# Comparing Feature Selection Techniques for Software Quality Estimation Using Data-Sampling-Based Boosting Algorithms

**Article** *in* International Journal of Reliability Quality and Safety Engineering · May 2015

DOI: 10.1142/S0218539315500138

CITATIONS
4

READS
258

**4 authors**, including:

Taghi Khoshgoftaar
Florida Atlantic University
**521** PUBLICATIONS **25,595** CITATIONS

SEE PROFILE

Kehan Gao
Eastern Connecticut State University
**71** PUBLICATIONS **1,437** CITATIONS

SEE PROFILE

International Journal of Reliability, Quality and Safety Engineering
Vol. 22, No. 3 (2015) 1550013 (28 pages)

DOI: 10.1142/S0218539315500138

# Comparing Feature Selection Techniques for Software Quality Estimation Using Data-Sampling-Based Boosting Algorithms

Taghi M. Khoshgoftaar

*Department of Computer and Electrical Engineering and Computer Science*
*Florida Atlantic University, 777 Glades Road*
*Boca Raton, Florida 33431, USA*
*khoshgof@fau.edu*

Kehan Gao* and Ye Chen†

*Department of Mathematics and Computer Science*
*Eastern Connecticut State University, 83 Windham Street*
*Willimantic, Connecticut 06226, USA*
**gaok@easternct.edu*, †*tom.chen@infocore.cn*

Amri Napolitano

*Department of Computer and Electrical Engineering and Computer Science*
*Florida Atlantic University, 777 Glades Road*
*Boca Raton, Florida 33431, USA*
*amrifau@gmail.com*

Received 25 February 2015
Revised 12 April 2015
Accepted 7 May 2015
Published 10 June 2015

Software defect prediction is a classification technique that utilizes software metrics and fault data collected during the software development process to identify fault-prone modules before the testing phase. It aims to optimize project resource allocation and eventually improve the quality of software products. However, two factors, *high dimensionality* and *class imbalance*, may cause low quality training data and subsequently degrade classification models. Feature (software metric) selection and data sampling are frequently used to overcome these problems. Feature selection (FS) is a process of choosing a subset of relevant features so that the quality of prediction models can be maintained or improved. Data sampling alters the dataset to change its balance level, therefore alleviating the problem of traditional classification models that are biased toward the over-represented (majority) class. A recent study shows that another method, called boosting (building multiple models, with each model tuned to work better on instances misclassified by previous models), is also effective for addressing the class imbalance problem. In this paper, we present a technique that uses FS followed by a boosting algorithm in the context of software quality estimation. We investigate four FS approaches: individual FS, repetitive sampled FS, sampled ensemble FS, and repetitive sampled ensemble FS,

*Corresponding author.

and study the impact of the four approaches on the quality of the prediction models. Ten base feature ranking techniques are examined in the case study. We also employ the boosting algorithm to construct classification models with no FS and use the results as the baseline for further comparison. The empirical results demonstrate that (1) FS is important and necessary prior to the learning process; (2) the repetitive sampled FS method generally has similar performance to the individual FS technique; and (3) the ensemble filter (including sampled ensemble filter and repetitive sampled ensemble filter) performs better than or similarly to the average of the corresponding individual base rankers.

*Keywords*: Software quality classification; feature selection; data sampling; RUSBoost; SMOTEBoost.

## 1. Introduction

Software defects and failures occurring during system deployment and operation can lead to severe consequences, especially for some high-assurance and mission-critical systems. One effective method for avoiding such problems is early detection, i.e., to detect and correct software defects (bugs) early enough during the software development life cycle and prior to system deployment and operation. An attractive technique that has been created for this purpose is software quality modeling, where practitioners use software metrics gathered during the software development process and various data mining techniques to construct classification models for predicting whether a given program module is fault-prone (fp) or not-fault-prone (nfp).[1,2] Such predictions can realize optimal project resource allocation, allowing the potentially problematic modules to receive intensive inspection and testing and therefore improving quality of the product.

A significant amount of research has shown that high dimensionality and class imbalance are two factors that may cause low quality of training data and as a result, deteriorate the quality of classification models.[3,4] We are interested in investigating these two problems that also frequently appear in software measurement datasets. *High dimensionality* refers to when a large number of features (attributes or software metrics) are available for building classification models. This may lead to several problems, including high computational cost and memory usage, a decline in prediction performance, and difficulty of understanding and interpreting the model. Feature selection (FS) is often used to deal with this problem. The main goal of FS is to choose a subset of relevant features from the original dataset so that prediction performance of the classification model is maintained or improved, while learning time can significantly drop. FS methods can be categorized as either wrappers or filters based on whether a learning algorithm is involved in the selection process.[5] FS can also be classified into feature subset selection and feature ranking depending on whether attributes are assessed collectively or individually to have good predictive capability.[6] Filter-based feature ranking techniques have been proven as simple, fast, and effective FS methods.[7] A typical filter-based feature ranking technique consists of two steps. First, it ranks features according to a particular method. Then, the features with the highest rankings are selected to construct classification models.

