Empirical Studies of a Two-Stage Data Preprocessing Approach for Software Fault Prediction

Wangshu Liu, Shulong Liu, Qing Gu, *Member, IEEE*, Jiaqiang Chen, Xiang Chen, *Member, IEEE*, and Daoxu Chen, *Member, IEEE*

***Abstract*—Software fault prediction is a valuable exercise in software quality assurance to best allocate limited testing resources. Classification is one of the effective methods for software fault prediction. The classification models are trained based on the datasets obtained by mining software historical repositories. However, the performance of the models depends on the quality of datasets. In this paper, we propose a novel two-stage data preprocessing approach which incorporates both feature selection and instance reduction. Specifically, in the feature selection stage, we first perform relevance analysis, and then propose a threshold-based clustering method, called novel threshold-based clustering algorithm, to conduct redundancy control. In the instance reduction stage, we apply random under-sampling to keep the balance between the faulty and non-faulty instances. In empirical studies, we chose datasets from real-world software projects, such as Eclipse and NASA. Then we compared our approach with some classical baseline methods, and further investigated the influencing factors in our approach. The final results demonstrate the effectiveness of our approach, and provide a guideline for achieving cost-effective data preprocessing when using our two-stage approach.**

***Index Terms*—Software fault prediction, data preprocessing, feature selection, instance reduction, redundancy control, feature ranking, classification model.**

Acronyms and Abbreviations

SFP	Software fault prediction
FP	Fault-prone
NFP	Non-fault-prone
NTC	Novel threshold-based clustering algorithm
RUS	Random under-sampling
ROS	Random over-sampling
X + Y	Feature selection stage of our approach in which X is used as the relevance measure, and Y as the similarity measure
X + Y + Z	X + Y is the feature selection stage as the above described, and Z is the another instance reduction stage
ALL	Included all original feature set
CFS	Correlation-based feature selection
Consist	Feature selection method Consistency
ROC	Receiver operating characteristic curve
AUC	Area under ROC measure

Notation

IG	Information gain measure
CS	Chi-Square measure
RF	ReliefF measure
SU	Symmetric uncertainty measure
COS	Cosine similarity measure
T	The original given dataset
f_i	The ith feature belongs to the dataset
l	The class label
β	The similarity threshold for feature clustering
F_{out}	The final feature set
m	The number of original features
$FF\_Sim()$	The similarity between any pair of two features
C_i	The neighbor cluster of the feature f_i
AS_i	The average similarity of C_i

Manuscript received September 01, 2014; revised December 25, 2014 and April 09, 2015; accepted July 26, 2015. Date of publication August 10, 2015; date of current version March 01, 2016. This work was supported in part by National Basic Research Program of China (Grant No. 2009CB320705), National Natural Science Foundation of China (Grant Nos. 61373012, 61321491, 91218302, 61202006), The University Natural Science Research Project of Jiangsu Province (Grant No. 12KJB520014), The Open Project of State Key Laboratory for Novel Software Technology at Nanjing University (Grant No. KFKT2012B29), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Associate Editor: W. E. Wong. (*Corresponding author: Qing Gu.*)

W. Liu, S. Liu, Q. Gu, J. Chen, and D. Chen are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China (e-mail: liuws0707@gmail.com; lshl0107@163.com; guq@nju.edu.cn; chensonglin1989@gmail.com; cdx@nju.edu.cn).

X. Chen is with the School of Computer Science and Technology, Nantong University, Nantong, China (e-mail: xchencs@ntu.edu.cn).

Digital Object Identifier 10.1109/TR.2015.2461676

I. Introduction

SOFTWARE fault prediction (SFP) is a hot research topic in the domain of software engineering [1]–[17]. It can allocate the limited test resources effectively by predicting the fault proneness of software modules. Classification is one of the prevalent methods used for software fault prediction. Its main task is to categorize modules, represented by a set of software metrics, into two classes: fault-prone (FP), or non-fault-prone (NFP). For a specific classification model, the classifier should be trained in advance based on the training data obtained by mining historical software repositories, such as change logs in software configuration management, bug reports in bug tracking systems, and the e-mails of developers.

In recent years, researchers have found that the quality of software datasets had serious effect on the performance of predicting software faults. Issues concerned in data quality include biased datasets [18]–[21], noise [22], [23], a large number of features [24]–[26], and class imbalance [17], [27], [28]. In this paper, we focus on the latter two aspects to improve the quality of software datasets.

One is the high dimensionality problem caused by too many unnecessary features (i.e., software metrics), and the other is the class imbalance problem caused by superfluous instances of some classes. The former can be solved by feature selection, which selects a small fraction of the features by removing irrelevant or redundant ones. The latter is handled by instance sampling (or reduction), which samples a subset of the instances from majority classes. Both have been proven useful by previous experimental studies [4], [29]–[31].

However, to the best of our knowledge, few researchers have combined feature selection and instance reduction simultaneously to improve data quality in software fault prediction. In this paper, we provide a novel two-stage data preprocessing approach. In particular, for feature selection, we first perform relevance analysis to remove irrelevant features, and then propose a threshold-based clustering algorithm for redundancy control to eliminate redundant features. For instance reduction, we apply random under-sampling to reduce the NFP instances to keep balance between the two classes.

We design experiments based on real-world software projects to demonstrate the effectiveness of our approach. Based on the experimental studies, we found that the two-stage approach can greatly reduce both the number of features and the number of instances of original datasets, without sacrificing the prediction performance of the classifier built after. Compared with other commonly used data preprocessing methods, our approach has presented consistently better performance over different datasets and classification models.

This paper extends our previous research work [16] by incorporating more datasets from real-world projects, feature relevance measures, feature similarity measures, as well as by making more extensive empirical studies. The main contributions of the paper can be summarized as follows.

- To the best of our knowledge, we propose a novel two-stage data preprocessing approach, which performs both feature selection and instance reduction in sequence, for SFP.
- In the feature selection phase, we propose a novel algorithm using both feature selection and threshold-based clustering for relevance analysis and redundancy control. Then, in the instance reduction phase, we apply random under-sampling.
- We use real-world projects to conduct empirical studies to verify the effectiveness of our proposed approach, and further investigate the influencing factors in our approach to provide a guideline for effectively using our approach.

The remainder of this paper is organized as follows. Section II introduces background and related work on data preprocessing for software fault prediction. Section III describes our proposed approach in detail, including the framework, and the methods used for either feature selection or instance reduction. Section IV describes the experimental design, including the subject datasets, and the performance measure. Section V performs result analysis, and discusses some threats to validity. Finally, Section VI concludes this paper, and discusses plans for future work.

II. Related Work

In this section, first we briefly introduce the background of software fault prediction. Second, we describe some related work in data preprocessing, especially on feature selection and instance reduction.

Software fault prediction is valuable to predict the fault-proneness of software modules. Numerous researchers have successfully used classification models to categorize software modules into faulty and non-faulty, based on the features (i.e., metrics) collected from each module by mining software development repositories [1]–[17]. These classification models require training data collected from previous projects where faulty modules have been identified. Evidence [18]–[23], [28], [32] shows that the quality of the datasets is essential for high prediction performance.

The quality of software datasets can be improved by data preprocessing [25]–[27], which includes feature selection and instance reduction (or sampling). Feature selection is the process of identifying and removing irrelevant and redundant features from a dataset, so that only beneficial features are left for training the classification models. A variety of feature selection methods have been developed, which can be roughly grouped into two categories: *filter-based*, and *wrapper-based*. Filter-based methods evaluate and select the most relevant features, based on the correlation between the features and class labels. Wrapper-based methods require feedback from the classification model, and compose the feature set iteratively, which may lead to high computational complexity [33].

Researchers have made comparisons among available feature selection methods for software fault prediction. Shivaji *et al.* [26] evaluated five feature selection methods (including three filter-based ranking methods, and two wrapper-based ones) using two different classification models (i.e., Naive Bayes, and SVM). Their experimental results showed that all the feature selection methods were effective to improve the prediction performance, while the extent of improvement was more evident between the two classifiers than among the feature selection methods. Gao *et al.* [25] compared feature selection methods in predicting faulty software modules of a large legacy telecommunication system. They employed seven filter-based methods, and three wrapper-based ones implemented by search based greedy techniques. Their experimental results demonstrated that removing 85% of the software features did not decrease the prediction accuracy, and in some cases did even better. Wang *et al.* [24] presented a comprehensive empirical study evaluating 17 ensembles of 18 feature ranking methods. Their results suggested that the ensemble of a few (e.g., 2 to 4) rankers may improve the prediction performance.

Instance reduction is the process of selecting a representative subset of the instances (i.e., software modules) to build the training data for a classification model [29]. Recently, instance reduction has drawn more attention by both researchers and practitioners for software fault prediction [27]. By removing superfluous instances, the problem of inter-class imbalance can be alleviated. One efficient method to handle instance reduction is random sampling, which is effective in minimizing the impact of imbalanced distributions among classes [34]. Wang *et al.* [28] investigated different types of instance sampling methods before the training of multiple classification models for software fault prediction. The results showed that, for highly imbalanced data, the combination of random under-sampling and Naive Bayes could be a good option. Pelayo *et al.* [35] compared random under-sampling and over-sampling methods using 6 software datasets. By statistical analysis, they proved that under-sampling was useful in improving the prediction performance of the classifiers trained afterward. Khoshgoftaar *et al.* [36] discussed the effects of random sampling combined with other data preprocessing methods (including feature ranking). Their results also proved the effectiveness of random sampling in dealing with imbalanced datasets.

While both feature selection and instance reduction are effective in improving the performance of classification models, to the best of our knowledge, few researchers have combined them in handling data preprocessing for software fault prediction. Liu *et al.* [37] combined feature selection with instance sampling. But the purpose of the instance sampling was to reduce the total number of instances instead of handling class imbalance. Khoshgoftaar *et al.* [36] combined filter-based feature ranking methods and random under-sampling. Their purpose was to use instance sampling for selecting features iteratively, but they did not train the classifier with balanced data. Different from their work, we designed a new approach for data preprocessing, which involved both feature selection and instance reduction, and built the classification model after the data preprocessing. Specifically, for feature selection, we made a relevance analysis to remove irrelevant features, and proposed a threshold-based clustering algorithm for redundancy control to eliminate redundant features.

