# An Empirical Study for Software Fault-Proneness Prediction with Ensemble Learning Models on Imbalanced Data Sets

Renqing Li
School of Reliability and System Engineering, Beihang University, Beijing, China
Science and Technology on Reliability and Environmental Engineering Laboratory
Email: lirenqingbuaa@dse.buaa.edu.cn

Shihai Wang*
School of Reliability and System Engineering, Beihang University, Beijing, China
Science and Technology on Reliability and Environmental Engineering Laboratory
Email: wangshihai@buaa.edu.cn

***Abstract*—Software faults could cause serious system errors and failures, leading to huge economic losses. But currently none of inspection and verification technique is able to find and eliminate all software faults. Software testing is an important way to inspect these faults and raise software reliability, but obviously it is a really expensive job. The estimation of a module's fault-proneness is important to minimize the software testing resources required by guiding the resource allocation on the high-risk modules. Consequently the efficiency of software testing and the reliability of the software are improved. The software faults data sets, however, originally have the imbalanced distribution. A small amount of software modules holds most faults, while the most of modules are fault-free. Such imbalanced data distribution is really a challenge for the researchers in the field of prediction for software fault-proneness. In this paper, we make an investigation on software fault-prone prediction models by employing C4.5, SVM, KNN, Logistic, NaiveBayes, AdaBoost and SMOTEBoost based on software metrics. We perform an empirical study on the effectiveness of these models on imbalanced software fault data sets obtained from NASA's MDP. After a comprehensive comparison based on the experiment results, the SMOTEBoost reveals the outstanding performances than the other models on predicting the high-risk software modules with higher recall and AUC values, which demonstrates the model based on SMOTEBoost has a better ability to estimate a module's fault-proneness and furthermore improve the efficiency of software testing.**

***Index Terms*—SMOTEBoost, software fault-prone, prediction model, imbalanced data sets**

## I. Motivation and Introduction

With the fast development of software technology, software systems become more complex, requiring higher reliability and stability. Although the software would never be worn out, it may fail for some reasons at any time. The faults and mistakes in the software may lead to system errors and failures, resulting in huge losses in socio-economic activities. There are numerous similar cases [1, 2]. However, the current technologies are impossible to find and eliminate all faults in software. Software testing is a critical approach to ensure the software reliability [3], but a complete software testing is really time consuming and expensive. Some studies have proven that software development companies spend 50% to 80% of the costs on software testing [2]. Fault-proneness makes the definition on the probability of fault detection in software [4]. An accurate prediction on software faults or fault-prone at an early stage of software development is important to make a proper plan on testing activities and allocate the limited resources to focus on those modules, consequently improving the efficiency of software testing and the reliability of software system. Unfortunately, software fault-proneness cannot be directly measured. It can be estimated, however, based on software metrics, which provide quantitative descriptions of program attributes. A number of studies provide empirical evidence that correlation exists between software metrics and fault-proneness [5].

The software faults data sets are imbalanced, which means the amounts of the fault modules and fault free modules in a software system are imbalanced [5, 6, 7] and rises up the imbalanced data set learning problem for software fault-proneness prediction models based on pattern recognition technologies. In other words, most of faults involved in a software system are located in a small part of modules, which is called high-risk modules or minority class. On the contrary, most of models are defects free called low-risk modules or majority class. The accurate prediction on these high-risk modules is more important, because these modules may lead to serious consequences. For pattern recognition approaches, most prediction models are based on an assumption that the amounts of each class samples are balanced roughly.

Manuscript received June 8, 2013; revised July 19, 2013; September 1, 2013.
This work was supported by the National Natural Science Foundation of China (61300069).

When the assumption is satisfied, most of prediction models reveal good performance. But in these scenarios that the majority class typically represents most of the population, these models get poor predictive accuracy over the minority class [8]. This is a bias. Actually the accuracy of the minority class is far more critical than the one of the majority class. When high-risk modules are predicted wrongly, the following testing would ignore or pay a little attention on these modules, and consequently these defects are retained in a high probability after testing.

In this paper we present a comprehensive empirical study for software fault-proneness prediction with ensemble learning models on imbalanced data sets. The experiments results show that SMOTEBoost model has a better performance on predicting the imbalanced software fault-proneness than other models. The layout of the rest of the paper is as follows. Description of over-sampling algorithm SMOTE [9], Boosting [10] and SMOTEBoost [11] algorithm is presented in Section II. In Section III, data sets from NASA MDP (Metrics Data Program) [12], software metrics and performance metrics employed in our experiments are presented. Section IV describes the prediction models built in our study: SMOTEBoost model, AdaBoost [13] model and single learner models, such as C4.5 [14], SVM [15], KNN [16], Logistic [17], NaiveBayes [18]. Sections V and VI, present the experimental results and conclusions of our comparative study.

## II. The Software Fault-proneness Prediction Models Based on Imbalanced Data Sets

### *A. Synthetic Minority Over-sampling Techniques*

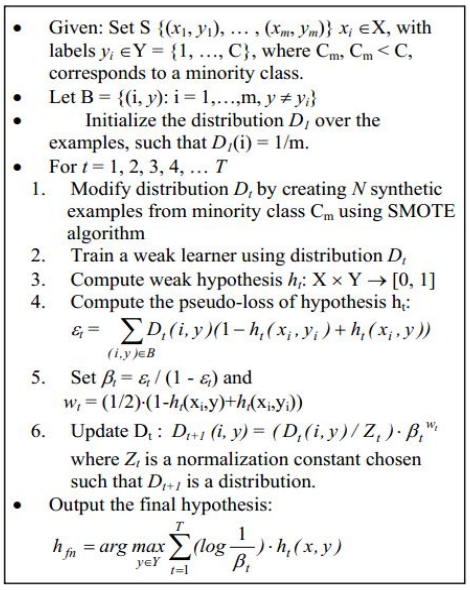
There are many real world data mining approaches learning from imbalanced datasets, which always leads to a bias on these classifiers, very high classification performance on the majority class(es), however, unacceptable performance over those minority ones. Over-sampling and under-sampling approaches are commonly employed to deal with the problem [19]. As one of most successful over-sampling technologies, SMOTE (Synthetic Minority Over-sampling Techniques) [9] is an approach specifically designed for learning with imbalanced datasets, which operates in the "feature space" rather than the "data space" and generates synthetic samples of the minority class. In this way the inductive learners are able to broaden their decision regions for the minority class and make a better performance.