In this work, we study four FS approaches based on 10 individual filter-based feature ranking techniques (also called filters or rankers): individual FS, repetitive sampled FS, sampled ensemble FS, and repetitive sampled ensemble FS. The individual FS method is the simplest: each filter is used separately on the original data, and is used only once. The repetitive sampled FS method also uses only a single filter, but instead of applying this filter directly to the original data, it creates multiple balanced versions from the original dataset using a sampling approach. The same filter is applied to all of these balanced datasets, and the results are aggregated into a single ranked feature list. Sampled ensemble FS uses more than one filter in the process of FS, combining their results through ensemble FS. Similar to the repetitive sampled technique, it starts by applying a sampling technique to the original data in order to create many balanced datasets; but sampled ensemble uses a different ranker for each sampled dataset rather than the same ranker for all of them. Repetitive sampled ensemble FS is implemented using the normal repetitive sampled FS, but instead of using a single ranker as the base ranker, it uses the ensemble of all 10 rankers. Therefore, 10 sets of 10 individual rankings lists are each aggregated individually to produce 10 new rankings lists, which are then aggregated together to produce the final rankings.

In addition to high dimensionality, class imbalance is frequently encountered for a two-class classification problem. *Class imbalance* (in terms of two-class classification) refers to one class (minority) having far fewer instances than the other class (majority). When training data are imbalanced, traditional machine learning algorithms may have difficulty distinguishing between instances of the two classes. They tend to create models that are biased toward the overrepresented (majority) class. Although this may result in high overall accuracy, it rarely results in useful models. For instance, in software engineering practice, most modules have no errors (or no significant errors) during the final stage of system testing, but the accurate detection of the few faulty modules is of most importance; otherwise, it may result in defective software in deployment and operation. Data sampling is often employed to handle the class imbalance problem. We use two different sampling methods: random undersampling (RUS)[8] and synthetic minority oversampling (SMOTE).[9] Boosting is another effective technique for dealing with imbalanced data. Boosting is a meta-learning technique designed to improve the classification performance of weak learners by building multiple models, with each model tuned to work better on instances misclassified by previous models. Although boosting is not specifically developed to handle the class imbalance problem, it has been shown to be very effective in this regard.[8] In this study, we use two hybrid boosting and data sampling approaches, called RUSBoost and SMOTEBoost, depending on which sampling technique, RUS or SMOTE, is integrated into the boosting algorithm (AdaBoost[10] used for this study).

Following FS, we employ the sampled ensemble learning algorithm (RUSBoost or SMOTEBoost). The primary goal of this paper is to investigate the four different FS approaches and examine their impact on the subsequent learning (RUSBoost or

SMOTEBoost) process. To our knowledge, no similar studies have ever done this before. We choose 10 base feature ranking techniques and apply them to the individual FS and repetitive sampled FS approaches. We also use the sampled ensemble FS and the repetitive sampled ensemble FS methods based on the 10 rankers to select the feature subsets. In the case study, we applied all the techniques to three datasets from three separate releases of a real-world software system. All these three datasets exhibit a high degree of class imbalance. We employed five learners to build classification models. The results demonstrate that the repetitive sampled FS technique had similar performance to the individual FS method, on average, and that the ensemble filter performed better than or similarly to the average of the 10 base rankers. We also built classification models using the RUSBoost and SMOTEBoost algorithms without FS and compared the results to those with FS. The empirical result shows that FS is an important step prior to the learning process, and can significantly improve the classification performance.

The rest of the paper is organized as follows: Sec. 2 discusses related work, Sec. 3 provides methodology, including more detailed information about the four FS approaches, 10 base feature ranking techniques, two data sampling methods, two hybrid boosting and data sampling approaches (RUSBoost and SMOTEBoost), five learners, and the performance metric. The datasets used in the case study are described in Sec. 4. Section 5 presents the case study. Section 6 discusses threats to validity. Finally, the conclusion and future work are summarized in Sec. 7.

## 2. Related Work

FS is an effective technique to resolve the problems presented by high dimensionality, and as such it has been heavily researched. Guyon and Elisseeff[11] have pointed out three benefits obtained from FS: (1) improving the prediction performance of the predictors, (2) providing faster and more cost-effective predictors, and (3) providing a better understanding of the underlying process that generated the data. Liu *et al.*[5] provided a comprehensive survey of FS and reviewed its developments with the growth of data mining. FS techniques can be divided into wrappers, filters, and embedded categories.[5] Wrappers use a search algorithm to search through the space of possible feature subsets, evaluate each subset through a learning algorithm, and determine which ones are finally selected in building a classifier. Filters use a simpler statistical measure or some intrinsic characteristic to evaluate each subset or individual feature rather than using a learning algorithm. Embedded approaches, like the wrapper methods, are also linked to a learning algorithm; however this link is much stronger compared to wrappers, as FS is included in the classifier construction in this case. FS can also be categorized as ranking or subset selection.[6] Feature ranking scores the attributes based on their individual predictive power, while subset selection selects subset of attributes that collectively have good prediction capability. In this study, we use filter-based feature ranking techniques.