III. The Two-Stage Data Preprocessing Approach

In this section, we present our two-stage data preprocessing approach for software fault prediction, which combines feature selection and instance reduction to eliminate both irrelevant and redundant units in software datasets. In particular, for the feature selection, we propose an improved version of our previous method [38]. First, feature ranking is used to eliminate irrelevant features. Second, a novel threshold-based clustering algorithm is used to remove redundant features. For the instance reduction, because the similarity among NFP instances (i.e., software modules) is usually high (which will be demonstrated in Section V), random sampling is applied to reduce the NFP instances. By the above two-phase preprocessing, we could get a balanced, high quality dataset for training the classification models, which would improve the performance of fault prediction.

A. The Framework of Our Approach

Fig. 1 gives the framework of our two-stage approach. Specifically, the feature selection stage aims to eliminate irrelevant and redundant features, and then the instance reduction stage aims

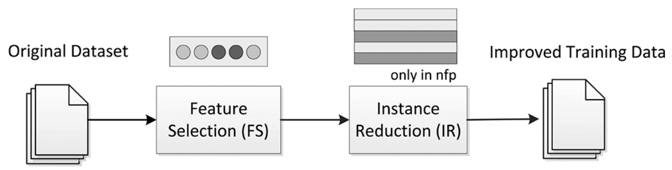


Fig. 1. The Framework of the Two-stage Approach.

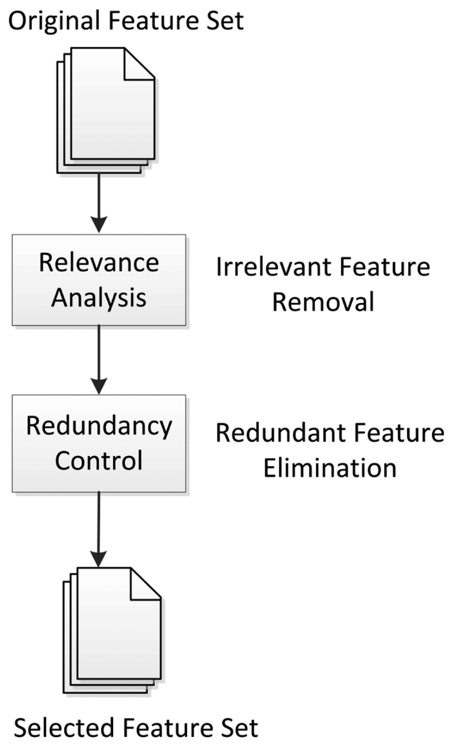


Fig. 2. The Process in the Feature Selection Stage.

to reduce the NFP instances, so that the balance of the dataset can be achieved.

In the framework, the feature selection (FS) stage performs feature ranking and threshold-based clustering in sequence, while the instance reduction (IR) stage uses random sampling to reduce the NFP instances. Here, FS is performed before IR (denoted as FS $\rightarrow$ IR) in our approach. For feature selection, we need to keep as much information as possible to select the valuable features, and after eliminating instances the ranking of the features may be changed. On the other hand, as random sampling is used for instance reduction in our approach, the effect of feature selection on it can be ignored.

B. The Feature Selection Stage

Fig. 2 gives the process in the feature selection stage. The first step is performing relevance analysis to remove irrelevant features, and the second step is performing redundancy control to eliminate redundant features. Feature ranking is used for the first step, where individual features are ranked according to the importance (i.e., weights) in differentiating instances of different classes (i.e., FP or NFP). The subset of features from the top-k of the ranking list is selected. Here, k is set to select $\lceil \sqrt{m} \times \lg m \rceil$ features from the total set, as recommended by Song *et al.* [39], where m is the number of features in the original data set.

To weight the relevance of a feature to the class, we should measure the correlation between them. The commonly used measures can be categorized into three groups: entropy based (such as information gain, gain ratio, and symmetric uncertainty), statistical based (such as chi-square), and instance based (such as Relief and ReliefF). In our method, we choose one representative from each group, each of which has been proven good in software fault prediction [42]. These measures are information gain (IG), chi-square (CS), and ReliefF (RF).

1) Information Gain (IG): Given a feature f, IG measures the amount of information f can provide about whether an instance is FP or NFP [9]. The IG measure can be computed by the following formula.

$$IG(X|Y) = H(X) - H(X|Y) \tag{1}$$

In (1), $H(X)$ computes the entropy of a discrete random variable X (i.e., the class). Let $p(x)$ denote the prior probability of a value x of X. $H(X)$ can be computed by the following formula.

$$H(X) = -\sum_{x\in X} p(x)log_2 p(x) \tag{2}$$

$H(X|Y)$ computes the conditional entropy which quantifies the uncertainty of X given the observed variable Y (i.e., the feature). Let $p(x|y)$ denote the posterior probability of x given the value y of Y. $H(X|Y)$ can be computed by the following formula.

$$H(X|Y) = -\sum_{y\in Y} p(y) \sum_{x\in X} p(x|y)log_2 p(x|y) \tag{3}$$

2) Chi-Square (CS): Given a feature f, Chi-Square is a non-parametric statistical measure that examines the correlation between the distribution of f and that of the class [43]. We use the Pearson Chi-Square statistic, which can be computed by the following formula.

$$CS = \sum_{i=1}^{r}\sum_{j=1}^{n_c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{4}$$

In (4), r is the number of distinct values of the feature, n_c is the number of classes (here n_c is 2), $O_{i,j}$ is the observed number of instances with the feature value i in the class j, and $E_{i,j}$ is the expected (i.e., mean) number of instances with any value i in the class j.

3) ReliefF (RF): ReliefF can measure how well a feature differentiates instances from different classes by searching for the nearest neighbors of the instances from both the same and different classes [44]. The ReliefF measure can be computed by the following formula.

$$RF(X) = \frac{1}{m}\sum_{i=1}^{m}[diff(x_i, x_M) - diff(x_i, x_H)] \tag{5}$$

In (5), X is the random variable representing a feature, m is the total number of instances sampled, and x_i is the feature value of the ith instance. For the instance i, both its nearest neighbor from the same class, and the nearest neighbor from the other class, are determined first. Then the differences between the feature values (i.e., $diff(x_i, x_H)$ for the former, and $diff(x_i, x_M)$ for the latter) are computed. The function $diff()$ is defined by the following formula, where x_{max}, and x_{min} stand for the maximum, and minimum values of X respectively.

$$diff(X_1, X_2) = \frac{|x_1 - x_2|}{x_{max} - x_{min}} \tag{6}$$

All the three measures (i.e., IG, CS, and RF) range from 0 to 1. The greater the measure, the more relevant the feature is to the class.

The novel threshold-based clustering algorithm (NTC) is used for the second step. Algorithm 1 shows the details of NTC. At first, features are grouped into clusters using a pre-specified threshold β. To cluster these features, the algorithm starts by computing the similarity between each pair of features, and constructs a β-nearest neighbor graph over the features. The nodes of the graph correspond to the features, and a link between two features f_i and f_j exists iff the similarity is no less than β. After that, the feature which has the most compact neighborhood (i.e., the average similarity of its neighbor set is the highest) is selected, while all the remaining neighbors are removed. Finally, the graph is reconstructed without the selected feature and its neighbors. This procedure is repeated until all the features are either selected or removed. The methods for computing the similarity between any pair of features will be described in the following. Then the optimal setting of β, which needs to be considered for NTC, will be discussed later in Section V.

To measure the similarity between any pair of features, we choose the non-linear similarity measure Symmetric Uncertainty (SU) [45], and the commonly used linear similarity measure Cosine Similarity (COS).

1) *Symmetric uncertainty (SU):* Symmetric uncertainty [45] measures the entropy coefficient between two random variables, and has been used to evaluate the similarity between features by many researchers [39], [46], [47]. The paper [48] approved that SU can compensate for the information gain's bias toward variables (i.e., features) with more values, and restrict its values to the range [0,1]. SU can be computed by the following formula.

$$SU(X,Y) = 2 \times \frac{IG(X|Y)}{H(X) + H(Y)} \tag{7}$$

In (7), X and Y denote the two features. The functions $IG(X|Y)$, and $H()$ are defined in (1), and (2) respectively.

2) *Cosine similarity (COS):* Cosine similarity measures the cosine of the angle between two Euclidean vectors, constructed by the values of the two features according to the instance order [49]. One of the merits of COS is that the scaling factor has no effect on it. COS can be computed by the following formula.

$$COS(X,Y) = \frac{\sum x_i y_i}{\sqrt{\sum {x_i}^2}\sqrt{\sum {y_i}^2}} \tag{8}$$

In (8), X and Y denote the two features, while x_i and y_i are the feature values in the i^{th} instance for features X and Y.

Both the SU and the COS measures range from 0 to 1. The greater the measure, the more likely the two features are similar.

In Algorithm 1, $FF\_Sim()$ computes the similarity between any pair of features, C_i denotes the neighbor set (i.e., cluster) of the feature f_i, $\boldsymbol{C} = \{C_1, C_2, \ldots, C_k\}$ is the set of constructed

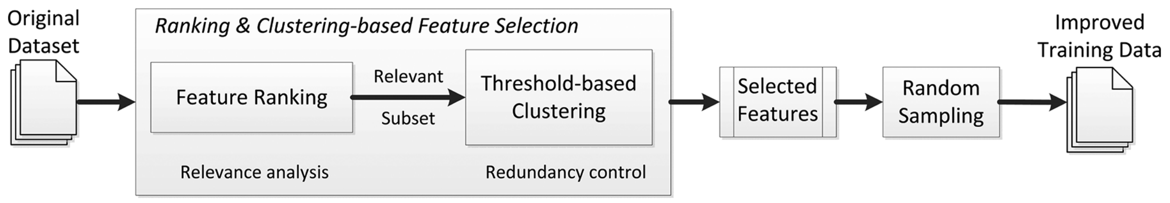


Fig. 3. The Details of the Two-stage Data Preprocessing Approach.