### *B. Boosting Algorithm*

Boosting is an ensemble-based learning algorithm, which is able to dramatically improve the ensemble classification performance by creating and combining a set of weak hypotheses [10, 28]. AdaBoosting, a popular Boosting algorithm applied in many fields [13] [20], builds an ensemble of classifiers iteratively and adjust the weights of each training sample based on the performance of the iteration's classifier. These samples which were misclassified by pervious classifiers, will be assigned a bigger weight relatively, while those that were correctly classified have their weights decreased. Therefore, in the next iteration the learner is more likely to concentrate on those samples that were misclassified during the previous iterations and classify them correctly. At last, the classifiers built during each iteration participate in a weighted vote to classify unlabeled samples.

### *C. SMOTEBoost Algorithm*

SMOTEBoost [11] is an approach to learn from imbalanced data sets. More details are shown in Figure 1, which is based on an effective combination of the SMOTE algorithm and Boosting procedure.



```
• Given: Set S $\{(x_1, y_1), \dots, (x_m, y_m)\}$ $x_i \in X$, with
  labels $y_i \in Y = \{1, \dots, C\}$, where $C_m$, $C_m < C$,
  corresponds to a minority class.
• Let $B = \{(i, y): i = 1, \dots, m, y \neq y_i\}$
• Initialize the distribution $D_1$ over the
  examples, such that $D_1(i) = 1/m$.
• For $t = 1, 2, 3, 4, \dots T$
    1. Modify distribution $D_t$ by creating $N$ synthetic
       examples from minority class $C_m$ using SMOTE
       algorithm
    2. Train a weak learner using distribution $D_t$
    3. Compute weak hypothesis $h_t$: $X \times Y \to [0, 1]$
    4. Compute the pseudo-loss of hypothesis $h_t$:
       $\varepsilon_t = \sum_{(i,y)\in B} D_t(i,y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$
    5. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$ and
       $w_t = (1/2)\cdot(1 - h_t(x_i, y) + h_t(x_i, y_i))$
    6. Update $D_t$ : $D_{t+1}(i, y) = (D_t(i,y) / Z_t) \cdot \beta_t^{w_t}$
       where $Z_t$ is a normalization constant chosen
       such that $D_{t+1}$ is a distribution.
• Output the final hypothesis:
  $h_{fn} = \arg\max_{y \in Y} \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) \cdot h_t(x, y)$
```

Figure1. SMOTEBoost algorithm.

SMOTEBoost algorithm proceeds in a series of T iterations. In each iteration, a weak learner is called and presented with a different distribution $D_t$ altered by emphasizing particular training samples. The distribution is updated to give wrong classification samples with higher weights than correct ones. SMOTE is introduced in each round of Boosting and creates more samples of the minority class for each learner. Thus, more learning information is supplied for the minority class. Then the entire weighted training set is given to the weak learner to compute the weak hypothesis $h_t$ . At end, the different hypothesizes are combined into a final hypothesis $h_{fn}$ . Boosting procedure ensures the accuracy over the entire data sets and SMOTE improves the accuracy of the minority classes. SMOTEBoost has been applied to imbalanced data sets in many fields and shows a significant improvement in prediction performance [11].

## III. Experimental Design

### *A. Data Sets*

Our experiments are carried out on 12 data sets from the data metrics program NASA's MDP (Metrics Data Program) data repository [12]. Details about these data sets are in TABLE I, which provides the total number of software modules in the dataset (Size), as well as the number of minority defected modules (NumMin), the developing language and the LOC of the dataset (NumLOC).

TABLE I.
DATA SETS

| Data set | Language | Size | NumMin | NumLOC |
|---|---|---|---|---|
| CM1 | C | 505 | 48 | 16903 |
| JM1 | C | 10878 | 2102 | 457177 |
| KC1 | C++ | 2107 | 325 | 42963 |
| KC3 | JAVA | 522 | 107 | 7749 |
| MC1 | C++ | 9466 | 68 | 66583 |
| MC2 | C++ | 161 | 52 | 6134 |
| MW1 | C | 403 | 31 | 8134 |
| PC1 | C | 1107 | 76 | 25922 |
| PC2 | C | 5589 | 23 | 26863 |
| PC3 | C | 1563 | 160 | 36473 |
| PC4 | C | 1458 | 178 | 30055 |
| PC5 | C++ | 17186 | 516 | 161695 |

All the data sets listed in the Table 1 are imbalanced obviously, and the percentage of the minority class of PC2 is 0.4% only. In MDP, all of the software are implemented in different languages, such as CM1 is in C, KC1 in C++ and KC3 in JAVA. And the software presented in each data set is from different projects at NASA. For instance, CM1 is from the spacecraft software instruction module, KC1 from the storage management module and KC3 from the data transmitting and receiving module. Therefore the versatility of the prediction model can be tested and verified.