Currently, FS has been widely applied in a range of fields, such as text categorization, remote sensing, intrusion detection, genomic analysis, and image retrieval.[12] Ilczuk *et al.*[13] highlighted the importance of FS in judging the qualification of patients for cardiac pacemaker implantation. Wald *et al.*[14] applied a number of FS techniques for helping understand which users are most vulnerable to social bots on Twitter. FS also gets more attention in the software quality assurance domain. Akalya *et al.*[15] proposed a hybrid FS model that combines wrapper and filter methods and applied it to NASA's public KC1 dataset obtained from the NASA IV&V Facility Metrics Data Program (MDP) data repository.

In addition to an excess number of attributes, many real-world classification datasets encounter the class imbalance problem. A considerable amount of research has been done to investigate this problem. Weiss *et al.*[16] provided a survey of the class imbalance problem and techniques for reducing the negative impact imbalance on classification performance. Two important techniques discussed for alleviating the problem of class imbalance are data sampling and boosting. Data sampling attempts to overcome imbalanced class distributions by adding instances to (oversampling), or removing instances from (undersampling), the dataset. The simplest form of sampling is random sampling. Besides that, several more intelligent algorithms for sampling data have been proposed, such as SMOTE,[9] Borderline SMOTE,[17] and Wilson's Editing.[18]

Another technique for alleviating class imbalance is boosting,[19] which is a common form of ensemble learning. Although it is not specifically developed to handle the class imbalance problem, it has been shown to be very effective in this regard.[20] The most commonly used boosting algorithm is AdaBoost,[10] which has been shown to improve the performance of any weak (poor) classifier, provided that the classifier results have better performance than random guessing. Several variations[8,21] have been proposed to make AdaBoost improve its performance on imbalanced data. One promising technique is SMOTEBoost.[21] SMOTEBoost combines an intelligent oversampling technique (SMOTE) with AdaBoost, resulting in a highly effective hybrid approach to learning from imbalanced data. Seiffert *et al.* proposed RUSBoost,[8] which is based on the SMOTEBoost algorithm but provides a faster, simpler, yet effective alternative for learning from imbalanced data.

While a great deal of work has been done for FS and data sampling separately, limited research has been done and reported on both together, especially in the context of software quality assurance. Among the few studies, Wahono *et al.*[22] proposed the combination of genetic algorithms with the bagging (bootstrap aggregation) technique for improving the performance of software defect prediction. Genetic algorithms were applied to deal with the FS, and bagging was employed to deal with the class imbalance problem. In one of our recent studies,[23] we used an iterative FS approach which repeatedly applies data sampling (to overcome class imbalance) followed by FS (to overcome high dimensionality), and finally combines the ranked feature lists from the separate iterations of sampling. After FS, models were built either using a plain learner or by using a boosting algorithm which incorporates

sampling. This previous study was more focused on the comparison between the boosting algorithm and the plain learner, and the FS technique used was the repetitive sampled FS (one of the four FS approaches referred in this work). The present research concentrates more on four different FS approaches (including the repetitive sampled FS) and their effects on the boosting algorithm performance. In another recent study,[24] we compared sampled FS (sampling performed before FS) and repetitive sampled FS approaches. Following data preprocessing, a plain learner was applied to the new training datasets. The study concluded that repetitive sampled FS had significantly better performance than the nonrepetitive sampled FS method when a plain learner was applied. The present work also compares the repetitive sampled FS method to the nonrepetitive (individual FS, where no sampling is used prior to FS) technique when both working along with an ensemble learning algorithm and concludes that these two approaches have very similar performance. For this reason, we recommend the use of nonrepetitive (individual FS) method in the external FS step (due to its simplicity) when the ensemble learning algorithm is applied.

## 3. Methodology

The process of combining FS with the ensemble learning approach is presented in Fig. 1. We propose four different approaches for the external FS step. Following FS, we construct classification models using the hybrid boosting and data sampling