Algorithm 1: The Algorithm NTC

```
Input:
The dataset $T(f_1, f_2, \cdots, f_k, l)$, where $l \in \{FP, NFP\}$
denotes the class label, and $k$ is the number of features
selected by feature ranking
The similarity threshold $\beta$ for feature clustering
Output:
The final feature set $F_{out}$
1  $F_{out} = \varnothing$
2  for each pair of features $(f_i, f_j)$ in $T$ do
3      Computing $FF\_Sim(f_i, f_j)$
4  for i=1 to $k$ do
5      $C_i = \varnothing$
6      for each feature $f_j$ $(1 \le i \le k$ and $i \ne j)$ do
7          if $FF\_Sim(f_i, f_j) \ge \beta$ then
8              Adding $f_j$ into $C_i$
9  $\boldsymbol{C} = \{C_1, C_2, \cdots, C_k\}$
10 while $\boldsymbol{C}$ is not empty do
11     for each $C_i \in \boldsymbol{C}$ do
12         $AS_i = \frac{\sum_{f_j \in C_i} FF\_Sim(f_i, f_j)}{|C_i|}$
13     $C_{i'} = \arg\max_{C_i \in \boldsymbol{C}} AS_i$
14     $F_{out} = F_{out} \cup \{f_i\}$
15     $\boldsymbol{C} = \boldsymbol{C} \setminus C_{i'}$
16     Updating all the rest clusters in $\boldsymbol{C}$ (i.e., removing
       both $f_{i'}$ and features in $C_{i'}$)
17 return $F_{out}$
```

(or remained) clusters in the nearest neighbor graph, and AS_i denotes the average similarity of C_i. The algorithm is rather effective, and can be optimized. The most time-consuming part is the computation of the similarities between pairs of the features. But the similarity computation is commutative, and at most $k(k-1)/2$ times are required.

C. The Instance Reduction Stage

In the instance reduction stage, we apply random sampling to reduce the NFP instances. Random sampling is a simple but effective technique for instance reduction [40]. It may bear the risk of losing information by removing valuable instances, because all the instances are treated similarly. However, during our experiments, we found that the similarities among the NFP instances are usually higher than 0.9 on average, no matter which function is used to compute them. Because the datasets used in our experiments are commonly used by other researchers, we think it is reasonable to reduce the NFP set by random sampling without replacement, which leads to the simplest implementation. To keep balance between the FP and NFP classes, we set the FP/NFP ratio as the terminal condition of sampling. In the experiments, the FP/NFP ratio is set to 35%/65% according to the recommendation by Khoshgoftaar *et al.* [41].

In summary, Fig. 3 shows the details of our proposed approach.

IV. Experimental Setup

In this section, we design experiments to demonstrate the effectiveness of the two-stage data preprocessing approach. First, we design the research questions for the empirical study. Second, we describe the datasets used in the experiments. Third, we put forward the experimental design based on the research questions. Last we introduce the performance measure used in our research.

A. Research Questions

The empirical study is performed in two steps: one is to validate whether the novel threshold-based clustering algorithm NTC of the feature selection stage can improve the performance of the classification models built after. The other is to testify whether the instance reduction stage is valuable in enhancing further the final prediction performance. Hence, we design experiments to study the following five research questions.

- RQ1. In the feature selection stage, what is the optimal scheme of our approach, concerning the relevance functions, the similarity functions in Section III-B, and the similarity threshold β in Algorithm 1, to result in good-enough performance of the classification models built after?
- RQ2. Is our two-stage approach better to improve the performance of the classification models built after, compared with other commonly used feature selection methods?
- RQ3. Taking the feature selection stage alone, by adding the proposed algorithm NTC, can our method produce better performance compared with other feature selection methods?
- RQ4. Does combining the feature selection and instance reduction together have a more significant improvement than by using any one of them individually for dataset preprocessing?
- RQ5. What are the effects of the characteristics of individual datasets, including level of instance sufficiency, and temporal issues, on the final performance of the approach, and the selection of β?

B. Datasets

To evaluate the effectiveness of our proposed approach, we design and perform a series of experiments using datasets collected from real-world software projects, including the Eclipse projects, and the NASA software projects, which are commonly used by researchers in software fault prediction. In particular, the Eclipse projects are used in [50]–[54], and the NASA software projects are used in [1], [4], [6], [23], [55]–[58]. The Eclipse datasets are obtained from the PROMISE data repository, [59] while the NASA datasets are obtained from the publicly available MDP (Metrics Data Program) repository [60].

For the Eclipse datasets, three releases of Eclipse (i.e., Eclipse 2.0, Eclipse 2.1, and Eclipse 3.0) are collected with instances measured at the Java class level. Each of the Eclipse datasets contains 198 features, including the code complexity measures (such as LOC, cycloramic complexity, and number of classes), the syntax tree based measures, and many others [50]. Before the experiments, we perform the following treatments to the datasets.

1) Remove all the non-numeric measures. 2) Transform the post-release faults measure (which counts the number of faults in the post release versions) into the binary class label. In particular, those containing one or more faults are labeled as FP, whereas those with zero faults are labeled as NFP. 3) Remove the measures which have only one distinct value. After performing these treatments, each Eclipse dataset leaves 155 features.

For the NASA datasets, the measures collected from each project are different from one another, while the commonly used measures include LOC, Halstead complexity metrics, and McCabe complexity metrics [4]. The instances are generally measured at the method level. Before the experiments, we select 10 out of the 13 datasets from the MDP repository. The selection is based on two principles: one requires the dataset containing enough features (e.g., >10), which makes feature selection sensible; the other requires enough ratio of fault instances to total (e.g., >1%), which makes instance reduction and classification meaningful. For the selected datasets, we remove measures which have only one distinct value. Table I lists the statistics of all the selected datasets used in the experiments.

TABLE I
THE DETAILS OF THE SELECTED DATASETS

Dataset	Level	#Features	#Instances	#FP Instances
Eclipse 2.0	Class	155	6729	975(14.49%)
Eclipse 2.1	Class	155	7888	854(10.83%)
Eclipse 3.0	Class	155	10593	1568(14.80%)
cm1	Method	37	505	48(9.5%)
kc1	Class	86	145	60(41.37%)
kc3	Method	39	458	43(9.389%)
kc4	Method	14	125	61(48.8%)
mc2	Method	39	161	52(32.3%)
mw1	Method	37	403	31(7.69%)
pc1	Method	37	1107	76(6.87%)
pc3	Method	37	1563	160(10.24%)
pc4	Method	37	1458	178(12.21%)
pc5	Method	38	17186	516(3%)

C. The Experimental Design

To study the research questions, 6 different preprocessing schemes for the feature selection stage are designed according to the combinations of the relevance measures and similarity measures. Combined with options for the instance reduction stage, these schemes are applied with three different classification models to comprehensively study the effects on the performance of software fault prediction. All the experiment results are averaged over 10×10-fold cross validation, which means 10-fold cross validation repeated 10 times in each experiment. A 10-fold cross validation means that the dataset is equally divided into 10 parts, and instances of each part are used in turn as the testing set, while the remaining instances are used as the training set [2], [11], [22], [49], [61]. The performance measure is averaged over the 10 folds. To further overcome the effect of randomness, the 10-fold cross validation is repeated 10 times, and the grand average value over the 10 repetitions is used as the final performance measure.

1) Feature Selection Methods: To study the feature selection stage, besides our proposed method, which includes both feature ranking and threshold-based clustering, another four baseline methods are also implemented for comparison.

For feature selection, our method combines both relevance analysis and redundancy control to select the most relevant features while eliminating redundant ones. Feature ranking is used to implement the relevance analysis, while NTC is used to implement the redundancy control. During the experiments, for feature ranking, we set the number of pre-selected features k to be $\lceil \sqrt{m} \times \lg m \rceil$ (m is the number of original features), as suggested by Song *et al.* [39]. For NTC, the threshold β is set to result in a comparable size of the final feature set, to match those resulted by other feature selection methods. The optimal assignments of β will also be studied. To denote these different methods, we use both the relevance and the similarity measures, e.g., IG + COS means that IG is used as the relevance measure, and COS as the similarity measure.

For the baseline methods, first, we implement ALL, which means no feature selection is applied. Second, we implement pure feature ranking using the relevance measure IG (denoted as IG), which can outperform the other pure ranking techniques [62]. Here we select the top $\lceil log_2 m \rceil$ features, as suggested by Gao *et al.* [25]. Third, we implement two well-known feature selection methods other than ranking. One is Correlation-Based Feature Selection (CFS) [46], which selects features that are both highly relevant to the class labels and little correlated with each other. The other is Consistency (denoted as Consist) [63], which searches for the feature set that separates the classes consistently.

2) Instance Reduction Methods: For the instance reduction stage, we use random under-sampling (RUS), which randomly discards instances from the NFP class to build a more balanced dataset. To make a comparison, we also implement random over-sampling (ROS), which randomly duplicates instances in the FP class until the dataset reaches a balanced radio. The ratio of the #NFP class to the #FP class, which is set to about 65% (NFP) to 35% (FP), as recommended by Khoshgoftaar *et al.* [41], is used as the terminal condition of both the sampling methods.

Because the ordering of the instances may have significant effects on the sampling methods [64], which is not the subject

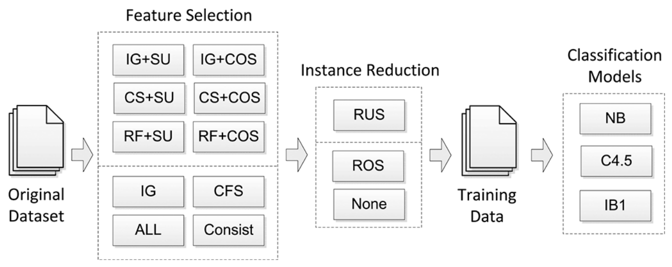


Fig. 4. The Configurations of the Empirical Study.

of this study, we perform 10-fold cross validation, and repeat it 10 times. Each time, the order of the instances is randomized first, so that the order effect in a dataset can be effectively obviated [1].

3) Classification Models: We implement three different classification models, which are commonly used in software fault prediction. The models are Naive Bayes (NB), C4.5 decision tree (C4.5), and k-Nearest Neighbors ($k = 1$, denoted as IB1). NB is a simple probabilistic classifier, which assumes the features are statistically independent of each other. It can provide good enough classification results, even though some of the features are inter-related [49]. C4.5 implements a decision tree [65], where nodes are built by the features according to the information entropy against the classes. IB1 uses a simple distance measure to vote appropriate features weighted by the (inverse) distances from the nearest neighbors [39]. In our experiments, we use the implementation of three classification models provided by the Weka package [49] to avoid external threats to validity.

Fig. 4 summarizes the configurations of our empirical study. For RQ1, we will discuss the optimal schemes of our approach in the feature selection stage. For RQ2, we will compare our two-stage approach to other feature selection methods. For RQ3, we validate if NTC is valuable for feature selection. For RQ4, we add the methods in the instance reduction stage, and validate if random sampling can further improve the prediction performance.

