum_{i\in\mathcal{I}_u} \mathrm{sgn}(f(x_i)) = r, \hat{\mathrm{y}}_i = \mathrm{sgn}(\mathrm{f}(\mathrm{x}_i)), \forall i \in \mathcal{I}_u, \end{aligned} \tag{2}$$

Where *H* is the reproducing kernel Hilbert space (RKHS). $C_1$ and $C_2$ are the regularization parameters used for trading off the complexity and empirical errors on the labeled data and unlabeled data. The parameter *r* is defined by the user, which is the balance constraint to avoid the trivial solution that assigns all unlabeled instances to the same class. $\hat{y}'\mathbf{1}=r$, where 1 is the all-one vector. $l(y_i, f(x))$ is the weighted hinge loss, which is defined as:

$$l(y, f(x)) = C(y)\max\{0, 1 - yf(\mathrm{x})\} \tag{3}$$

Where $C(y)$ is the cost of misclassifying $y$ class, which reflects the cost-sensitive aspect of the CSDP model. To compare performance of different methods on defect detection, four widely used evaluation measures are used in our evaluation of CSDP. They are MR, FPR, FNR, and

NECM [22]. MR is the misclassification rate that indicates the ratio of number of wrongly predicted modules to the total number of modules. FPR and FNR denote the false positive rate (a non-defective module is misclassified as defect one) and false negative rate (a defective module is misclassified as non-defective one). The three metrics are commonly used performance metrics in software defect prediction [11], which are shown as follows:

$$MR = \frac{FP + FN}{TP + TN + FP + FN} \tag{4}$$

$$FPR = \frac{FP}{TN + FP} \tag{5}$$

$$FNR = \frac{FN}{TP + FN} \tag{6}$$

Where *TP* is the number of true positives, *FP* is the number of false positives, *TN* is the number of true negatives, and *FN* is the number of false negatives. Smaller values of *FNR*, *FPR* and *MR* represent better performance of models. Considering the misclassification costs and defective module rate, the Expect Cost of Misclassification (ECM) is generally used as a singular measure to compare the performances of different classification model. The *ECM* measure is defined in (7), which includes both the prior probabilities of the two classes and the misclassification costs. Since it is not practical to obtain the individual misclassification costs for the two type errors. The ECM is normalized over C(-1). The Normalized Expected Cost of Misclassification (NECM) is given in (8).

$$ECM = C(-1) \times FPR \times P_{ndp} + C(+1) \times FNR \times P_{dp} \tag{7}$$

$$NECM = FPR \times P_{ndp} + \frac{C(+1)}{C(-1)} \times FNR \times P_{dp} \tag{8}$$

Where $P_{ndp}$ and $P_{dp}$ are the non-defective rate and defect rate of modules in the dataset, C(-1) and C(+1) are the costs of false positive rate and false negative rate, respectively.

## IV. EXPERIMENTAL DESCRIPTION

***Datasets and Metrics***. We evaluate the effectiveness of CSDP using the *MR, FPR, FNR*, and *NECM* metrics on four popular datasets namely CM1, KC1, KC2, PC1 that are publicly available from NASA MDP [28]. Each dataset consists of several records and each record is composed of quality metrics extracted from software module. Table IV provides a brief description of five datasets. The proportion of the defective modules shows the high imbalance distribution with data varying from 6.9 to 20.5.

During the process of unsupervised sampling, we first label the sample instances according to sampling rate and then randomly choose some instances from the remaining data. To avoid the bias, this process is repeated 20 times so that the partition of dataset is different each time. For each performance measure, the mean value is calculated from the results of these 20 runs. For evaluating, the cost ratio is set from 1 to 10. That is, the experiments were run 200 times for each dataset.