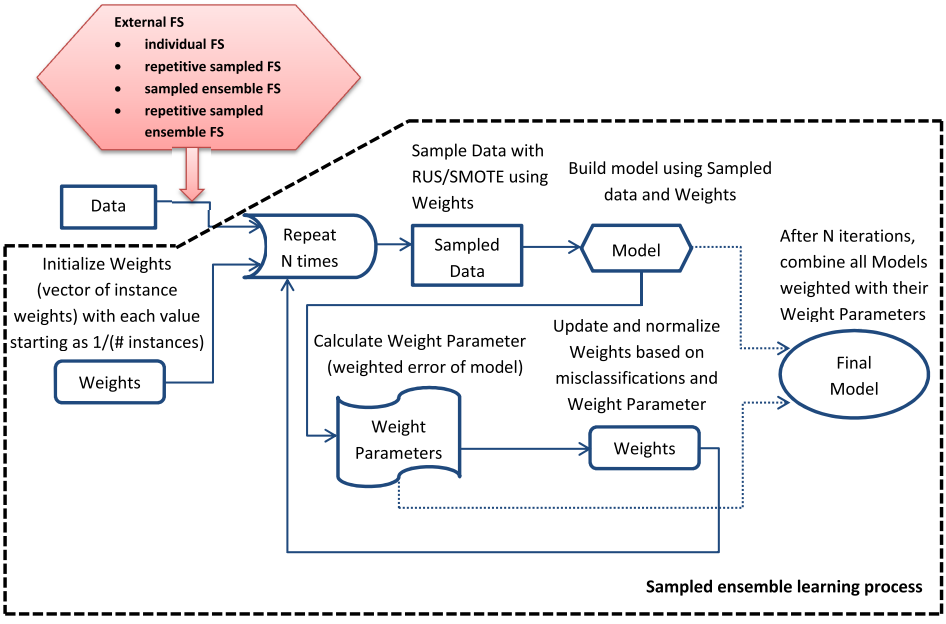



Fig. 1. FS combined with the ensemble learning (RUSBoost/SMOTEBoost) approach.

approach (RUSBoost or SMOTEBoost), in which RUS or SMOTE is embedded into the AdaBoost algorithm. More details about the process are discussed in the following sections.

### 3.1. *External FS*

We study four FS approaches: individual FS, repetitive sampled FS, sampled ensemble FS, and repetitive sampled ensemble FS.

#### 3.1.1. *Individual FS*

Each feature ranking algorithm is used separately on the original data, and is used only once; the results from this single run of a single FS technique are used for all subsequent boosting steps. In this study, we investigate 10 base feature ranking techniques from three different categories: three standard methods, six threshold-based feature selection techniques (TBFS), and one first-order statistics (FOS)-based FS approach. We here give only the names and references of the techniques (as shown in Table 1). More detailed information can be found in Sec. 3.2.

#### 3.1.2. *Repetitive sampled FS*

The repetitive sampled FS method is designed to deal with FS for imbalanced data. Algorithm 1 presents the procedure of this approach. It consists of two basic steps: (1) using a sampling technique to balance data (in this study we used the RUS and SMOTE sampling techniques with post-sampling class ratio 50:50 between the two classes), and (2) applying a filter-based feature ranking technique to the balanced data and ranking all the features according to their predictive powers (scores). In order to alleviate the problem of biased results generated due to the sampling

Table 1. Filter-based feature ranking techniques.

| Abbreviation | Name |
|---|---|
| Standard techniques | |
| CS | Chi squared[24] |
| IG | Information gain[24] |
| RF | ReliefF[24] |
| TBFS techniques | |
| MI | Mutual information[25] |
| KS | Kolmogorov–Smirnov statistic[25] |
| Dev | Deviance[25] |
| GM | Geometric mean[25] |
| ROC | Area under receiver operating characteristic curve[25] |
| PRC | Area under precision–recall curve[25] |
| FOS-based FS techniques | |
| S2N | Signal-to-noise ratio[26] |

**Algorithm 1:** Repetitive Sampled Feature Selection Algorithm

**input** : Dataset $S$ with features $F^j, j = 1, \ldots, m$
: Each instance $\mathbf{x} \in S$ is assigned to one of two classes $c(\mathbf{x}) \in \{fp, nfp\}$
: Filter-based feature ranking technique: $\omega \in$ {CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, S2N}
: Data sampling (balance) technique: $\varphi \in$ {RUS, SMOTE}
: A threshold: number (%) of the features to be selected
: The number of repetitions: $k$
**output**: Selected features

**for** $i = 1$ *to* $k$ **do**
  Use $\varphi$ to balance $S$ and get the balanced data $S_i$
  Employ $\omega$ to rank features $F^j$ on $S_i$, and get new rankings $\omega_i(F^j)$, $j = 1, \ldots, m$.

Create feature ranking $\mathbb{R}$ by combining the $k$ different rankings $\{\omega_i(F^j)|i = 1, \ldots, k\}\ \forall j$ with mean (average).
Select features according to feature ranking $\mathbb{R}$ and a predefined threshold.

**Algorithm 2:** Sampled Ensemble Feature Selection Algorithm

**input** : Dataset $S$ with features $F^j, j = 1, \ldots, m$
: Each instance $\mathbf{x} \in S$ is assigned to one of two classes $c(\mathbf{x}) \in \{fp, nfp\}$
: Base rankers: $\{\omega_i|i = 1, \ldots, r\}$. e.g., {CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, S2N}
: Data sampling (balance) technique: $\varphi \in$ {RUS, SMOTE}
: A threshold: number (%) of the features to be selected
**output**: Selected features

**for** $i = 1$ *to* $r$ **do**
  Use $\varphi$ to balance $S$ and get the balanced data $S_i$
  Employ $\omega_i$ to rank features $F^j$ on $S_i$, and get new rankings $\omega_i(F^j)$, $j = 1, \ldots, m$.