In our study, we take the software metric information of the software modules as input for the prediction model [5, 6]. Our study selects 22 metrics [21], which includes the McCabe metrics set, Halstead metrics set and Line of Code metrics set, etc. The details about these software metrics are in TABLE II. We select the first 21 metrics as the input for the prediction model [22] [23] and the last one to classify the software modules. The software module, whose ERROR_COUNT metric value is 0(This means the module contains no faults), is assigned as low-risk or majority class, with label “0”. The rest is high-risk or minority class, with label “1”.

TABLE II.
SOFTWARE METRICS

| | |
|---|---|
| CYCLOMATIC_COMPLEXITY | NUM_UNIQUE_OPERATORS |
| DESIGN_COMPLEXITY | HALSTEAD_LENGTH |
| ESSENTIAL_COMPLEXITY | HALSTEAD_CONTENT |
| LOC_TOTAL | HALSTEAD_DIFFICULTY |
| NUM_OPERANDS | HALSTEAD_EFFORT |
| NUM_OPERATORS | HALSTEAD_ERROR_EST |
| UM_UNIQUE_OPERANDS | HALSTEAD_PROG_TIME |
| LOC_BLANK | HALSTEAD_LEVEL |
| LOC_CODE_AND_COMMENT | HALSTEAD_VOLUME |
| LOC_COMMENTS | BRANCH_COUNT |
| LOC_EXECUTABLE | ERROR_COUNT |

For each time of our experiments, 70 percent of the majority class samples and minority class samples were randomly selected as training set respectively, the rest of samples is for testing.

## B. *Performance Metrics*

In this paper, we have employed two performance measures: a confusion matrix and the area under the ROC [24, 25, 26]. The confusion matrix as shown in TABLE III is typically used to evaluate performance of a machine learning algorithm.

TABLE III.
THE CONFUSION MATRIX

| | Predicted class “C” | Predicted class “NC” |
|---|---|---|
| Actual class “C” | True Positives (TP) | False Negatives (FN) |
| Actual class “NC” | False Positives (FP) | True Negatives (TN) |

The recall (Recall) and accuracy (ACC) as the experiment results are reported in this paper, which are defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{2}$$

The main focus of all learning algorithms is to improve the recall, without sacrificing the accuracy. A receiver operating characteristic curve (ROC) plots the true positive rate (TPR = TP/(TP+FP)) on the y-axis versus the false positive rate (FPR = FN/(FN+TN)) on the x-axis. The resulting curve represents the traded off between correctly identifying positive class examples and false alarms across the complete range of possible decision thresholds. The area under the ROC curve is used to measure the classifier performance.

## IV. PREDICTION MODELS

In order to make a comprehensive comparison study, besides SMOTEBoost(SMBM), four other kinds of prediction models are involved in our study: AdaBoost (ADAM), SMOTE+AdaBoost (SADAM), single weak learner (SINM), SMOTE+ single weak learner (SSINM). In this paper, five weak learners are selected: C4.5, SVM, 3NN, Logistic, NaiveBayes. The five learners are implemented in WEKA [27], an open source data mining suite. Except KNN, the other four learners have no parameters. Thus, in our study, the influence from the parameter setting is avoided.

## V. EXPERIMENTS AND RESULTS

In our study, we use 50 iterations for all Boosting models. In the experiments, Boosting resampling [28] rate is set to 80% in each iteration. Every prediction model runs five times of each data set independently. At last the mean (Mean) and variance (Var) values are shown in TABLE IV-TABL VIII. Bold values in the tables indicate SMBM resulted in the highest recall for the given datasets. The ROC results of models with C4.5 on dataset KC1 are presented in Figure 2.

TABLE IV.
C4.5 RESULTS

| C4.5 | | ACC | | | | | Recall | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data sets | | SINM | SSINM | ADAM | SADAM | SMBM | SINM | SSINM | ADAM | SADAM | SMBM |
| CM1 | Mean | 0.901 | 0.880 | 0.903 | 0.891 | 0.902 | 0.061 | 0.110 | 0.162 | 0.141 | **0.191** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.007 | 0.006 | 0.003 | 0.018 | 0.008 |
| JM1 | Mean | 0.814 | 0.811 | 0.797 | 0.807 | 0.793 | 0.198 | 0.254 | 0.388 | 0.318 | 0.381 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.002 | 0.001 | 0 | 0 | 0 |
| KC1 | Mean | 0.851 | 0.851 | 0.845 | 0.851 | 0.853 | 0.272 | 0.250 | 0.412 | 0.361 | **0.452** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.004 | 0.009 | 0.003 | 0.002 | 0.001 |
| KC3 | Mean | 0.828 | 0.820 | 0.810 | 0.822 | 0.802 | 0.411 | 0.333 | 0.533 | 0.483 | 0.478 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.02 | 0.010 | 0.003 | 0.001 | 0.001 |
| MC1 | Mean | 0.993 | 0.993 | 0.987 | 0.993 | 0.989 | 0.200 | 0.217 | 0.556 | 0.400 | 0.500 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.006 | 0.007 | 0.014 | 0.015 | 0.082 |
| MC2 | Mean | 0.702 | 0.717 | 0.694 | 0.683 | 0.713 | 0.188 | 0.353 | 0.341 | 0.294 | **0.376** |
| | Var | 0 | 0.002 | 0.003 | 0 | 0.002 | 0 | 0.017 | 0.006 | 0.019 | 0.018 |
| MW1 | Mean | 0.919 | 0.887 | 0.891 | 0.887 | 0.876 | 0.240 | 0.200 | 0.240 | 0.200 | 0.180 |
| | Var | 0 | 0 | 0 | 0.001 | 0 | 0.003 | 0.035 | 0.003 | 0 | 0.002 |
| PC1 | Mean | 0.945 | 0.935 | 0.922 | 0.935 | 0.933 | 0.224 | 0.224 | 0.360 | 0.028 | **0.392** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.004 | 0.004 | 0.012 | 0.009 | 0.006 |
| PC2 | Mean | 0.996 | 0.994 | 0.967 | 0.994 | 0.965 | 0 | 0 | 0.025 | 0 | **0.100** |
| | Var | 0 | 0 | 0 | 0 | 0 | | | 0.003 | | 0.026 |
| PC3 | Mean | 0.883 | 0.887 | 0.882 | 0.880 | 0.887 | 0.158 | 0.181 | 0.309 | 0.223 | 0.307 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.007 | 0 | 0.002 | 0.002 | 0.005 |
| PC4 | Mean | 0.901 | 0.893 | 0.901 | 0.916 | 0.905 | 0.322 | 0.442 | 0.582 | 0.582 | **0.592** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.029 | 0.009 | 0.005 | 0.004 | 0.003 |
| PC5 | Mean | 0.973 | 0.974 | 0.965 | 0.973 | 0.968 | 0.427 | 0.456 | 0.605 | 0.486 | 0.600 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.002 | 0.002 | 0.001 | 0.003 | 0.002 |