***Comparative Methods***. We compared CSDP with three cost-sensitive learning models: CSBNN-WU1, CSBNN-WU2, CSBNN-TM [22], and two semi-supervised learning models: ACoForest[9] and S4VM[29]. CSBNN-WU1, CSBNN-WU2 and CSBNN-TM are different types of neural networks also studied in [8]. CSBNN-WU1 and CSBNN-WU2 introduce the cost matrix into the weight-updating process by making two modifications from the original weight-updating process. CSBNN-TM directly applies the threshold-moving in final output of AdaBoost algorithm. ACoForest is a sample based prediction model with active and semi-supervised learning. S4VM is a safe semi-supervised support machines learning approach.

***The Parameters of CSDP.*** The cost matrix given in Table II is used in the experiments reported in this paper. It is worth to note that in practice, such a cost matrix may vary for different software projects and various scenarios should be considered to define and obtain such a cost matrix. In this study, we use the cost ratio $C = C(+1)/C(-1)$ and vary the ratio from 1 to 10 to evaluate the prediction performance. For the kernel function of support vector machines, there is no commonly agreed-upon standard for choosing it. In our experiments, we found that Gaussian kernel performs better than linear kernel. Other core parameters and their settings are given in Table III. $C_1$ and $C_2$ are the parameters for trading off the complexity and empirical errors on the labeled and unlabeled data. Parameter *MaxIter* denotes maximal iteration number of process of estimating the label means for unlabeled training data. For each dataset, we sample a small portion of modules to be labeled according the sample rate $\mu$, while the remaining instances are unlabeled data for the training set and the testing set.

Table III. PARAMETERS OF CSDP ALGORITHM

| Parameter | Value |
|---|---|
| Kernel | 1-gaussian kernel |
| $C_1$ | 300 |
| $C_2$ | 300 |
| MaxIter | 100 |
| $\mu$ | 0.2 |

Table IV. SOFTWARE DEFECT DATASETS

| Data Set | Language | System | #Modules | #defectives | %Defect rate |
|---|---|---|---|---|---|
| CM1 | C | Spacecraft instrument | 498 | 49 | 9.8 |
| KC1 | C++ | Storage management | 2109 | 326 | 15.5 |
| KC2 | C++ | Scientific data processing | 522 | 107 | 20.5 |
| PC1 | C | Flight software | 1109 | 77 | 6.9 |

Table V. PERFORMANCE COMPARISON OF CSDP WITH TWO SEMI-SUPERVISED METHODS (SAMPLE RATE U=0.1)

| Datasets | MR | | | FPR | | | FNR | | | F-measure | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u=0.1 | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP |
| **cm1** | 0.128 | 0.128 | 0.204 | 0.047 | 0.043 | 0.158 | 0.877 | 0.892 | **0.655** | 0.136 | 0.136 | **0.234** |
| **pc1** | 0.080 | 0.085 | 0.264 | 0.020 | 0.025 | 0.243 | 0.880 | 0.902 | **0.595** | 0.150 | 0.129 | **0.154** |
| **kc1** | 0.181 | 0.164 | 0.482 | 0.070 | 0.060 | 0.488 | 0.790 | 0.731 | **0.445** | 0.258 | 0.332 | 0.229 |
| **kc2** | 0.195 | 0.194 | 0.458 | 0.070 | 0.092 | 0.484 | 0.669 | 0.589 | **0.334** | 0.381 | 0.451 | 0.338 |

Table VI. PERFORMANCE COMPARISON OF CSDP WITH TWO SEMI-SUPERVISED METHODS (SAMPLE RATE U=0.2)

| Datasets | MR | | | FPR | | | FNR | | | F-measure | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| u=0.2 | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP | ACoForest | S4VM | CSDP |
| **cm1** | 0.116 | 0.140 | 0.236 | 0.029 | 0.060 | 0.207 | 0.910 | 0.861 | **0.568** | 0.122 | 0.156 | **0.232** |
| **pc1** | 0.072 | 0.079 | 0.182 | 0.013 | 0.022 | 0.163 | 0.853 | 0.858 | **0.509** | 0.213 | 0.191 | **0.231** |
| **kc1** | 0.173 | 0.163 | 0.628 | 0.060 | 0.055 | 0.711 | 0.789 | 0.742 | **0.070** | 0.270 | 0.329 | 0.214 |
| **kc2** | 0.211 | 0.189 | 0.324 | 0.107 | 0.091 | 0.342 | 0.604 | 0.569 | **0.218** | 0.410 | 0.480 | 0.432 |