Create feature ranking $\mathbb{R}$ by combining the $r$ different rankings $\{\omega_i(F^j)|i = 1, \ldots, r\}\ \forall j$ with mean (average).
Select features according to feature ranking $\mathbb{R}$ and a predefined threshold.

process, we repeat the two steps $k$ times ($k = 10$ in this study) and aggregate $k$ rankings using the average. Finally, the best set of attributes is selected from the original data to form the training dataset.

#### 3.1.3. *Sampled ensemble FS*

This approach combines different ranking techniques to yield more stable and robust results. Like with the repetitive sampled technique, sampled ensemble samples the original data multiple (e.g., 10) times (with a different random number seed each time so the results differ), but sampled ensemble uses a different ranker for each sampled dataset rather than the same ranker for all of them. Diversity can be achieved by using various rankers.[27] In this study, we use the 10 rankers discussed in Sec. 3.2 to form the ensemble filter. The aggregation method used is arithmetic mean. Algorithm 2 shows the detailed steps for this approach.

**Algorithm 3:** Repetitive Sampled Ensemble Feature Selection Algorithm

**input** : Dataset $S$ with features $F^j, j = 1, \ldots, m$
: Each instance $\mathbf{x} \in S$ is assigned to one of two classes $c(\mathbf{x}) \in \{fp, nfp\}$
: Base rankers: $\{\omega_i | i = 1, \ldots, r\}$. e.g.,{CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, S2N}
: Data sampling (balance) technique: $\varphi \in$ {RUS, SMOTE}
: A threshold: number (%) of the features to be selected
: The number of repetitions: $k$
**output**: Selected features

**for** $i = 1$ *to* $k$ **do**
  Use $\varphi$ to balance $S$ and get the balanced data $S_i$
  **for** $l = 1$ *to* $r$ **do**
    Employ $\omega_l$ to rank features $F^j$ on $S_i$, and get new rankings $\omega_{il}(F^j)$, $j = 1, \ldots, m$.
  Create feature ranking $\omega_i(F^j)$ by combining the $r$ different rankings $\{\omega_{il}(F^j) | l = 1, \ldots, r\}$ $\forall j$ with mean (average).
Create feature ranking $\mathbb{R}$ by combining the $k$ different rankings $\{\omega_i(F^j) | i = 1, \ldots, k\}$ $\forall j$ with mean (average).
Select features according to feature ranking $\mathbb{R}$ and a predefined threshold.

#### 3.1.4. *Repetitive sampled ensemble FS*

This approach is similar to the repetitive sampled FS method. Instead of using single ranker as the base ranker, it uses the ensemble of all 10 rankers. There are two essential steps: (1) applying a sampling technique to the original data in order to create a balanced dataset; and (2) applying a number of rankers (10 in this study) to the same sampled dataset, and aggregating all of the results into a single feature list. The aggregation method used is arithmetic mean. These two steps are repeated 10 times and finally the results are aggregated to produce the final rankings. The procedure of this approach is presented in Algorithm 3.

### 3.2. *Ten base feature ranking techniques*

The 10 base feature ranking techniques are from three different categories: three standard methods, six TBFS techniques, and one FOS-based FS approach (signal-to-noise ratio (S2N)).

#### 3.2.1. *Standard techniques*

The three standard filter-based feature ranking techniques used in this work include: chi-squared (CS), information gain (IG), and ReliefF (RF). All three use the implementation available in the WEKA machine learning tool[28,a] and use default parameters unless otherwise noted.

[a]Waikato Environment for Knowledge Analysis (WEKA) is a popular suite of machine learning software written in Java, developed at the University of Waikato. WEKA is free software available under the GNU General Public License.

**Algorithm 4:** Threshold-Based Feature Selection

**input** :
1. Dataset $S = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, n \text{ and } \ y_i \in \{P, N\}\}$ with features $F^j, j = 1, \ldots, m$, where $P = fp$ and $N = nfp$;
2. The value of attribute $F^j$ for instance $\mathbf{x}_i$ is denoted $F^j(\mathbf{x}_i)$;
3. Metric $\omega \in$ {CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, S2N}.

**output**: Ranking $\mathbb{R} = \{r^1, \ldots, r^m\}$ where $r^j$ represents the rank for attribute $F^j$, i.e., the $r^j$-th most significant attribute as determined by metric $\omega$.

**for** $F^j, j = 1, \ldots, m$ **do**

Normalize $F^j \mapsto \widehat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;

Calculate metric $\omega$ using attribute $\widehat{F}^j$ and class attribute $\{y_i | \ y_i \in \{P, N\}, \ i = 1, \ldots, n\}$, $\omega(\widehat{F}^j)$ ; (The detailed formula of each metric $\omega$ is provided in Section 3.2.2.)

Create attribute ranking $\mathbb{R}$ using $\omega(\widehat{F}^j) \ \forall \, j$

The CS test is used to examine whether two variables are independent. CS is more likely to find significance to the extent that (1) the relationship is strong, (2) the sample size is large, and/or (3) the number of values of the two associated features is large. IG is a measure based on the concept of entropy from information theory. It is the information provided about the target class attribute Y, given the value of another attribute X. IG measures the decrease of the weighted average impurity of the partitions, compared with the impurity of the complete set of data. Relief is an instance-based feature ranking technique. RF is an extension of the Relief algorithm that can handle noise and multiclass datasets.