D. The Performance Measure

In the empirical study, leveraging the receiver operating characteristic (ROC) curve, we use the area under the ROC (AUC) measure to evaluate the performance of the classification models built on the datasets, which have been preprocessed by different approaches. The ROC curve characterizes the trade-off between the true positive rate and false positive rate obtained by a classifier, and illustrates the classifier's performance under a varied decision threshold (i.e., a value between 0 and 1 that separates the FP and NFP classes) [66]. AUC can provide a general indication of the predictive potential of the classifier [4], [67], and has been widely used to evaluate the performance of classifiers for software fault prediction. Evidence has shown that AUC has lower variance, and thus is more reliable than other performance measures, such as precision, recall, or F-measure [49]. AUC is a single-value measure that ranges from 0 to 1, where an AUC value closer to 1 means better performance.

V. Result Analysis

In this section, we study the five research questions based on the experimental results. After that, we give a brief discussion on the main findings. Finally we describe the possible threats to the validity of the empirical study.

A. RQ1: The Optimal Scheme of the Approach

To investigate the optimal scheme of our two-stage data preprocessing approach, we do experiments on all the combinations of the relevance (i.e., IG, CS, and RF) and the similarity measures (i.e., SU and COS). Totally, we have six different feature selection schemes, and all of them added by RUS for instance reduction.

The similarity threshold β used in the algorithm NTC is altered to investigate its effects on each scheme. By the way, the three classification models (i.e., NB, C4.5, and IB1) are trained after each scheme respectively, and the AUC measures are used to compare the performance.

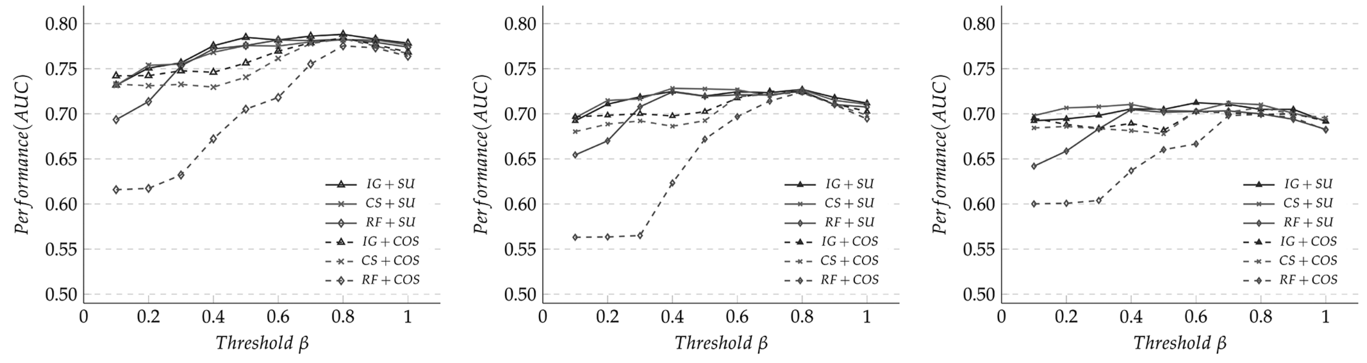
Fig. 5 shows the AUC measures of the six schemes using three different classifiers, when β is increased from 0.1 to 1.0 by steps of size 0.1. Note that, when β is set to 0, all the features will be removed by NTC, and as β is set higher, lesser features will be removed. β can be set to 1.0, which means only identical features (judged by the similarity measure) can be removed; and if β is set higher than 1.0, no feature will be removed, which means NTC is ignorable. In each subfigure, six lines with different colors and styles are used to represent different schemes. The dots on each line represent the mean values across the 13 datasets (Table I). For space consideration, the prediction results on singular datasets are not given.

From Fig. 5, we observe that, between the two similarity measures, SU always has a better, steadier performance than COS, no matter which one of the three relevance measures or the three classifiers is used. The schemes IG + SU, CS + SU, and RF + SU always outperform the counterparts using COS, and the classifiers make little difference. Considering the threshold β, a scheme X + SU (X denotes one of the relevance measures) always comes close to the best performance, when β is still small, while X + COS usually needs a great β value, which means more features are required.

On the other hand, among the three relevance functions, IG + SU, and CS + SU have more stable performance than RF + SU, over the settings of β, and the three classifiers. The only exception is in NB; when β is between 0.4 and 0.7, RF + SU outperforms CS+SU. However, when β is greater, CS+SU performs better again. The possible reason for the poor performance of RF is that RF already maximizes the differences among the selected features, which makes NTC less effective.

From Fig. 5, considering the threshold β used in NTC, we find that, on the whole, a greater β value will result in a better prediction performance with few exceptions, for all the three classifiers trained after the six schemes. The AUC measure will become steady when β reaches a big enough value, e.g., 0.8 in NB or C4.5, and 0.7 in IB1. However, if only taking SU into consideration, as discussed above, this threshold in β becomes much smaller, i.e., 0.4 in all the three classifiers.

Moreover, it is valuable to investigate which features are more likely to be selected by our approach. Table II lists the



(a) (b) (c)

Fig. 5. Comparison of the Prediction Performance among the Six Schemes by Varying β Values, and Using Different Classifiers. (a) NB. (b) C4.5. (c) IB1.

TABLE II
THE TOP 10 FEATURES SELECTED BY OUR TWO-STAGE APPROACH

Eclipse Datasets		NASA Datasets	
Feature Name	Average Frequency	Feature Name	Average Frequency
pre-release defects	1.00	Cyclomatic_density	0.70
NORM_ImportDeclaration	0.78	Maintenance_severity	0.56
Javadoc	0.77	NORM_Cylomatic_complexity	0.54
PAR_max	0.72	Design_complexity	0.50
StringLiteral	0.71	Essential_complexity	0.47
NORM_Javadoc	0.64	LOC_code_and_comment	0.46
NORM_FieldDeclaration	0.46	Decision_density	0.46
ConditionalExpression	0.43	Call_pairs	0.45
MLOC_sum	0.40	Essential_density	0.44
ThisExpression	0.39	LOC_Blank	0.44

TABLE III
THE SETTING OF THE THRESHOLD β OF NTC FOR THE DATASETS

	Eclipse 2.0	Eclipse 2.1	Eclipse 3.0	cm1	kc1	kc3	kc4	mc2	mw1	pc1	pc3	pc4	pc5
IG+SU	0.3	0.4	0.5	0.6	0.7	0.8	0.8	0.7	0.7	0.4	0.5	0.8	0.8
CS+SU	0.3	0.4	0.5	0.5	0.7	0.6	0.7	0.5	0.6	0.4	0.5	0.7	0.8

top 10 features selected by both IG+SU, and CS+SU from the Eclipse datasets, and the NASA datasets respectively (each has distinct feature types and numbers), over the settings of β. The average frequency of selection (i.e., the proportion of appearance in the selected feature sets) by the two schemes is used to order the features. For the Eclipse datasets, the defect based features are mostly selected by the two schemes. The other useful features are relevant to the counting of the syntax units in codes and Javadocs, such as parameter (PAR), Field, and Import. For the NASA datasets, the selected feature sets show more diversity. The reason is that the features used in the NASA datasets are different from each other. Among the top selected features, the McCabe based features are dominant. The rest are relevant to LOC and the counting of syntax units in codes. All these features are relatively easy to collect. Additionally, it may be useful to weight the features according to the collection costs, which can be used to improve our approach in our future work.

Based on the above analysis, in the following experiments, we choose IG + SU and CS + SU as representatives of our approach, and set the threshold β for each dataset to obtain the comparable sizes of the final feature sets to the baseline methods. The purpose is to demonstrate the potential of our approach in improving the prediction performance of the classifiers trained after, instead of to find the best possible scheme, which may be specific to the classifier and dataset used. Table III lists all the β values set for each dataset (β is slightly different between the two schemes because IG and CS will lead to dissimilar feature sets).

B. RQ2: Effectiveness of the Two-Stage Approach

To investigate the effectiveness of the two-stage data preprocessing approach for software fault prediction, we compare the approach to other commonly used feature selection methods, with or without instance reduction. To make a fair comparison, we need to control both the proportion of selected features for

TABLE IV
PROPORTION (IN PERCENT) OF SELECTED FEATURES BY DIFFERENT FEATURE SELECTION METHODS

Dataset	Percentage of Selected Features				
	IG+SU	CS+SU	IG	CFS	Consist
Eclipse 2.0	**1.29**	2.58	5.16	22.00	10.19
Eclipse 2.1	**4.52**	**4.52**	5.16	9.61	16.26
Eclipse 3.0	8.39	8.39	**5.16**	17.94	17.10
cm1	**13.51**	**13.51**	16.22	20.54	15.68
kc1	4.65	**3.49**	8.14	9.65	11.05
kc3	**12.82**	20.51	15.38	14.62	32.82
kc4	21.43	**14.29**	28.57	30.71	31.43
mc2	**12.82**	**12.82**	15.38	20.77	22.05
mw1	**13.51**	16.22	16.22	18.11	20.00
pc1	13.51	**10.81**	16.22	15.68	42.16
pc3	**10.81**	13.51	16.22	21.62	46.22
pc4	13.51	13.51	16.22	**6.22**	53.78
pc5	15.79	10.53	15.38	**8.46**	34.62
Average	11.27	**11.13**	13.80	16.61	27.18

the first stage, and the balance of instances for the second stage. Table IV lists the proportion of selected features by our approach, and the other baseline methods over the 13 datasets. From Table IV, on average the proportion of features selected by IG + SU is 11.27%, and by CS + SU is 11.13%; both are smaller than the proportions selected by the other methods, including IG, CFS, and Consist. The only exception is in the dataset Eclipse 3.0, when compared to IG, both IG + SU and CS + SU select slightly more features.

To balance the instances between the NFP and FP classes, random sampling is applied. During the experiments, the target ratio between NFP and FP is set between 65% and 35%, as recommended by Khoshgoftaar *et al.* [41]. Table V lists the proportion of instances (to all instances) decreased, or increased in NFP and FP classes by RUS, or ROS respectively. From Table V, RUS removes 60.5% instances on average, which suggest a serious imbalance exists in most datasets (except kc1 and mc2, which have relatively few instances, as shown in Table I). On the other hand, ROS makes replication of 21.5% of the original instances on average. The effects of ROS will be further studied in Section VII-D.