## A. *Performance of CSDP*

We first measure and compare the effectiveness of CSDP with the other five existing representative methods using the NECM metric as it is the most widely used metric for software defect prediction evaluation [23]. Figure 3 shows the performance comparison results of CSDP with three cost-sensitive neural network boosting algorithms (CSBNN-WU1, CSBNN-WU2, CSBNN-TM) on the four datasets, CM1, PC1, KC1, KC2, respectively. We evaluate the prediction performance by varying the cost ratio from 1 to 10 and show the NECM measurement results in the y-axis by varying cost ratios in the x-axis. From the Figure 3, we observe that CSDP achieves the lowest NECM in comparison, and is consistently lower than that of CSBNN-WU1 and CSBNN-WU2 for all four datasets and all cost ratios. This shows that when the data sets have highly skewed class distribution, CSDP is more effective than the other three methods.

Table V and Table VI show the prediction results of CSDP, compared with the two existing representative semi-supervised learning methods on the four datasets, with two different sampling rates, u=0.1 and u=0.2 respectively. The performance measurements for each of the three methods in comparison are averaged over the total of 20 runs. The cost ratio is estimated at 10. For each dataset, CSDP performs the best in terms of FNR, and lower FNR indicates that recall of defects is higher. This is consistent with the theory of Menzies et al. [18]: prediction models with low precision and high recall are useful in practice. For the two datasets (cm1 and pc1), CSDP achieves better performance than ACoForest and S4VM in the terms of F-measure. MR and FPR are not the lowest indicate the accuracy is compromised to a lesser extent as the recall increases. Our results shows that the proposed CSDP approach outperforms the ACoForest method by 35.7% and 38.2% for average F-measure when the sampling rate *u* is 0.1 and 0.2, respectively. Furthermore, CSDP outperforms S4VM by 46.4% and 33.4% in terms of average F-measure when the sampling rate u is 0.1 and 0.2, respectively. From above experiments, we can see that CSDP gives comparable results to CSBNN-WU1, CSBNN-WU2 and CSBNN-TM in MR, FPR, FNR, and NECM. Note that CSDP only needs little labeled data while other methods are all supervised learning. On the other hand, compared with semi-supervised models, CSDP achieves slightly higher recall, and it is more effective under the limited imbalance labeled data set.

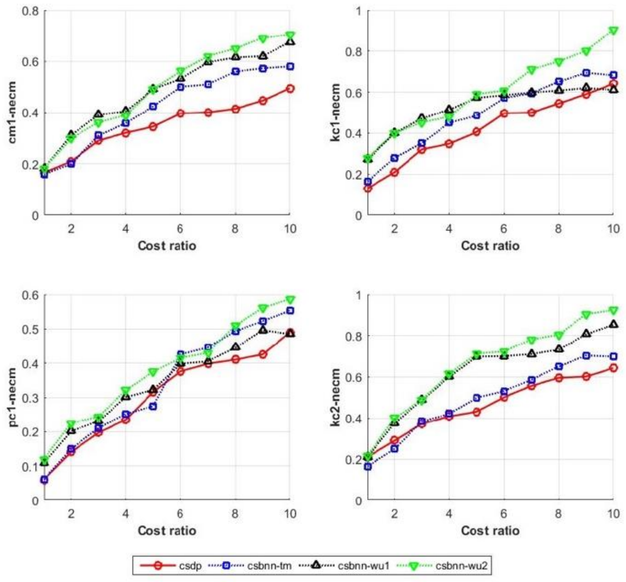


Figure 3. NECM Performance comparison of CSDP with three cost-sensitive neural networking boosting algorithms for cm1 dataset