### 3.2.2. *Threshold-based feature selection*

The TBFS technique was proposed by our research team[29] and implemented within WEKA. The procedure is shown in Algorithm 4. Each independent attribute works individually with the class attribute, and this two-attribute dataset is evaluated using different classification performance metrics. More specifically, the TBFS procedure includes two steps: (1) normalizing the independent attribute values so that they fall between 0 and 1; and (2) treating those values as the posterior probabilities from which to calculate performance metrics. The feature rankers we propose utilize four rates.[b] The value is computed in both directions: first treating instances above the threshold ($t$) as positive and below as negative, then treating instances above the threshold as negative and below as positive. The better result is used. In this manner, the attributes can be ranked from most to least predictive based on

[b]Analogous to the procedure for calculating rates in a classification setting with a posterior probability, the true positive rate, TPR($t$), true negative rate, TNR($t$), and false positive rate, FPR($t$) can be calculated at each threshold $t \in [0, 1]$ relative to the normalized attribute $\widehat{F}^j$. Precision, PRE($t$) is defined as the fraction of the predicted-positive examples which are actually positive.

each metric. Six metrics used in this study are presented as follows:

(a) Mutual Information (MI). Let $c(\mathbf{x}) \in \{P, N\}$ denote the actual class of instance $\mathbf{x}$, and let $\widehat{c}^t(\mathbf{x})$ denote the predicted class based on the value of the attribute $F^j$ and a given threshold $t$. MI computes the mutual information criterion with respect to the number of times a feature value and a class co-occur, the feature value occurs without the class, and the class occurs without the feature value. The MI metric is defined as:

$$\mathrm{MI} = \max_{t\in[0,1]} \sum_{\widehat{c}^t\in\{P,N\}} \sum_{c\in\{P,N\}} p(\widehat{c}^t, c) \log \frac{p(\widehat{c}^t, c)}{p(\widehat{c}^t)p(c)},$$

where

$$p(\widehat{c}^t = \alpha, c = \beta) = \frac{|\{\mathbf{x}|(\widehat{c}^t(\mathbf{x}) = \alpha) \cap (c(\mathbf{x}) = \beta)\}|}{|P| + |N|},$$

$$p(\widehat{c}^t = \alpha) = \frac{|\{\mathbf{x}|\widehat{c}^t(\mathbf{x}) = \alpha\}|}{|P| + |N|},$$

$$p(c = \alpha) = \frac{|\{\mathbf{x}|c(\mathbf{x}) = \alpha\}|}{|P| + |N|},$$

$$\alpha, \beta \in \{P, N\}.$$

(b) Kolmogorov–Smirnov statistic (KS). KS is a measurement of separability. The goal of KS is to measure the maximum difference between the distributions of the members of each class. The formula for the KS statistic is:

$$\mathrm{KS} = \max_{t\in[0,1]} |\mathrm{TPR}(t) - \mathrm{FPR}(t)|.$$

(c) Deviance (Dev). Dev is a metric in which it is the minimum value over all the thresholds that is the chosen value for the attribute. Dev measures the sum of the squared errors from the mean class based on a threshold $t$.

(d) Geometric Mean (GM). GM is a quick and useful metric for FS. The equation for the geometric mean is

$$\mathrm{GM} = \max_{t\in[0,1]} \sqrt{\mathrm{TPR}(t) \times \mathrm{TNR}(t)}.$$

A geometric mean of 1 would mean that the attribute is perfectly correlated. The maximum geometric mean across the thresholds is the score of the attribute.

(e) Area Under the Receiver Operating Characteristics (ROC) Curve. ROC curves graph TPR on the $y$-axis versus the FPR on the $x$-axis. The resulting curve illustrates the trade-off between TPR and FPR. In this study, ROC curves are generated by varying the decision threshold $t$ used to transform the normalized attribute values into a predicted class. The area under the ROC ranges from 0 to 1, and an attribute with more predictive power results in an area under the ROC closer to 1.

(f) Area Under the Precision–Recall Curve (PRC). PRC is a single-value measure that originated from the area of information retrieval. A PRC is generated by varying the decision threshold $t$ from 0 to 1 and plotting the recall (equivalent to TPR) on the $y$-axis and precision on the $x$-axis at each point in a similar manner to the ROC curve. The area under the PRC ranges from 0 to 1, and an attribute with more predictive power results in an area under the PRC closer to 1.

#### 3.2.3. *Signal-to-noise ratio*

S2N[30] is a simple univariate ranking technique which defines how well a feature discriminates between two classes in a two class problem. This method belongs to FOS-based FS category.[26] S2N, for a given feature, separates the means of the two classes relative to the sum of their standard deviation. The formula to calculate S2N is

$$\text{S2N} = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N},$$

where $\mu_P$ and $\mu_N$ are the mean values of a particular attribute for the samples from class $P$ and class $N$, and $\sigma_P$ and $\sigma_N$ are the corresponding standard deviations. The larger the S2N value, the more relevant the feature is to the class attribute.