In TABLE IV, C4.5 is selected as the weak learner. When the models are applied on the 12 datasets, SMBM accounts 6 highest recall results (50.0%) based on datasets: CM1, KC1, MC2, PC1, PC2, PC4 and PC5. ADAM 6 highest recall results (50.0%). The ACC values of all models are very close on all datasets and the variance of SMBM's ACC values is 0.882.

TABLE V.
SVM RESULTS

| SVM | | ACC | | | | | Recall | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data sets | | SINM | SSINM | ADAM | SADAM | SMBM | SINM | SSINM | ADAM | SADAM | SMBM |
| CM1 | Mean | 0.901 | 0.602 | 0.775 | 0.762 | 0.724 | 0 | 0.704 | 0.621 | 0.463 | **0.752** |
| | Var | 0 | 0.013 | 0 | 0.191 | 0.002 | | 0.032 | 0.012 | 0.021 | 0.050 |
| JM1 | Mean | 0.809 | 0.669 | 0.779 | 0.698 | 0.779 | 0.01 | 0.590 | 0.523 | 0.475 | 0.520 |
| | Var | 0 | 0.005 | 0 | 0.002 | 0 | 0 | 0.007 | 0.001 | 0.007 | 0.002 |
| KC1 | Mean | 0.851 | 0.803 | 0.765 | 0.816 | 0.762 | 0.042 | 0.574 | 0.744 | 0.471 | 0.712 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.002 | 0.002 | 0.005 | 0.001 |
| KC3 | Mean | 0.832 | 0.701 | 0.829 | 0.770 | 0.750 | 0.206 | 0.222 | 0.744 | 0.472 | **0.833** |
| | Var | 0 | 0.016 | 0 | 0.004 | 0.036 | 0.007 | 0.006 | 0.014 | 0.010 | 0.014 |
| MC1 | Mean | 1 | 0.949 | 0.904 | 0.963 | 0.912 | 0 | 0.565 | 0.739 | 0.522 | 0.652 |
| | Var | | 0 | 0 | 0 | 0 | | 0.003 | 0.002 | 0.004 | 0.002 |
| MC2 | Mean | 0.709 | 0.721 | 0.743 | 0.728 | 0.713 | 0.141 | 0.341 | 0.553 | 0.388 | 0.482 |
| | Var | 0 | 0.004 | 0.004 | 0.005 | 0 | 0.008 | 0.005 | 0.011 | 0.025 | 0.016 |
| MW1 | Mean | 0.925 | 0.770 | 0.809 | 0.779 | 0.824 | 0 | 0.440 | 0.560 | 0.340 | 0.520 |
| | Var | 0 | 0.004 | 0.002 | 0.002 | 0 | | 0.048 | 0.033 | 0.023 | 0.057 |
| PC1 | Mean | 0.932 | 0.877 | 0.773 | 0.876 | 0.776 | 0 | 0.160 | 0.656 | 0.280 | 0.584 |
| | Var | 0 | 0 | 0 | 0 | 0.003 | | 0.003 | 0.011 | 0 | 0.020 |
| PC2 | Mean | 0.996 | 0.892 | 0.924 | 0.943 | 0.898 | 0 | 0.725 | 0.500 | 0.175 | 0.525 |
| | Var | 0 | 0 | 0 | 0 | 0 | | 0.011 | 0.086 | 0.005 | 0.104 |
| PC3 | Mean | 0.898 | 0.833 | 0.825 | 0.845 | 0.825 | 0 | 0.585 | 0.619 | 0.483 | **0.660** |
| | Var | 0 | 0 | 0 | 0 | 0 | | 0.010 | 0.006 | 0.008 | 0.004 |
| PC4 | Mean | 0.881 | 0.901 | 0.862 | 0.851 | 0.852 | 0.052 | 0.430 | 0.623 | 0.403 | **0.664** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0 | 0.004 | 0.002 | 0.004 | 0.004 |
| PC5 | Mean | 0.973 | 0.937 | 0.935 | 0.951 | 0.935 | 0.203 | 0.869 | 0.884 | 0.826 | **0.884** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.002 | 0.003 | 0.002 | 0.002 | 0.001 |

In TABLE V, SVM is employed as the weak learner. When the models are applied on the 12 datasets, SMBM accounts 5 highest recall results (41.7%) based on datasets: CM1, KC3, PC3, PC4, and PC5. ADAM 6 highest recall results (50.0%) and the other models less. The ACC values of all models are very close on all datasets and the variance of SMBM's ACC values is 0.812.