## B. *Effect of cost setting*

Figure 4 shows the performance of CSDP in FPR, FNR and MR, which is affected by different settings of C(+1), C(-1) pairs. As we can see, while the cost of misclassification defective modules (C(+1)) increases (higher cost ratio), FNR performance decreases. It means that recall performance is improved as C(+1) increases. And as the cost of misclassification non-defective modules (C(-1)) increases (lower cost ratio), the FPR performance decreases. Since the major modules are non-defective modules, the MR performance increases with FPR performance increases. Lower FPR indicates that less errors in non-defective modules prediction. Experiments show that CSDP is sensitive to the defect cost.

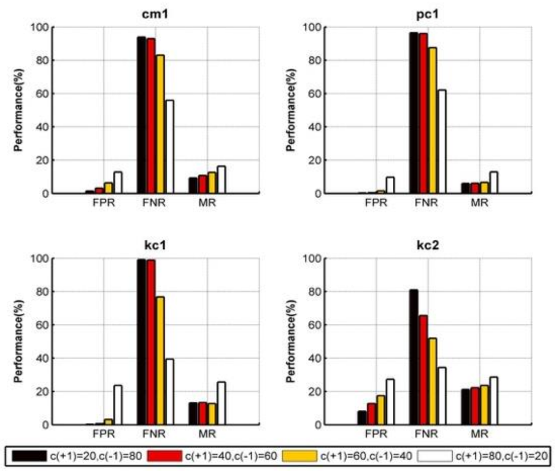


Figure 4. Performance of CSDP with different cost setting

### C. *Effect of unsupervised sample method*

Figure 5 and Figure 6 show the prediction performance of model using our proposed unsupervised sampling method compared with that of model with random sample method. It can be observed that CSDP performs better result in NECM and FNR, except for the case that the cost ratio is larger than 6 and the dataset kc1 is used. However, in that case the PNR of CSDP with sample method is lower. Moreover, CSDP obtain lower FNR (hi*gher recall) than random sample* method at most time except for the case that the cost ratio is larger than 5 and the dataset kc2 is used. In fact, the lower total misclassification cost and higher recall is useful for practical application. The experimental results suggest that using proposed sample method can improve the performance of prediction model.

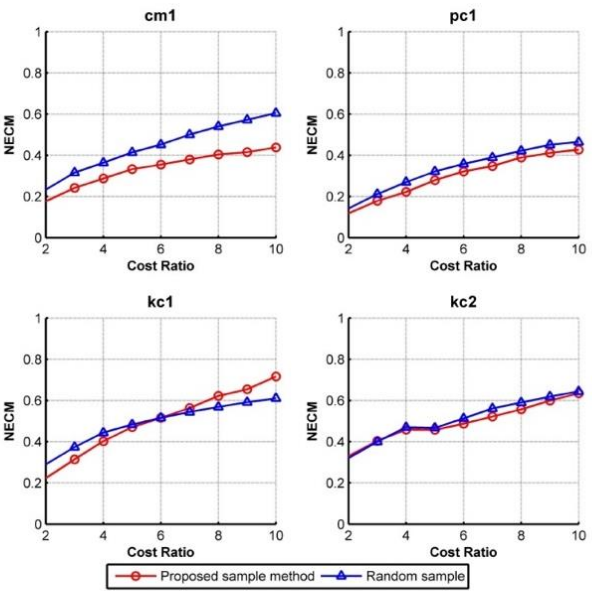


Figure 5. The NECM performance of the CSDP when uses sample method and when uses random sample method