### 3.3. *Sampling techniques*

The two data sampling techniques used in this study are RUS and SMOTE. RUS alleviates the problem with class imbalance in a dataset by randomly discarding instances from the majority class (nfp class for our study). SMOTE is an intelligent oversampling method proposed by Chawla *et al.*[9] SMOTE adds new, artificial minority examples by extrapolating between preexisting minority instances rather than simply duplicating original examples. The newly created instances cause the minority regions of the feature-space to become fuller and more general.

### 3.4. *The RUSBoost and SMOTEBoost techniques*

RUSBoost combines RUS and boosting for improving classification performance. Boosting is a meta-learning technique designed to improve the classification performance of weak learners by iteratively creating an ensemble of weak hypotheses which are combined to predict the class of unlabeled examples. This technique uses AdaBoost,[10] a well-known boosting algorithm shown to improve the classification performance of weak classifiers. Initially, all examples in the training dataset are assigned equal weights. During each iteration of AdaBoost, a weak hypothesis is formed by the base learner. The error associated with the hypothesis is calculated and the weight of each example is adjusted such that misclassified examples have their weights increased while correctly classified examples have their weights

decreased. Therefore, subsequent iterations of boosting will generate hypotheses that are more likely to correctly classify the previously mislabeled examples. After all iterations are completed, a weighted vote of all hypotheses are used to assign a class to unlabeled examples. In this study, the boosting algorithm is performed using 10 iterations. RUSBoost applies the same steps as the regular boosting, but prior to constructing the weak hypothesis during each round of boosting, RUS is applied to the training data to achieve a more balanced class distribution. The base learners used are naïve Bayes (NB), multilayer perceptron (MLP), $k$-nearest neighbors (KNN), support vector machine (SVM), and logistic regression (LR) (discussed in the next section). The procedure of RUSBoost is described in part of Fig. 1. More details about the algorithm can be found in the work of Seiffert *et al.*[8]

SMOTEBoost has the same mechanism as RUSBoost. Instead of using RUS, SMOTEBoost is produced by combining SMOTE with boosting with the same purpose of improving classification performance on imbalanced datasets. For the complete algorithm, one can refer to the SMOTEBoost technique developed by Chawla *et al.*[21]

### 3.5. *Learners*

The software defect prediction models are built using five different classification algorithms, including NB,[28] MLP,[31] KNN,[28] SVM,[32] and LR.[33] These learners were selected because they do not have a built-in FS capability and they are commonly used in both the software engineering and data mining domains.[24,34,35] We employ WEKA to implement these classifiers. Parameter settings are subject to the data explored. Unless stated, we use default parameter settings as specified in WEKA.

#### 3.5.1. *Naïve Bayes*

To determine the classification of an instance, one method is to use a probability model in which the features that were chosen by the feature rankers are used as the conditions for the probability of the sample being a member of the class. A basic probability model would look like $p(C|F_1, \ldots, F_n)$ where $F_i$ is the value of each feature used and $C$ is the class of the instance. This model is known as the posterior, and we assign the instance to the class for which it has the largest posterior.[36]

Unfortunately, it is quite difficult to determine the posterior directly. Thus, it is necessary to use Bayes's rule which states that the posterior equals the ratio of the prior multiplied by the likelihood over the evidence, or

$$p(C|F_1, \ldots, F_n) = \frac{p(C)p(F_1, \ldots, F_n|C)}{p(F_1, \ldots, F_n)}.$$

In reality, the formula above can be simplified by certain assumptions. The evidence, $p(F_1, \ldots, F_n)$, is always constant for the specific dataset and therefore can be

ignored for the purposes of classification. The likelihood formula, $p(F_1, \ldots, F_n|C)$, can be simplified to $\prod_i p(F_i|C)$ due to the naive assumption that all of the features are conditionally independent of all of other features. This naïve assumption with the removal of the evidence parameter creates the NB classifier.

$$p(C|F_1, \ldots, F_n) = p(C) \prod_i p(F_i|C).$$

3.5.2. *Multilayer perceptron*

A MLP is a type of artificial neural network. MLP consist of nodes which are arranged in sets called layers. Each node in a layer has a connection coming from every node in the layer before it and to every node in the layer after it. Each node takes the weighted sum of all of the input nodes. Along with the weighted sums, an activation function is also applied. The application of the activation function to the result of the weighted sum allows for a more clearly defined result by further separating the instances in the two classes from each other. Neural networks are well known for being robust to redundant features. However, neural networks sometimes have problems with overfitting.[31] In our study, the `hiddenLayers` parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the `validationSetSize` parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process.

3.5.3. *K-nearest neighbors*

The KNN learner is an example of an instance-based and lazy learning algorithm. Instance based algorithms use only the training data without creating statistics on which to base their hypotheses. The KNN learner does this by calculating the distance of the test sample from every training instance, and the predicted class is derived from the KNN.