Tables VII through IX present in more detail the expected prediction performance (AUC) of the three classifiers (NB, C4.5, and IB1) built after the six feature selection methods with or without instance reduction. Our two-stage approach is represented by IG + SU and CS + SU with RUS. Additionally, both the random sampling methods are applied after the baseline methods including ALL, which means all features are used. Each AUC value is the mean result by 10×10-fold cross validation, and averaged over the 13 datasets using one of the three classifiers.

In the following, we mainly investigate the effects of our approach (denoted as IG + SU + RUS and CS + SU + RUS), and the three baseline methods (IG, CFS, and Consist), either with or without random under-sampling (postfixed by RUS). Table VI summarizes the comparison results using AUC values over all three classifiers.

In Table VI, the first row presents the methods in descending order according to the average value of AUC, which is listed in the second row. The third row gives the Win, Draw, Loss, which makes detailed comparison between the best method and the other ones. The last row gives the standard deviation of AUC for each method.

From Table VI, the representative IG + SU + RUS of our approach is the top best method according to the average AUC. By adding RUS, the performances of all the three baseline methods are improved. This result demonstrates the merit of instance reduction for software fault prediction. To validate the significance of the ranking, we apply the Friedman test [68], which is a non-parametric statistical test to check whether the ranking of multiple columns (i.e., methods) is consistent across the rows (datasets) [4], [69]. The resulted p-value is 0.019 (smaller than 0.05), which confirms that the ranking is significant. The Win, Draw, Loss records also support the dominance of the best method IG + SU + RUS, which always wins no less than two third of all the 39 cases (3 classifiers $\times$ 13 datasets). Our approach performs well on datasets which contain a great volume of features and instances, e.g., the three Eclipse datasets, and pc5, while less effective on datasets which contain few features or instances, e.g., on kc4, and mw1. By checking the standard deviation of AUC, we can find that all the methods perform rather steadily over the 13 datasets, instead of the difference among the three classifiers.

Considering the classifiers individually, For Naive Bayes (refer to Table VII), instance reduction is less effective. Here, RUS makes little difference in the prediction performance, no matter which feature selection method is used. As a matter of fact, the average AUC is slightly decreased by adding RUS, which is true even when IG+SU and CS+SU are used for feature selection. The possible reason is that Naive Bayes is not sensitive to the imbalance between the classes [70], while the final performance is affected by RUS as many NFP instances are removed. However, considering feature selection, IG + SU, and CS + SU are still the top two best methods, with or without RUS. The possible reason is that NB depends on the conditional probability of the classes against individual features; and by removing redundant features, the conditional probability will be more distinguishable.

For the C4.5 decision tree (refer to Table VIII), the results are nearly consistent to Table VI. In C4.5, the decision tree is built on the information gain of individual features, while the calculation of SU is based on the same facility. Hence, by eliminating redundant features according to SU, the tree nodes built after can make more accurate predictions. Considering instance reduction, C4.5 is suffered by the imbalanced dataset. By adding RUS, the prediction performance is improved significantly for all the feature selection methods.

For the nearest neighbors IB1 (refer to Table IX), the results are slightly different from Table VI. By adding RUS, CFS performs the second best, and exceeds CS + SU. Taking it further, according to Table IV, in most datasets, CFS selects the greatest number of features among the feature selection methods (except ALL). Because IB1 makes predictions based on the nearest

TABLE V
PROPORTION (IN PERCENT) OF REMOVED OR REPLICATED INSTANCES FROM EACH DATASET

	Eclipse2.0	Eclipse2.1	Eclipse3.0	cm1	kc1	kc3	kc4	mc2	mw1	pc1	pc3	pc4	pc5
RUS	-62.8	-72.2	-62.0	-75.8	-10.3	-76.2	-11.2	-18.0	-80.7	-82.5	-73.8	-68.7	-92.3
ROS	+18.4	+23.5	+18.0	+25.0	+5.0	+25.1	+23.9	+4.2	+27.3	+28.8	+24.2	+21.5	+34.3

TABLE VI
COMPARISON AMONG THE DATA PRE-PROCESSING METHODS USING AVERAGE AUC OVER THE THREE CLASSIFIERS

	IG+SU +RUS	CFS +RUS	CS+SU +RUS	IG +RUS	Consist +RUS	IG	CFS	Consist
AUC	0.748	0.741	0.739	0.733	0.717	0.713	0.707	0.701
Win,Draw,Loss[1]	-	26/0/13	27/1/11	28/0/11	30/0/9	31/0/8	29/1/9	32/2/5
StDev.	0.089	0.086	0.091	0.096	0.079	0.105	0.094	0.096

[1] Win,Draw,Loss presents the number of datasets on which the left most method (column 1) performs better, equal, or worse than each of the other method respectively. All Win,Draw,Loss records in the following tables have the same meaning.

TABLE VII
MEAN AUC OF NB AFTER USING DIFFERENT FEATURE SELECTION OR INSTANCE REDUCTION METHODS OR BOTH

Dataset	Random Under-Sampling						Without Random Sampling						Random Over-Sampling					
	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL
Eclipse2.0	0.830	0.825	0.794	0.807	0.705	0.792	0.823	0.813	0.788	0.810	0.803	0.795	0.749	0.798	0.794	0.808	0.701	0.796
Eclipse2.1	0.766	0.761	0.778	0.749	0.730	0.743	0.766	0.764	0.758	0.745	0.762	0.746	0.768	0.765	0.778	0.740	0.721	0.744
Eclipse3.0	0.780	0.781	0.791	0.758	0.758	0.762	0.779	0.779	0.762	0.756	0.769	0.762	0.780	0.781	0.791	0.756	0.757	0.761
cm1	0.767	0.730	0.753	0.753	0.684	0.744	0.776	0.770	0.772	0.763	0.598	0.756	0.768	0.749	0.765	0.764	0.704	0.740
kc1	0.810	0.787	0.807	0.804	0.753	0.806	0.809	0.805	0.808	0.820	0.813	0.796	0.795	0.787	0.807	0.804	0.753	0.806
kc3	0.786	0.805	0.753	0.774	0.753	0.804	0.809	0.807	0.800	0.797	0.795	0.808	0.792	0.817	0.793	0.792	0.781	0.800
kc4	0.787	0.764	0.725	0.742	0.742	0.753	0.784	0.790	0.764	0.751	0.770	0.757	0.787	0.764	0.725	0.742	0.742	0.753
mc2	0.685	0.662	0.626	0.650	0.641	0.719	0.683	0.658	0.661	0.657	0.679	0.719	0.685	0.665	0.625	0.649	0.647	0.724
mw1	0.778	0.771	0.783	0.804	0.714	0.740	0.795	0.790	0.785	0.780	0.789	0.744	0.780	0.777	0.783	0.785	0.765	0.746
pc1	0.767	0.712	0.691	0.785	0.744	0.736	0.773	0.772	0.768	0.790	0.743	0.757	0.776	0.744	0.735	0.787	0.761	0.736
pc3	0.809	0.803	0.799	0.796	0.745	0.773	0.800	0.800	0.791	0.788	0.743	0.773	0.807	0.802	0.796	0.791	0.748	0.769
pc4	0.841	0.839	0.830	0.827	0.770	0.835	0.839	0.838	0.835	0.816	0.839	0.842	0.840	0.835	0.830	0.828	0.783	0.844
pc5	0.951	0.932	0.943	0.908	0.852	0.941	0.954	0.932	0.943	0.883	0.935	0.939	0.954	0.929	0.945	0.867	0.773	0.937
Average	0.797	0.782	0.775	0.781	0.738	0.781	0.799	0.794	0.787	0.781	0.772	0.784	0.791	0.786	0.782	0.778	0.741	0.781

neighbors of the target instances, it could (although not always) be the case that the more the features are left, the more wholesome the nearest neighbors are determined. On the other hand, by removing excessive instances in NFP, RUS can evidently improve the final performance. This outcome may be due to the reason that it is beneficial for IB1 if the instance sets of different classes have comparable densities in distance, and RUS is helpful in decreasing the density of the NFP class, making it comparable to the FP class.

In summary, for RQ2, we can conclude that the two-stage data preprocessing approach can further improve the prediction performance of the classification models trained after, compared with other commonly used feature selection methods. For the feature selection stage, the proposed method NTC is a good option. In addition, the data preprocessing can greatly reduce both the number of features (up to 89%) and the number of instances of the original dataset (up to 60%), which will simplify the training process of the classifiers.

C. *RQ3: The Effectiveness of NTC*

In this subsection, we investigate whether our proposed algorithm NTC, which is the core part of the feature selection stage, has advantage over other feature selection methods, including IG, CFS, Consist, and ALL. Table X summarizes the comparison results using AUC values over all the three classifiers.

Table X is organized in the same fashion as Table VI. From Table X, the two methods which involve NTC (i.e., $IG + SU$ and $CS + SU$) are the top two best methods, which demonstrate the merit of NTC for software fault prediction. ALL is not a good choice; and by Win, Draw, Loss records, it cannot win Consist, although its statistical mean AUC is slightly higher. This result demonstrates that feature selection is valuable. To validate the significance of the ranking, we apply the Friedman test. The resulted p-value is 0.0011 (much smaller than 0.05), which states that the ranking is significant in general. The Win, Draw, Loss records support the dominance of the best method