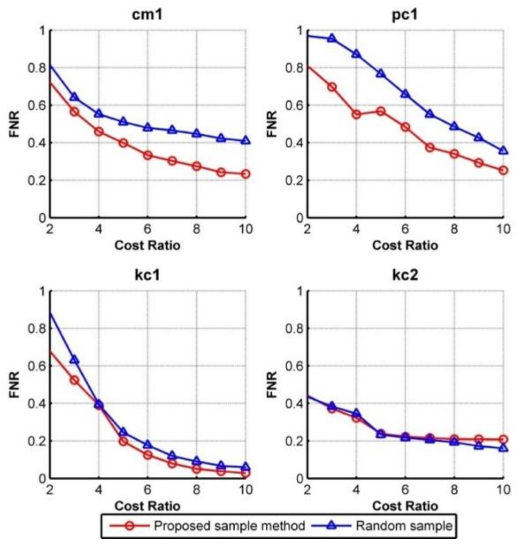


Figure 6. FNR performance of the CSDP when uses sample method and when uses random sample method

## V. RELATED WORK

*Software detect prediction.* Typical software defect prediction methods include Naïve Bayes [17], logistic regression [18], SVM [2], decision tree [1], neural network [3]. Some optimizations have also been proposed: [4] used multiple kernel ensemble learning classifier to predict defective module. [2] applied Convolutional Neural Network for feature generation with better performance compared to Deep Belief Network and logistic regression.

*Semi-supervised learning* describes a class of machine learning techniques that make use of a small set of labeled data together with a large amount of unlabeled data. [20] present a constraint-based semi-supervised clustering method with no defect data or quality-based class labels. [20] proposed an interactive self-training based semi-supervised approach named FTF, in which random forest as the base supervised learner and repeatedly updates by the labels of originally unlabeled software modules. [9] presents CoForest and ACoForest for sample-based semi-supervised defect prediction classifier construction.

*Class-imbalanced learning*. To address the class imbalanced detection, various techniques have been proposed to re-balance the class distribution by resampling, such as majority under-sampling and minority over-sampling, and to adapt existing classifier to bias towards the small class, such as cost-sensitive learning [22]. Cost-sensitive learning aims to minimize the total expected costs by utilizing the cost information. Some studies employed cost-sensitive learning methods including cost-sensitive neural network [23], cost-sensitive decision tree [24], and cost-sensitive boosting [22]. Recently, [11] proposed two-stage cost-sensitive learning method, which utilizes cost information in both classification stage and feature selection stage. [26] proposed three-way decision framework, which takes the uncertainty of two-way decisions into consideration and classifies objects based on a set of criteria. These studies focus solely on one of the two issues for improving performance of predictors. Few have considered addressing both issues simultaneously. [13] proposed Rocus, a semi-supervise learning approach, which employs under-sampling in learning process to solve the class-imbalance problem and did not take into account the different cost of misclassifications.

## VI. Conclusion

Software defect prediction is important to improve software quality of software system. However, software datasets lack labeled well-labeled software data and highly class imbalanced. To address the two practical challenges, we propose a cost-sensitive semi-supervised defect prediction (CSDP) approach by exploiting semi-unsupervised SVM and incorporating the cost information for better detection. First, we employ unsupervised sampling to increase the defect rate for semi-supervised SVM. Second, we built a predicting model with the lowest overall cost after considering the cost of different misclassification errors. Experimental results on real-world software datasets show that CSDP outperforms the state of the art semi-supervised learning methods which ignores imbalance classification costs, especially in terms of F-measure rate. CSDP can achieve comparable performance compared with the state of the art supervised learning models when only few labeled data is available.

**Acknowledgement**. This research is partially supported by National Natural Science Foundation of China (61772093), and NSF of Chongqing (cstc2015jcyjA40037), and China Postdoctoral Science Foundation (No.2017M621931), and Ling Liu's research is partially supported by USA NSF CISE SaTC grant No.1564097 and an IBM faculty award.

## References

[1] Song Q, Shepperd M, Cartwright M, et al. "Software defect association mining and defect correction effort prediction,"IEEE Transactions on Software Engineering, vol 32, Feb 2006, pp. 269-82.

[2] F. Xing, P. Guo, Lyu M R, "A novel method for early software quality prediction based on support vector machine," IEEE International Symposium on Software Reliability Engineering (ISSRE 2005).