TABLE VI.
LOGISTIC RESULTS

| Logistic | | ACC | | | | | Recall | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data sets | | SINM | SSINM | ADAM | SADAM | SMBM | SINM | SSINM | ADAM | SADAM | SMBM |
| CM1 | Mean | 0.901 | 0.803 | 0.823 | 0.791 | 0.801 | 0.152 | 0.392 | 0.444 | 0.352 | **0.492** |
| | Var | 0 | 0.002 | 0.001 | 0.001 | 0 | 0.001 | 0.011 | 0.009 | 0.020 | 0.02 |
| JM1 | Mean | 0.816 | 0.805 | 0.758 | 0.804 | 0.757 | 0.106 | 0.243 | 0.581 | 0.256 | 0.574 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0 | 0.005 | 0 | 0.005 | 0 |
| KC1 | Mean | 0.861 | 0.811 | 0.762 | 0.814 | 0.763 | 0.202 | 0.420 | 0.653 | 0.400 | **0.670** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.002 | 0.001 | 0.005 | 0.001 |
| KC3 | Mean | 0.826 | 0.707 | 0.810 | 0.723 | 0.805 | 0.394 | 0.433 | 0.670 | 0.483 | **0.672** |
| | Var | 0 | 0.004 | 0 | 0.003 | 0 | 0.005 | 0.004 | 0.003 | 0.007 | 0.006 |
| MC1 | Mean | 0.992 | 0.984 | 0.908 | 0.985 | 0.915 | 0 | 0.234 | 0.722 | 0.217 | 0.678 |
| | Var | 0 | 0 | 0 | 0 | 0 | | 0.025 | 0.002 | 0.022 | 0.002 |
| MC2 | Mean | 0.725 | 0.721 | 0.740 | 0.740 | 0.766 | 0.376 | 0.435 | 0.541 | 0.506 | **0.600** |
| | Var | 0.003 | 0.002 | 0.002 | 0.001 | 0.004 | 0.013 | 0.020 | 0.037 | 0.018 | 0.020 |
| MW1 | Mean | 0.912 | 0.760 | 0.830 | 0.790 | 0.542 | 0.160 | 0.420 | 0.340 | 0.380 | **0.620** |
| | Var | 0 | 0.003 | 0 | 0.003 | 0.134 | 0.003 | 0.047 | 0.008 | 0.037 | 0.162 |
| PC1 | Mean | 0.932 | 0.879 | 0.811 | 0.871 | 0.808 | 0.120 | 0.210 | 0.648 | 0.280 | 0.608 |
| | Var | 0 | 0 | 0.001 | 0 | 0 | 0.006 | 0.007 | 0.005 | 0.020 | 0.008 |
| PC2 | Mean | 0.994 | 0.929 | 0.938 | 0.931 | 0.924 | 0 | 0.225 | 0.325 | 0.200 | **0.350** |
| | Var | 0 | 0 | 0 | 0 | 0 | | 0.034 | 0.020 | 0.036 | 0.010 |
| PC3 | Mean | 0.903 | 0.839 | 0.792 | 0.841 | 0.793 | 0.166 | 0.491 | 0.702 | 0.475 | 0.680 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.005 | 0 | 0.004 | 0.003 |
| PC4 | Mean | 0.912 | 0.882 | 0.854 | 0.872 | 0.862 | 0.311 | 0.503 | 0.725 | 0.491 | **0.755** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.006 | 0.011 | 0.014 | 0.012 | 0.013 |
| PC5 | Mean | 0.973 | 0.945 | 0.929 | 0.945 | 0.930 | 0.344 | 0.649 | 0.876 | 0.556 | **0.886** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.002 | 0 | 0.002 | 0 |

In TABLE VI, Logistic is used as the weak learner. When the models are applied on the 12 datasets , SMBM accounts 8 highest recall results (66.7%) based on datasets: CM1, KC1, KC3, MC2, MW1, PC2, PC4, and PC5. ADAM 4 highest recall results (33.3%). The ACC values of all models are very close on all datasets and the variance of SMBM's ACC values is 0.805.