TABLE VIII
MEAN AUC OF C4.5 AFTER USING DIFFERENT FEATURE SELECTION OR INSTANCE REDUCTION METHODS OR BOTH

Dataset	Random Under-Sampling						Without Random Sampling						Random Over-Sampling					
	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL
Eclipse2.0	0.766	0.784	0.734	0.730	0.752	0.714	0.761	0.757	0.745	0.707	0.723	0.671	0.742	0.703	0.707	0.669	0.672	0.669
Eclipse2.1	0.745	0.747	0.751	0.739	0.692	0.659	0.734	0.734	0.700	0.718	0.668	0.590	0.578	0.590	0.642	0.611	0.607	0.613
Eclipse3.0	0.750	0.751	0.740	0.722	0.706	0.670	0.740	0.741	0.732	0.704	0.705	0.637	0.731	0.734	0.717	0.644	0.638	0.642
cm1	0.674	0.669	0.651	0.656	0.583	0.632	0.658	0.657	0.540	0.548	0.525	0.528	0.589	0.627	0.626	0.556	0.527	0.602
kc1	0.715	0.705	0.726	0.714	0.741	0.716	0.730	0.676	0.727	0.715	0.701	0.702	0.715	0.705	0.726	0.714	0.741	0.716
kc3	0.665	0.710	0.666	0.706	0.676	0.702	0.620	0.551	0.599	0.606	0.611	0.578	0.498	0.564	0.544	0.570	0.623	0.631
kc4	0.748	0.727	0.719	0.751	0.764	0.763	0.750	0.745	0.733	0.754	0.765	0.775	0.748	0.727	0.719	0.751	0.764	0.763
mc2	0.589	0.568	0.594	0.582	0.582	0.627	0.587	0.575	0.594	0.582	0.589	0.636	0.583	0.562	0.596	0.591	0.566	0.631
mw1	0.552	0.530	0.545	0.523	0.570	0.560	0.620	0.506	0.614	0.570	0.602	0.569	0.468	0.480	0.474	0.484	0.483	0.461
pc1	0.784	0.763	0.742	0.783	0.748	0.752	0.701	0.691	0.725	0.677	0.672	0.650	0.693	0.663	0.666	0.677	0.668	0.651
pc3	0.741	0.739	0.740	0.738	0.677	0.694	0.610	0.600	0.536	0.590	0.615	0.635	0.598	0.603	0.629	0.600	0.590	0.616
pc4	0.859	0.850	0.862	0.870	0.808	0.802	0.873	0.866	0.854	0.866	0.771	0.760	0.761	0.750	0.747	0.823	0.717	0.708
pc5	0.936	0.924	0.929	0.921	0.933	0.913	0.903	0.902	0.890	0.853	0.821	0.774	0.773	0.756	0.706	0.756	0.669	0.638
Average	0.733	0.728	0.723	0.726	0.710	0.708	0.714	0.692	0.691	0.684	0.674	0.654	0.652	0.651	0.654	0.650	0.636	0.642

TABLE IX
MEAN AUC OF IB1 AFTER USING DIFFERENT FEATURE SELECTION OR INSTANCE REDUCTION METHODS OR BOTH

Dataset	Random Under-Sampling						Without Random Sampling						Random Over-Sampling					
	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL	IG+SU	CS+SU	IG	CFS	Consist	ALL
Eclipse2.0	0.700	0.706	0.679	0.724	0.694	0.731	0.674	0.672	0.652	0.665	0.625	0.694	0.674	0.668	0.652	0.665	0.623	0.694
Eclipse2.1	0.661	0.661	0.656	0.665	0.704	0.670	0.636	0.628	0.632	0.635	0.639	0.638	0.636	0.629	0.632	0.636	0.639	0.636
Eclipse3.0	0.675	0.674	0.669	0.687	0.726	0.684	0.664	0.662	0.660	0.663	0.660	0.654	0.668	0.662	0.665	0.663	0.657	0.654
cm1	0.686	0.619	0.601	0.618	0.575	0.652	0.652	0.642	0.603	0.629	0.557	0.607	0.652	0.642	0.603	0.629	0.557	0.607
kc1	0.707	0.682	0.691	0.727	0.701	0.722	0.680	0.670	0.671	0.671	0.670	0.669	0.681	0.675	0.673	0.671	0.670	0.670
kc3	0.683	0.692	0.673	0.721	0.717	0.721	0.550	0.530	0.539	0.528	0.600	0.576	0.550	0.530	0.539	0.528	0.610	0.577
kc4	0.743	0.749	0.788	0.722	0.703	0.688	0.747	0.734	0.756	0.752	0.705	0.690	0.747	0.734	0.753	0.752	0.705	0.690
mc2	0.614	0.594	0.550	0.613	0.578	0.688	0.608	0.577	0.533	0.610	0.564	0.673	0.608	0.577	0.533	0.610	0.564	0.673
mw1	0.644	0.615	0.612	0.634	0.639	0.639	0.578	0.562	0.560	0.568	0.561	0.564	0.575	0.562	0.560	0.564	0.561	0.564
pc1	0.754	0.770	0.751	0.765	0.705	0.751	0.727	0.710	0.706	0.724	0.665	0.701	0.727	0.710	0.704	0.725	0.665	0.705
pc3	0.690	0.703	0.692	0.701	0.717	0.714	0.638	0.631	0.630	0.637	0.631	0.638	0.637	0.630	0.630	0.637	0.631	0.638
pc4	0.808	0.805	0.815	0.837	0.836	0.769	0.798	0.775	0.790	0.809	0.808	0.684	0.798	0.774	0.789	0.809	0.805	0.682
pc5	0.944	0.924	0.937	0.882	0.853	0.943	0.916	0.816	0.866	0.631	0.838	0.930	0.915	0.816	0.865	0.633	0.836	0.928
Average	0.716	0.707	0.701	0.715	0.704	0.721	0.682	0.662	0.661	0.656	0.656	0.671	0.682	0.662	0.661	0.656	0.656	0.671

TABLE X
COMPARISON AMONG THE FEATURE SELECTION METHODS BY AVERAGE AUC OVER THE THREE CLASSIFIERS

	IG+SU +None	CS+SU +None	IG	CFS	ALL	Consist
AUC	0.732	0.716	0.713	0.707	0.703	0.701
Win/Draw/Loss	-	34/3/2	36/0/3	33/0/6	29/0/10	31/0/8
StDev.	0.098	0.104	0.105	0.094	0.095	0.096

IG + SU, which always wins more than two thirds of all the 39 cases (3 classifiers × 13 datasets). According to the standard deviation of the AUC values, the variances over the datasets are small regardless of the different classifiers used.

Considering the three classifiers, both NB (refer to Table VII) and C4.5 (refer to Table VIII) are generally consistent to the ordering in Table X, where IG + SU, and CS + SU are the first, and second best feature selection methods. These results confirm the analysis results in the above section for both NB and C4.5. However, for IB1, ALL is the second best now, and outperforms CS+SU. This outcome may reinforce the comment that more features could be better in determining the wholesome neighbors.

In summary, for RQ3, we can conclude that feature selection is useful in improving the prediction performance of the classification models built after. In addition, a redundancy control facility such as NTC (which leads to the best performance among all the methods) is valuable in constructing an appropriate feature set for training the classifiers.

D. *RQ4: The Effectiveness of Instance Reduction Facilities*

In RQ4, we mainly investigate the possible impact of the instance reduction facilities in the two-stage approach. First, we

TABLE XI
COMPARISON AMONG THE METHODS WITH DIFFERENT INSTANCE SAMPLING TECHNIQUES BY AVERAGE AUC OVER THE THREE CLASSIFIERS

Instance ratio (NFP/FP)		IG+SU +RUS	CS+SU +RUS	ALL +RUS	IG+SU +None	CS+SU +None	IG+SU +ROS	CS+SU +ROS	IG +ROS	ALL +ROS	CFS +ROS	Consist +ROS
65%/35%	AUC	0.748	0.739	0.737	0.732	0.716	0.708	0.700	0.699	0.698	0.694	0.678
	Win/Draw/Loss	-	27/1/11	25/0/14	28/1/10	33/0/6	27/5/7	37/0/2	32/0/7	32/0/7	32/0/7	36/1/2
	StDev.	0.089	0.091	0.080	0.098	0.104	0.106	0.097	0.101	0.094	0.095	0.084
50%/50%	AUC	0.747	0.743	0.741	0.732	0.716	0.706	0.707	0.705	0.705	0.705	0.699
	Win/Draw/Loss	-	23/3/13	27/0/12	24/1/14	30/0/9	31/0/8	28/0/11	31/0/8	32/0/7	29/0/10	32/0/7
	StDev.	0.087	0.083	0.076	0.098	0.104	0.096	0.087	0.091	0.090	0.087	0.095

validate if, by combining NTC and RUS (under-sampling) together, the performances of the classifiers built after are better than by using either of them separately. Second, we validate whether RUS performs better than other instance sampling techniques, such as ROS for software fault prediction. Table XI summarizes the comparison results using AUC values over the three classifiers.

From Table XI, the two representatives of our approach (IG + SU + RUS, and CS + SU + RUS) are the top two best methods. The third best method is ALL + RUS, better than NTC alone (represented by IG + SU + None and CS + SU + None), which suggests that RUS can be valuable to use before the training of a classifier for software fault prediction. On the other hand, by adding ROS, all the performances of the six methods become worse with few exceptions (refer to Tables VII through IX), which suggest ROS is unfit for software datasets. To validate the significance of the ranking, the Friedman test is applied. The resulted p-value is 5.13E-07, far less than 0.05, which states that the ranking is significant. The Win, Draw, Loss records also support the dominance of the best method IG + SU + RUS, and the standard deviations demonstrate the steadiness of the performances over the three different classifiers.

In addition, Table XI presents the results when the ratio of the #NFP class to the #FP class is set to 50%/50%. The performance ranking of the methods is nearly unchanged compared to the 65%/35% instance ratio. By applying the Friedman test, the resulted p-value is 9.06E-07, which ensures a significant ranking. Some methods are slightly improved by using the 50%/50% ratio, especially those using ROS. Two of the top 3 methods which use RUS are also improved. These results suggest that a balanced dataset is always beneficial for training the classification models.

Considering the three classifiers, the classifier NB (refer to Table VII) again performs against the instance reduction stage. By adding either RUS or ROS, the performances of the six methods become worse. This result implies that it is necessary to find a good method at the second instance reduction stage to suit classifiers such as Naive Bayes. For C4.5 (refer to Table VIII), the results are consistent to Table XI. As stated under RQ2, C4.5 suffers from imbalanced datasets. However, although the datasets are made balanced by adding ROS, the prediction performance is decreased. The possible reason is that, by adding replicated instances, the calculated information gain of individual features could be misleading, and make the decision tree built less useful. For IB1 (refer to Table IX), the best method is now ALL+RUS, supported by both the average AUC and the Win, Draw, Loss records. This outcome reinforces the above analysis for IB1, that more features (remaining after feature selection) would be more helpful in determining suitable neighbors to make better predictions. In addition, ROS, which merely adds replicates, makes no difference (other than random perturbance) in the performance of IB1.

In summary, for RQ4, we can conclude that our proposed algorithm NTC can be further improved by the instance reduction facility (i.e., RUS) for data preprocessing, at least applied to the three classification models used for software fault prediction. According to the experimental results, data imbalance may be a general issue in software repositories, and the design of suitable instance sampling techniques deserves further study.