[3] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," Applied Soft Computing, vol. 33, 2015.

[4] T. M. Khoshgoftaar and N. Seliya, "Tree-based software quality estimation models for fault prediction," in Proc. 8th IEEE Symp. Software Metrics, 2002.

[5] Catal, Cagatay. "Software fault prediction: A literature review and current trends." Expert systems with applications vol38, Apr 2011.

[6] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semi-supervised learning with dimension reduction," 2012 Proceedings of the 27th IEEE/ACM International Conference on in Automated Software Engineering (ASE), 2012, pp. 314-317.

[7] F. Zhang, Q. Zheng, Y. Zou, and AE. Hassan, Cross-project defect prediction using a connectivity-based unsupervised classifier, International Conference on Software Engineering, 2016.

[8] S. Zhong, TM. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation," IEEE International Symposium on High Assurance Systems Engineering, 2004.

[9] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," Automated Software Engineering, vol. 19, 2012, pp. 201-230.

[10] Sun Z, Song Q, Zhu X. "Using coding-based ensemble learning to improve software defect prediction." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) vol.42. Jun 2012, pp.1806-1817.

[11] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," Reliability, IEEE Transactions on, vol. 63, 2014, pp. 676-686.

[12] Gao, Kehan, et al. "Choosing software metrics for defect prediction: an investigation on feature selection techniques." Software: Practice and Experience vol.41, May 2011, pp. 579-606.

[13] Y. Jiang, M. Li, and Z.-H. Zhou, "Software defect detection with ROCUS," Journal of Computer Science and Technology, vol. 26, 2011, pp. 328-342.

[14] Ma, Ying, et al. "An improved semi-supervised learning method for software defect prediction." Journal of Intelligent & Fuzzy Systems, 2014, pp. 2473-2480.

[15] J. Nam and S. Kim, "CLAMI: Defect Prediction on Unlabeled Datasets," in Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, 2015, pp. 452-463.

[16] Y.-F. Li, J. T. Kwok, and Z.-H. Zhou, "Cost-sensitive semi-supervised support vector machine," in Proceedings of the National Conference on Artificial Intelligence, 2010.

[17] Menzies, Tim, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors." IEEE transactions on software engineering vol.33, Jan 2007, pp: 2-13.

[18] Olague, Hector M., et al. "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes." IEEE Transactions on software Engineering vol.33, Jun 2007, pp. 402-419.

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

[20] Lu, Huihua, Bojan Cukic, and Mark Culp. "An iterative semi-supervised approach to software fault prediction." Proceedings of the 7th International Conference on Predictive Models in Software Engineering. ACM, 2011, p. 15.

[21] Kamei, Yasutaka, et al. "The effects of over and under sampling on fault-prone module detection." IEEE First International Symposium on Empirical Software Engineering and Measurement, 2007.

[22] Zheng, Jun. "Cost-sensitive boosting neural networks for software defect prediction." Expert Systems with Applications vol.37, June 2010, pp. 4537-4543.

[23] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," Applied Soft Computing, vol. 33, 2015.

[24] M. J. Siers and M. Z. Islam, "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," Information Systems, vol. 51, 2015.

[25] Thai-Nghe, Nguyen, Zeno Gantner, and Lars Schmidt-Thieme. "Cost-sensitive learning methods for imbalanced data." IEEE International Joint Conference on Neural Networks (IJCNN), 2010.

[26] Li, Weiwei, Zhiqiu Huang, and Qing Li. "Three-way decisions based software defect prediction." Knowledge-Based Systems. vol.91, 2016, pp. 263-274.

[27] Y.-F. Li, J. T. Kwok, and Z.-H. Zhou, "Semi-supervised learning using label mean," in Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 633-640.

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

[29] Li, Yu-Feng, and Zhi-Hua Zhou. "Towards making unlabeled data never hurt." IEEE transactions on pattern analysis and machine intelligence, vol. 37 Jan 2015, pp.175-188.