TABLE VII.
3NN RESULTS

| 3NN | | ACC | | | | | Recall | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data sets | | SINM | SSINM | ADAM | SADAM | SMBM | SINM | SSINM | ADAM | SADAM | SMBM |
| CM1 | Mean | 0.891 | 0.843 | 0.825 | 0.821 | 0.831 | 0.062 | 0.184 | 0.266 | 0.262 | 0.232 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.004 | 0.006 | 0.030 | 0.021 | 0.031 |
| JM1 | Mean | 0.811 | 0.812 | 0.738 | 0.760 | 0.739 | 0.244 | 0.267 | 0.450 | 0.403 | **0.457** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| KC1 | Mean | 0.852 | 0.842 | 0.823 | 0.831 | 0.824 | 0.243 | 0.291 | 0.451 | 0.412 | **0.475** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| KC3 | Mean | 0.853 | 0.823 | 0.784 | 0.792 | 0.769 | 0.489 | 0.489 | 0.539 | 0.483 | **0.578** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.005 | 0.005 | 0.007 | 0.010 | 0.011 |
| MC1 | Mean | 0.992 | 0.990 | 0.976 | 0.994 | 0.976 | 0 | 0 | 0.778 | 0.222 | **0.778** |
| | Var | 0 | 0 | 0 | 0 | 0 | | | 0.002 | 0.002 | 0.001 |
| MC2 | Mean | 0.725 | 0.728 | 0.706 | 0.735 | 0.687 | 0.271 | 0.388 | 0.435 | 0.459 | 0.353 |
| | Var | 0 | 0 | 0.002 | 0.006 | 0.003 | 0.008 | 0.018 | 0.025 | 0.026 | 0.022 |
| MW1 | Mean | 0.913 | 0.813 | 0.825 | 0.830 | 0.831 | 0.260 | 0.300 | 0.340 | 0.320 | **0.340** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.013 | 0.025 | 0.013 | 0.002 | 0.013 |
| PC1 | Mean | 0.934 | 0.918 | 0.875 | 0.885 | 0.881 | 0.208 | 0.208 | 0.504 | 0.472 | **0.512** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.003 | 0.003 | 0.007 | 0.004 | 0.001 |
| PC2 | Mean | 0.996 | 0.989 | 0.957 | 0.998 | 0.960 | 0 | 0 | 0.375 | 0.250 | **0.375** |
| | Var | 0 | 0 | 0 | 0 | 0 | | | 0.001 | 0.003 | 0.001 |
| PC3 | Mean | 0.885 | 0.867 | 0.841 | 0.841 | 0.839 | 0.162 | 0.325 | 0.468 | 0.400 | 0.453 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.008 | 0 | 0.005 | 0 |
| PC4 | Mean | 0.901 | 0.891 | 0.843 | 0.844 | 0.845 | 0.382 | 0.432 | 0.531 | 0.522 | **0.553** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.002 | 0.004 | 0 | 0 | 0.002 |
| PC5 | Mean | 0.969 | 0.971 | 0.971 | 0.972 | 0.9720 | 0.430 | 0.511 | 0.500 | 0.465 | **0.512** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.002 | 0.003 | 0.002 | 0．002 | 0.001 |

In TABLE VII, 3NN is the weak learner used. When the models are applied on the 12 datasets, SMBM accounts 9 highest recall results (75%) based on datasets: JM1, KC1, KC3, MC1, MW1, PC1, PC2, PC4, and PC5. ADAM 5 highest recall results (41.7%) and the other models less. The ACC values of all models are very close on all datasets and the variance of SMBM's ACC values is 0.846.

TABLE VIII.
NAIVEBAYES RESULTS

| NaiveBayes | | ACC | | | | | Recall | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data sets | | SINM | SSINM | ADAM | SADAM | SMBM | SINM | SSINM | ADAM | SADAM | SMBM |
| CM1 | Mean | 0.861 | 0.634 | 0.661 | 0.712 | 0.655 | 0.302 | 0.535 | 0.692 | 0.402 | 0.612 |
| | Var | 0.001 | 0.022 | 0.005 | 0.005 | 0.011 | 0.013 | 0.043 | 0.030 | 0.031 | 0.031 |
| JM1 | Mean | 0.814 | 0.808 | 0.813 | 0.788 | 0.814 | 0.212 | 0.290 | 0.213 | 0.298 | 0.261 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.001 | 0.012 | 0 |
| KC1 | Mean | 0.842 | 0.793 | 0.795 | 0.801 | 0.711 | 0.391 | 0.572 | 0.591 | 0.502 | **0.754** |
| | Var | 0 | 0.001 | 0.001 | 0.001 | 0.002 | 0 | 0.007 | 0.011 | 0.011 | 0.004 |
| KC3 | Mean | 0.859 | 0.857 | 0.860 | 0.850 | 0.853 | 0.511 | 0.544 | 0.556 | 0.500 | **0.622** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.009 | 0.013 | 0.014 | 0.014 | 0.028 |
| MC1 | Mean | 0.943 | 0.989 | 0.901 | 0.991 | 0.888 | 0.435 | 0.270 | 0.678 | 0.261 | 0.661 |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.028 | 0.003 | 0.006 | 0.003 | 0.007 |
| MC2 | Mean | 0.762 | 0.766 | 0.758 | 0.774 | 0.740 | 0.435 | 0.470 | 0.600 | 0.517 | **0.612** |
| | Var | 0.001 | 0.001 | 0 | 0.004 | 0.003 | 0.023 | 0.020 | 0.033 | 0.006 | 0.041 |
| MW1 | Mean | 0.842 | 0.584 | 0.815 | 0.782 | 0.757 | 0.500 | 0.320 | 0.500 | 0.300 | **0.560** |
| | Var | 0 | 0.020 | 0.005 | 0.001 | 0.006 | 0.015 | 0.007 | 0.005 | 0.015 | 0.023 |
| PC1 | Mean | 0.893 | 0.624 | 0.880 | 0.798 | 0.833 | 0.320 | 0.208 | 0.352 | 0.200 | **0.488** |
| | Var | 0 | 0.023 | 0 | 0.002 | 0.001 | 0.006 | 0.003 | 0.001 | 0.002 | 0.010 |
| PC2 | Mean | 0.969 | 0.741 | 0.928 | 0.941 | 0.870 | 0.525 | 0.325 | 0.525 | 0.200 | **0.625** |
| | Var | 0 | 0.014 | 0.009 | 0 | 0.004 | 0.019 | 0.044 | 0.011 | 0.028 | 0.023 |
| PC3 | Mean | 0.481 | 0.757 | 0.732 | 0.796 | 0.706 | 0.894 | 0.630 | 0.751 | 0.570 | 0.830 |
| | Var | 0.018 | 0.001 | 0.014 | 0.002 | 0.003 | 0 | 0 | 0.013 | 0.013 | 0.002 |
| PC4 | Mean | 0.881 | 0.772 | 0.842 | 0.841 | 0.790 | 0.181 | 0.525 | 0.601 | 0.532 | **0.722** |
| | Var | 0 | 0.008 | 0.005 | 0 | 0.002 | 0.004 | 0.007 | 0.024 | 0.02 | 0.005 |
| PC5 | Mean | 0.974 | 0.947 | 0.973 | 0.936 | 0.891 | 0.535 | 0.802 | 0.579 | 0.716 | **0.916** |
| | Var | 0 | 0 | 0 | 0 | 0 | 0.001 | 0 | 0.002 | 0.027 | 0.002 |

In TABLE VIII, the weak learner is KNN (K=3). When the models are applied on the 12 datasets, SMBM accounts 8 highest recall results (66.7%) based on datasets: KC1, KC3, MC2, MW1, PC2, PC4, and PC5. ADAM 2 highest recall results (16.7%) and the other models less. The ACC values of all models are very close on all datasets and the variance of SMBM's ACC values is 0.793.