E. *RQ5: The Impact of the Dataset Properties*

Based on the above analysis, we can find that the final prediction performance after our two-stage approach varies on different datasets. Hence, it is valuable to investigate how the properties of a dataset take effect on data preprocessing. In this subsection, we mainly focus on two such properties: the level of instance sufficiency, i.e., the proportion of the number of instances to that of features; and the temporal issue, i.e., whether later instances are more useful than earlier ones. We investigate if the optimal value of β may be determined by the level of instance sufficiency, and if different sections of instances may lead to a different prediction performance.

1) The Level of Instance Sufficiency: Here we define the level of instance sufficiency of a dataset as the ratio of the number of the instances to that of the features. A great instance to feature ratio means the dataset has sufficient instances to make sensible classification by the feature values. The datasets used in our experiments do have great differences in the instance sufficiency levels, which are listed in Table XII. For example, the dataset kc1 has 86 features, while it contains only 145 instances, which leads to the smallest sufficiency level of 1.69. On the other hand, pc5 contains 17186 instances with only 38 features, which leads to the greatest sufficiency level 452.26.

To investigate the effects of the instance sufficiency level of a dataset on the prediction performance, we list the optimal settings of β of the two-stage data preprocessing approach over the three classifiers on each dataset in Table XII. Because the two-stage approach will decrease both the number of features and instances, we also list the modified sufficiency levels correspondingly. Refer to Fig. 5; the optimal β differs little over the three classifiers. Because the optimal β will hold for a relatively wide range before the performance drops. The values shown in

TABLE XII
COMPARISON OF OPTIMAL β SETTINGS AMONG THE DATASETS WITH VARIANT INSTANCE SUFFICIENCY LEVELS

Datasets	Ratio Before[1]	Optimal β		Ratio After[2]	
		IG+SU +RUS	CS+SU +RUS	IG+SU +RUS	CS+SU +RUS
pc5	452.26	0.6	0.6	663.00	663.00
Eclipse 3.0	68.34	0.4	0.4	671.83	671.83
Eclipse 2.1	50.89	0.4	0.4	313.43	313.43
Eclipse 2.0	43.41	0.4	0.4	417.67	417.67
pc3	42.24	0.6	0.5	108.33	136.67
pc4	39.41	0.7	0.6	91.20	106.33
pc1	29.92	0.7	0.6	54.25	62.25
cm1	13.65	0.7	0.7	38.43	38.43
kc3	11.74	0.7	0.6	36.33	41.80
mw1	10.89	0.8	0.7	26.14	36.00
kc4	8.93	0.8	0.7	41.00	55.50
mc2	4.13	0.9	0.8	18.86	22.00
kc1	1.69	0.8	0.8	18.57	18.57

1. Ratio Before presents the ratio of #instance to #feature before the data preprocessing.
2. Ratio After presents the ratio of #instance to #feature after the data preprocessing.

Table XII are those that firstly reach at least 98% of the best performance (leaving 2% to accommodate random perturbance) on each dataset.

Considering the optimal β, according to the third and fourth columns in Table XII, the two schemes IG + SU + RUS and CS+SU+RUS have nearly similar β settings to accomplish the best performance over all the datasets. On individual datasets, the optimal values of β are variant. A general trend can be found when taking into consideration the instance sufficiency level of each dataset (refer to the second column, where the sufficiency levels are listed in descending order). The optimal β tends to be small when the sufficiency level is great, and vice versa. For example, the Eclipse datasets have plenty of instances, and the optimal β can be around a small value (i.e., 0.4), while datasets mc2 and kc1 are short of instances, so the optimal β will require a great value (i.e., 0.8–0.9, which means a large proportion of the features are selected). This result partly interprets the phenomenon in Fig. 5, where the best performance will not drop until β reaches 0.8, because some datasets (at least 4 out of 13) do require a great enough β. The dataset pc5 apparently contradicts the negative correlation between the sufficiency level and the optimal β. However, pc5 contains a great number of 17,186 instances with a small number of 38 features. The instances are sufficient enough to support a great number of features for making effective classification.

To make clear the effects of the optimal β (i.e., effective data preprocessing), we can investigate the instance sufficiency level afterwards (shown in the last two columns in Table XII). Here, the sufficiency levels are consistently greater than the initial counterparts. Especially on Eclipse 3.0, the sufficiency level after is made similar to that on pc5, although a great gap exists between their initial values. This result suggests that a great instance sufficiency level will help enhance the performance of classification. On the other hand, for software datasets, when the number of instances is small, the features should not be presumptuously removed to make a simpler but less effective classifier for fault prediction. Software is complex, and every feature may be valuable, especially when the software samples are insufficient. Further study on this issue will be part of our future work.

TABLE XIII
COMPARISON AMONG DIFFERENT TRAINING SECTIONS USING AVERAGE AUC OVER THE THREE CLASSIFIERS

Section	Eclipse 2.0		Eclipse 2.1		Eclipse 3.0	
	IG+SU +RUS	CS+SU +RUS	IG+SU +RUS	CS+SU +RUS	IG+SU +RUS	CS+SU +RUS
Early 30%	**0.712**	**0.722**	0.653	0.643	0.749	**0.728**
Midterm 30%	0.709	0.653	0.672	0.648	0.710	0.669
Late 30%	0.702	0.714	**0.688**	**0.712**	**0.752**	0.716
Benchmark 90%	0.740	0.753	0.737	0.761	0.746	0.728

2) The Temporal Issues: The temporal issue, i.e., the time when an instance (software module) is sampled, may influence its usefulness in predicting the fault proneness of new modules. During the experiments, the Eclipse datasets contain traceable time information [50], and its instances can be assumed as being stored in time order. To evaluate the effects of temporal issues on the final prediction performance, we separate the instances in each of the three Eclipse datasets into three sections (for training): the Early section, containing the first 30% instances; the Midterm section, containing instances within 30 to 60%; and the Late section, containing instances within 60 to 90%. The last 10% instances are used for testing. Table XIII presents the average AUC values resulting from the three classifiers using each of the three sections as the training set, respectively. The parameter β is set at the optimal values (i.e., 0.4) suggested in Table XII. The last row in Table XIII gives the average AUC when 90% instances are used for training without considering the time order.

According to Table XIII, different sections of instances have variant influences on the final prediction performance. On average, both the Early section and the Late section are more useful than the Midterm section. It is hard to determine whether either the Early section or the Late section is more useful. Each has cases under which one evidently outperforms the other. By exploring the datasets, the Early section may contain software modules whose updated versions appear in the testing set, while the Late section is temporally closer to software modules under prediction. The above suggests that temporal issues deserve consideration during data preprocessing. Further study on this question will be part of our future work.

By combining all the three sections, the prediction performance usually improves greatly. An exception is on Eclipse 3.0, where both the Early and Late sections can lead to good prediction performance. The reason may be that Eclipse 3.0 has the greatest number of instances among the three datasets, and a single section contains enough instances that reach the desired sufficiency level.

F. Discussion

In this paper, we design a two-stage data preprocessing approach which includes the feature selection stage, and the instance reduction stage. The experimental results have proved the effectiveness of the approach when compared to some typical baseline feature selection methods, such as IG, CFS, and Consist, with or without instance sampling.

For feature selection, the NTC is effective in eliminating redundant features while keeping relevant ones. Among the similarity functions used for clustering features, symmetric uncertainty, which is non-linear, is proved to be the best, especially when combined with IG and CS. This conclusion conforms to the findings of Yu and Liu [48], which suggest non-linear correlations among software measures. The similarity threshold needs to be set according to the dataset, but the resulted performance is quite stable for a large threshold range (e.g., 0.4 to 1.0). The greater the threshold, the more features will be selected.

For instance reduction, the random under-sampling technique is proved useful, and can be applied to other feature selection methods with profitable results, while the over-sampling technique is not suitable, at least for software datasets. Random sampling is effective in the experiments, because we have found that similarities among instances in the datasets are usually high (e.g., the average instance similarity measured by SU is greater than 0.9 in almost all the datasets).

We also investigate how the properties of a dataset will affect the data preprocessing approach, because its performance differs on different datasets. First, the optimal β may be determined by the level of instance sufficiency, i.e., the ratio of number of the instances to that of the features. Based on the experiments, when the sufficiency level is great, β can be safely set to a small value, which means a small proportion of the features are enough. On the other hand, if the sufficiency level is small, β requires a large value, and the features need to be preserved. Overall, β can be set within [0.4, 0.6] to get a promising result.

Second, the temporal issue requires consideration during data preprocessing. Based on experiments, either temporally adjacent software samples or highly coupled coding units may determine the usefulness of the software instances. However, the above statement still requires clear evidence. Further study on the issue will be part of our future work.

Additionally, we plan to design a data preprocessing approach whose effectiveness should be unaffected by the classification models used. However, the experimental results do not support such an assumption. Different classifiers may call for different data preprocessing techniques. For example, Naive Bayes performs worse if any instance reduction technique is applied, while the k-nearest neighbors ($k = 1$) might perform poorly with any feature selection technique. The C4.5 decision tree may require a compliant similarity measure used to eliminate redundant features. Above all, the two-stage data preprocessing approach can provide a useful framework to accommodate suitable feature selection and instance sampling techniques for improving the performance of the classification models, and incorporating new effective techniques into the framework still remains an open research issue.

G. Threats to Validity

Here we discuss the potential threats to the validity of our study.

Threats to the external validity include whether the observed experimental results can be generalized to other situations. To guarantee the representativeness of our experiments, we choose the commonly used Eclipse and NASA datasets, which include 13 carefully selected subjects. In addition, we choose three classification models, which are widely used in software fault prediction, to guarantee soundness of the results.

Threats to the internal validity include the faults that may reside in the implementations of the data preprocessing approach and classification models. To avoid this type of threat, all the implementations are cross-checked by our research group, and the classification models and other benchmark methods are provided by the commonly used Weka package. Moreover, the datasets are carefully examined, where non-numeric features and the features which have only one distinct value are eliminated.

Threats to the construct validity include whether the performance measures used in the experiments are suitable to reflect the real-world performance of the two-stage approach. We choose the AUC measure to evaluate the prediction performance. Compared with other performance measures, such as precision, recall, or F-measure, AUC has a lower variance, and is more reliable to indicate the predictive potential of the methods. Moreover, to alleviate the random effects, we perform 10×10-fold cross-validation, and use the mean AUC measure to represent the prediction performance. Finally we do the Friedman test to validate the significance of the performance ranking of the methods.