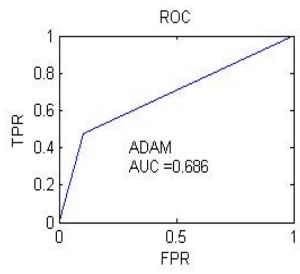


Figure 2.1 ADAM ROC and AUC

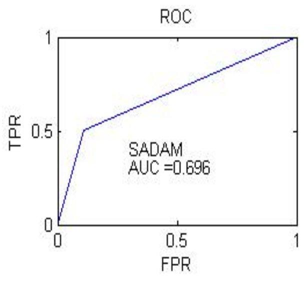


Figure 2.2 SADAM ROC and AUC

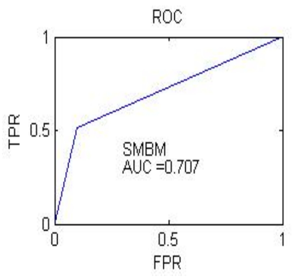


Figure 2.3 SMBM ROC and AUC

Figure 2. The ROC of Models Based on C4.5 on KC1.

From the ROC of models based on C4.5 on KC1 results in Figure 2.1-2.3, the AUC values of the three models (ADAM, SADAM, SMBM) are 0.686, 0.696, 0.707 and SMBM gets the highest AUC value.

TABLE IV-TABLE VIII shows the performance of the five prediction models (measured using ACC and Recall) for each of the datasets used in our experiments. In most cases, the ACC values of SMBM are close to the other models. And SMBM's ACC values are around 0.800, which means this model has a good performance over the entire classes. Without sacrificing the accuracy over the entire data sets, SMBM accounts 36 highest recall values in the all 60 (60.0%). ADAM accounts 23 (38.3%) and the others less. From all tables, different weak learner causes different results. From TABLE IV to TABLE VIII, the number of SMBM's highest recall results are: 6 (50.0%), 5(41.7%), 8(66.7%), 9(75%), 8(66.7%), while the number of ADAM are: 6 (50.0%), 6(50.0%), 4(33.3%), 5(41.7%), 2(16.7%) and the others' are less. Except that SMBM's recall result is worse than ADAM's in TABLEV, SMBM's recall results are better than the other 4 models. Therefore, SMBM has a better performance on the minority fault-proneness prediction than the rest models. The SINM and SSINM have higher accuracy values, but lower recall values on many datasets employed in our study, such as in CM1and PC2, which shows that the two models have a poor predictive accuracy over the minority class. The results of ADAM are close to the SMBM's, and it is recall values are slightly lower than the ones of SMBM, such as on KC1, PC4, which means ADAM has a good performance on imbalanced datasets. SADAM has general ability to predict minority fault-proneness.

Figure 2 reveals the SMBM has a higher AUC value than the other two models and the value is 0.707, which means SMBM has a better performance on the prediction of imbalanced datasets.

Therefore, the SMBM has a better performance than the other models in predicting the imbalanced high-risk minority software faults with higher recall and AUC values.

## VI. Conclusions

In this paper, we make a comprehensive investigation on the performances of the five types of prediction models with different weak learners on the MDP datasets that contains different degrees of imbalance and different sizes, thus providing a diverse test bed. The comparative experiment results present that the SMOTEBoost model has a good ability for imbalanced software fault-proneness prediction tasks, particularly on the minority class samples, which means the prediction model of SMOTEBoost is better to guide software testing resource allocation and improving software reliability more likely. The ADABoost model also has a certain capacity for imbalanced software fault-proneness prediction tasks.

SMOTEBoost algorithm combines the advantage of Boosting procedure and SMOTE algorithm. Boosting procedure is used to ensure the accuracy on the entire data sets. By introducing SMOTE algorithm into each iteration of Boosting, each weak learner is able to sample more of the minority class, and also better and broader decision regions for the minority class. In this way, SMOTEBoost model has a better ability of prediction on the imbalanced software fault-proneness.

Manually finding bugs from complicated software is an expensive work for software testing engineers. This causes a huge challenge on collecting the large amount of training samples for building a predictor. In other words, establishing a large software metric set with fault information is really expensive and time consuming. To address this issue, the semi-supervised learning technique will be introduced to expand the range of training data so that we can employ and effectively increase the performance of predictor with limited fault information found. This will bring a greater flexibility to modeling and a bigger range of application of the model, and further improving the prediction performance.

As we all know, it is unavoidable for people to make some mistakes, particularly when doing a heavy work, such as software testing. In other words, it is hard to guarantee that noise free in the bugs searching process. Therefore there is a very high possibility for some noise samples to be involved in the collection of software fault information or metrics. In our research plan, a noise tolerance learning mechanism will be introduced into ensemble predictor to help prevent the damage on predictive performance from noise samples.

## References

[1] Lions J L. Ariaen-5 flight 501 inquiry board report [EB/OL]. *http://sunnyday. mit. edu/accidents/ Ariane 5 accident report.html*, 1996-02-05.

[2] Gregory T. The Economic Impacts of Inadequate Infrastructure for Software Testing [R].RTI Health, Social, and Economic Research, RTI Protect Number: 7007.011,2002.

[3] Gang Ye, Xianjun Li, Dan Yu, Zhongwen Li, Jie Yin, ”The Design and Implementation of Workflow Engine for Spacecraft Automatic Testing,” Journal of Computers; Jun2011, Vol. 6 Issue 6, p1145.

[4] Aggarwal K.K., Singh Y., Kaur A., Malhotra R. (2008). Empirical Analysis for Investigating the Effect of Object Oriented Metrics on Fault Proneness: A Replicated Case Study, Forthcoming in Software Process Improvement and Practice, Wiley.

[5] Iker Gondra, Applying machine learning to software fault-proneness prediction, Department of Mathematics, Statistics, and Computer Science, St. Francis Xavier University, P.O. Box 5000, Antigonish, Canada NS B2G 2W5,Available online 7 June 2007

[6] Menizes T, Dekhtyar A, Distefano J, Greenwald J, Problems with Precision: A Response to “Comments on ‘Data Mining Static Code Attributes to Learn Defect Predictors’” [J]. IEEE Trans. Softw, 2007, 33(9):637-640.

[7] Tan P N, Steinbach M, Kumar V. Introduction to Data Mining [M]. New York: Addison Wesley, 1st edition, 2005

[8] Xin Jin, Yujian Li, Yihua Zhou, Zhi Cai, “Applying Average Density to Example Dependent Costs SVM based on Data Distribution,” Journal of Computers, Vol 8, No 1 (2013), 91-96, Jan 2013. doi: 10.4304/jcp.8.1.91-96.

[9] N. V. Chawla, K. W .Bowyer, L. O. Hall, and W. P. Keglmeyer, “SMOTE: Synthetic Minority Over-Sampling Technique,” Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, June 2002.

[10] Yoav Freund, Robert E. Schapire. A Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence,14(5):771-780,Sept,1999.

[11] Nitesh V. Chawla, Ar Lazarevic , Lawrence O. Hall , Kevin W. Bowyer, “SMOTEBoost: improving prediction of the minority class in boosting,”7th European Conference on Principle and Practice of Knowledge Discovery in Database (PKDD), pp. 107 to 119 ,Dubrovnik, Croatia, 2003.

[12] NASA/WVU VI&V Facility. Metrics data program. http://mdp.ivv.nasa.gov.

[13] Yoav Freud and Robert E. Schapire ,”Experiments with a New Boosting Algorithm,” 1996.

[14] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers,1993.

[15] V. N. Vapnik, An Overview of Statistical Learning Theory, IEEE Trans. Neural Netw. 1999, 10, 988-999.

[16] Palei, S. K.; Das, S. K. (2009). "Logistic regression model for prediction of roof fall risks in board and pillar workings in coal mines: An approach". Safety Science 47: 88. doi:10.1016/j.ssci.2008.01.002

[17] Gregory Shakhnarovich, Trevor Darrell, Piotr Indyk (2006). Nearest-Neighbor Methods in Learning and Vision. MIT Press.

[18] I.Rish. An empirical study of the NaiveBayes classifier.

[19] A. Estabrooks, T. Jo, and N. Japkowicz, “A Multiple Resampling Method for Learning from Imbalanced Data Sets,” Computational Intelligence, vol .20, 2004.

[20] Martha Varguez-Moo, Francisco Moo-Mena, Victor Uc-Cetina, “Use of Classification Algorithms for Semantic Web Services Discovery,” Journal of Computers, Vol 8, No 7 (2013), 1810-1814, Jul 2013. doi: 10.4304/ jcp.8.7.1810 -1814.

[21] A.A.Shahrjooi Haghighi, M.Abbasi Dezfuli. Applying Mining Schemes to Software Fault Prediction: A Proposed Approach Aimed at Test Cost Reduction, Proceedings of

the World Congress on Engineering 2012 Vol I, WCE 2012, July 4-6, 2012, London, U.K.

[22] Akalya C., Kannammal K.E., Surendiran B. A Hybrid Feature Selection Model for Software Fault Prediction [J]. International Journal on Computational & Application, 2012, 2(2):25-35

[23] Catal C., Diri B. Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem [J]. Information Sciences, 2009, 179(8): 1040-1058

[24] Lessmann S, Baesens B, Mues C, Pietsch, S, Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings [J]. IEEE Trans. Softw. Eng. 2008, 34(4): 485-496.

[25] Jiang Y, Cukic B, Ma Y. Techniques for Evaluating Fault Prediction Models [J]. Empirical Software Engineering, 2008, 13(5):561-595.

[26] Ling C X, Huang J, Zhang H. AUC: A Better Measure Than Accuracy in Comparing Learning Algorithms[C]//In : proc. ofartificial intelligence, Canad, 2003.

[27] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques. Morgn Kaufmann, San Francisco, California, 2nd edition, 2005.

[28] Chris Seiffert, Taghi M. Khoshgoftaar, “Resampling or Reweighting: A Comparison of Boosting Implementations,” 2008 20th IEEE International Conference on Tools with Artificial Intelligence.

**Renqing Li** received the B.Eng. degree in the school of Reliability and System Engineering from Beihang University, Beijing, in 2013.

**Shihai Wang** received his Ph.D. in Computer science from the University of Manchester UK at 2010. He joined the school of Reliability and System Engineering, Science and Technology on Reliability and Environmental Engineering Laboratory Beihang University, as a lecturer from 2011. Currently his research interests include software testing, software fault prediction and pattern recognition and application in software reliability.