VI. Conclusion

In this paper, we provide a two-stage data preprocessing approach, which incorporates both feature selection and instance reduction, to improve the quality of software datasets used by classification models for software fault prediction. In the feature selection stage, we propose a novel algorithm which involves both relevance analysis and redundancy control. In the instance reduction stage, we apply random under-sampling to keep the balance between the faulty and non-faulty instances. We systematically design experiments based on the Eclipse and NASA datasets, and compare our approach to other commonly used data preprocessing methods. The results demonstrate the potential of our approach in enhancing the prediction performance of the classifiers built thereafter.

In our future work, we plan to extend the two-stage approach in several ways. First, we plan to investigate the inter-relations between the feature selection and the instance reduction methods, and investigate key factors such as instance sufficiency level or inter-relation among the instances for optimal parameter setting. Second, we plan to apply our approach to other real-world software datasets to prove its effectiveness. Third, the temporal issues deserve more study. Finally, we plan to use other classification models to validate the generality of the approach, and design data preprocessing methods suitable for specific classification models.

References

[1] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, 2007.

[2] S. Kim, E. J. W. Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?," *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, 2008.

[3] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2008, pp. 181–190.

[4] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, 2008.

[5] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 852–864, 2010.

[6] Q. Song, Z. Jia, M. J. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, 2011.

[7] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 6, pp. 1146–1150, 2012.

[8] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proc. IEEE Int. Conf. Software Eng.*, 2013, pp. 382–391.

[9] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using Bayesian network classifiers," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 237–257, 2013.

[10] A. Monden, T. Hayashi, S. Shinoda, K. Shirai, J. Yoshida, M. Barker, and K. Ichi Matsumoto, "Assessing the cost effectiveness of fault prediction in acceptance testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1345–1357, 2013.

[11] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proc. Int. Conf. Automated Software Eng.*, 2013, pp. 279–289.

[12] F. Rahman and P. T. Devanbu, "How, and why, process metrics are better," in *Proc. IEEE Int. Conf. Software Eng.*, 2013, pp. 432–441.

[13] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. W. , Jr., "Does bug prediction support human developers? Findings from a Google case study," in *Proc. IEEE Int. Conf. Software Eng.*, 2013, pp. 372–381.

[14] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proc. Working Conf. Mining Software Repositories*, 2014, pp. 182–191.

[15] F. Rahman, S. Khatri, E. T. Barr, and P. T. Devanbu, "Comparing static bug finders and statistical prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2014, pp. 424–434.

[16] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, "A two-stage data preprocessing approach for software fault prediction," in *Proc. Int. Conf. Software Security and Reliability*, 2014, pp. 20–29.

[17] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2014, pp. 414–423.

[18] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. T. Devanbu, "Fair and balanced? Bias in bug-fix datasets," in *Proc. Joint Meeting on Foundations of Software Eng.*, 2009, pp. 121–130, ACM.

[19] F. Rahman, D. Posnett, I. Herraiz, and P. T. Devanbu, "Sample size vs. bias in defect prediction," in *Proc. ACM Joint Meeting on Foundations of Software Eng.*, 2013, pp. 147–157.

[20] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proc. ACM Joint Meeting on Foundations of Software Eng.*, 2011, pp. 15–25.

[21] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2013, pp. 392–401.

[22] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2011, pp. 481–490.

[23] M. J. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the Nasa software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013.

[24] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *Proc. Int. Conf. Mach. Learn. Applications*, 2010, pp. 135–140.

[25] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Softw.-Practice Exper.*, vol. 41, no. 5, pp. 579–606, 2011.

[26] S. Shivaji, E. J. W. , Jr., R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 552–569, 2013.

[27] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.

[28] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, 2013.

[29] H. Liu and H. Motoda, *Instance Selection and Construction for Data Mining*. New York, NY, USA: Springer-Science and Business Media, 2001.

[30] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Trans. Syst., Man, Cybern., Part A: Syst. Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[31] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, "An empirical study of feature ranking techniques for software quality prediction," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 22, no. 2, pp. 161–183, 2012.

[32] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, 2012.

[33] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proc. Int. Conf. Information Reuse and Integration*, 2007, pp. 667–672.

[34] H. Liu and H. Motoda, "On issues of instance selection," *Data Mining Knowl. Discov.*, vol. 6, no. 2, pp. 115–130, 2002.

[35] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE Trans. Rel.*, vol. 61, no. 2, pp. 516–525, 2012.

[36] T. M. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Inf. Syst. Front.*, vol. 16, 2014.

[37] H. Liu, H. Motoda, and L. Yu, "A selective sampling approach to active feature selection," *Artif. Intell.*, vol. 159, no. 1, pp. 49–74, 2004.

[38] J. Chen, S. Liu, X. Chen, Q. Gu, and D. Chen, "Empirical studies on feature selection for software fault prediction," in *Proc. Asia-Pacific Symp. Internetware*, 2013, pp. 163–166.

[39] Q. Song, J. Ni, and G. Wang, "A fast clustering-based feature subset selection algorithm for high-dimensional data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 1–14, 2013.

[40] D. J. Dittman, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "Comparison of data sampling approaches for imbalanced bioinformatics data," in *Proc. Int. Florida Artif. Intell. Res. Soc. Conf.*, 2014.

[41] T. M. Khoshgoftaar, C. Seiffert, J. V. Hulse, A. Napolitano, and A. Folleco, "Learning with limited minority class data," in *Proc. Int. Conf. Mach. Learn. Applications*, 2007, pp. 348–353.

[42] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, 2011.

[43] J. J. Peterson, "Regression analysis of count data," *Technometrics*, vol. 41, no. 4, pp. 371–371, 1999.

[44] I. Kononenko, "Estimating attributes: Analysis and extensions of relief," in *Proc. Eur. Conf. Mach. Learn.*, 1994, pp. 171–182.

[45] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. New York, NY, USA: Cambridge Univ. Press, 1992.

[46] M. A. Hall, "Correlation-based feature selection for discrete and numeric class machine learning," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 359–366.

[47] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proc. Int. Conf. Mach. Learn.*, 2003, vol. 3, pp. 856–863.

[48] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *J. Mach. Learn. Res.*, vol. 5, pp. 1205–1224, 2004.

[49] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA, USA: Morgan Kaufmann, 2005.

[50] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proc. Int. Workshop on Predictor Models in Software Eng.*, 2007, pp. 9–9.

[51] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. IEEE Int. Conf. Software Eng.*, 2008, pp. 181–190.

[52] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. ACM Joint Meeting on Foundations of Software Eng.*, 2009, pp. 91–100.

[53] S. Krishnan, C. Strasburg, R. R. Lutz, K. Goseva-Popstojanova, and K. S. Dorman, "Predicting failure-proneness in an evolving software product line," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1479–1495, 2013.

[54] H. Zhang and S. Cheung, "A cost-effectiveness criterion for applying software defect prediction models," in *Proc. ACM Joint Meeting on Foundations of Software Eng.*, 2013, pp. 643–646.

[55] A. G. Koru, D. Zhang, K. E. Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Trans. Softw. Eng.*, vol. 35, no. 2, pp. 293–304, 2009.

[56] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 852–864, 2010.

[57] G. Czibula, Z. Marian, and I. G. Czibula, "Software defect prediction using relational association rule mining," *Inf. Sci.*, vol. 264, pp. 260–278, 2014.

[58] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, 2014.

[59] PROMISE Data Repository [Online]. Available: https://code.google.com/p/promisedata/

[60] MDP Data Repository [Online]. Available: http://nasa-softwaredefect-datasets.wikispaces.com/

[61] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Think locally, act globally: Improving defect and effort prediction models," in *Proc. Working Conf. Mining Software Repositories*, 2012, pp. 60–69.

[62] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "FECAR: A feature selection framework for software defect prediction," in *Proc. Annu. Computer Software and Applications Conf.*, 2014, pp. 426–435.

[63] M. Dash, H. Liu, and H. Motoda, "Consistency based feature selection," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*. New York, NY, USA: Springer, 2000, pp. 98–109.

[64] D. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering," in *Proc. Int. Conf. Mach. Learn.*, 1992, pp. 163–168.

[65] J. R. Quinlan*, C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[66] T. Fawcett, "An introduction to roc analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[67] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in *Proc. Int. Symp. Software Reliability Eng.*, 2009, pp. 99–108.

[68] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Ann. Math. Statist.*, vol. 11, no. 1, pp. 86–92, 1940.

[69] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[70] M. A. Maloof, "Learning when data sets are imbalanced and when costs are unequal and unknown," in *Working Notes of the ICML Workshop on Learning From Imbalanced Data Sets*, 2003, vol. 2.

Wangshu Liu received the B.Sc. and M.Sc. degrees in computer science from Nanjing University of Science and Technology, China, in 2010, and 2013, respectively. He is now a Ph.D. candidate at the Department of Computer Science and Technology, Nanjing University, China. His research interests include software fault prediction, and machine learning.

Shulong Liu received the B.Sc. degree in computer science from Qingdao Technological University, China, in 2012, and the M.Sc. degree in computer science from Nanjing University, China, in 2015. His research interests include software fault prediction, and machine learning.

Qing Gu received his Ph.D. degree in computer science from Nanjing University. He is a professor of the State Key Laboratory of Novel Software Technology, and the Department of Computer Science and Technology, Nanjing University. His research interests include software testing, quality and process improvement, software maintenance and evolution, and complex networks.

Jiaqiang Chen received the B.Sc. and M.Sc. degrees in computer science from Nanjing University, Nanjing, China, in 2011, and 2014, respectively. His research interests include software fault prediction, data pre-processing, and machine learning.

Xiang Chen received the B.Sc. degree in the school of management from Xi'an Jiaotong University, China, in 2002. Then he received the M.Sc. and Ph.D. degrees in computer science from Nanjing University, China, in 2008, and 2011, respectively. He is with the Department of Computer Science and Technology at Nantong University as an associate professor. His research interests are mainly in software testing, such as combinatorial testing, regression testing, and fault localization. He has published over 30 papers in referred journals or conferences.

Daoxu Chen was a visiting professor at Purdue University and City University of Hong Kong. He is now a professor in the Department of Computer Science, Nanjing University. His research interests include distributed computing, parallel processing, and computer networks. He has published over 100 research papers in international conference proceedings and journals. He is a member of IEEE and ACM and a senior member of the China Computer Federation.