In the KNN learner, when we get a test sample we would like to classify, we tabulate the classes for each of the $k$ closest training samples (we used a $k$ of five for our experiment) and we determine the weight of each neighbor by taking a measurement of $\frac{1}{\text{distance}}$ where "distance" is the distance from the test sample. After the classes and weights are tabulated, we add all of the weights from the neighbors of the positive class together and all of the weights of the negative class together. The prediction will be the class with the largest cumulative weight.[36]

The KNN learner can use any metric that is appropriate to calculate the distance between the samples. The standard metric used in KNN is Euclidean Distance, defined as

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

3.5.4. *Support vector machine*

One of the most efficient ways to classify between two classes is to assume that both classes are linearly separated from each other. This assumption allows us to use a discriminant to split the instances into the two classes before looking at the distribution between the classes. A linear discriminant uses the formula $g(x|\mathbf{w}, \omega_0) = \mathbf{w}^T x + \omega_0$. In the case of the linear discriminant, we only need to learn the weight vector, $\mathbf{w}$, and the bias, $\omega_0$. One aspect that must be addressed is that there can be multiple discriminants that correctly classify the two classes. The SVM is a linear discriminant classifier which assumes that the best discriminant maximizes the distance between the two classes. This is measured in the distance from the discriminant to the samples of both classes.[37] In our study, the complexity parameter $c$ was set to '5.0', while the `buildLogisticModels` parameter was set to 'true' to produce proper probability estimates.

3.5.5. *Logistic regression*

LR is a statistical technique that can be used to solve binary classification problems. Based on the training data, a LR model is created which is used to decide the class membership of future instances.

### 3.6. *Performance metric*

The area under the ROC curve (i.e., AUC) is one of the most widely used single numeric measures that provides a general idea of the predictive potential of the classifier. The ROC curve graphs TPR versus the FPR (the positive class is always the minority class in these experiments). Traditional performance metrics for classifier evaluation consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. A perfect classifier provides an AUC that equals 1. AUC is of lower variance and is more reliable than other performance metrics such as precision, recall, and F-measure.[35]

Note that this measurement has two uses in this study, one of which is a measurement in a TBFS technique that has been discussed in Sec. 3.2.2, it is called ROC, while the other use is presented here called AUC that is used to evaluate the prediction performance of the classification models.

## 4. Datasets

In our experiments, we used publicly available data, namely the Eclipse defect counts and complexity metrics dataset obtained from the PROMISE data repository (http://promisedata.org). In particular, we used the metrics and defects data at the software packages level. The original data for the Eclipse packages consists of

Table 2. Data characteristics.

| Release | Threshold | #Attributes | #Instances | fp | | nfp | |
|---|---|---|---|---|---|---|---|
| | | | | # | % | # | % |
| 2.0 | 10 | 209 | 377 | 23 | 6.1 | 354 | 93.9 |
| 2.1 | 5 | 209 | 434 | 34 | 7.8 | 400 | 92.2 |
| 3.0 | 10 | 209 | 661 | 41 | 6.2 | 620 | 93.8 |

three releases denoted 2.0, 2.1, and 3.0 for each release, as reported by Zimmerman *et al.*[38] Membership in each class is determined by a post-release defects threshold, which separates fp from nfp packages by classifying packages with threshold or more post-release defects as fp and the remaining as nfp. In our study, we used threshold = 10 for releases 2.0 and 3.0 and threshold = 5 for release 2.1. The reason why a different threshold was chosen for release 2.1 is that we would like to keep similar class distributions for all three datasets. All datasets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table 2 shows the characteristics of the datasets after transformation for each group. These datasets exhibit a high degree of class imbalance.

## 5. A Case Study

### 5.1. *Design*

The main goal of this paper is to study four different FS approaches (individual FS, repetitive sampled FS, sampled ensemble FS, and repetitive sampled ensemble FS) and compare their effects on the hybrid boosting and data sampling approach. More specifically, we would like to investigate the 10 base feature ranking techniques and apply them to the individual FS and the repetitive sampled FS approaches. We also want to examine the sampled ensemble FS and the repetitive sampled ensemble FS based on the 10 individual ranking techniques. Following FS, we build classification models using two sampled ensemble learning approaches (RUSBoost and SMOTEBoost) and present the results separately.

In the experiment, the 10 base feature ranking methods used are CS, IG, RF, MI, KS, Dev, GM, ROC, PRC, and S2N. The sampling techniques used are RUS and SMOTE approaches. Note that the sampling technique used in the external FS step, if applicable, is consistent with the sampling technique used in the ensemble learning process. More specifically, if RUS is used in the external FS step, then RUSBoost is performed right after, while if SMOTE is used in the FS step, SMOTEBoost is carried out in the subsequent learning process. The post-sampling class ratio (between fp and nfp modules) was set to 50:50 throughout the experiment. In addition, five (base) learners, NB, MLP, KNN, SVM, and LR were used in the boosting process. The number of features selected in the feature subsets was set to $\lceil \log_2 n \rceil = 8$,[39] where $n$ is the number of independent attributes in the original dataset ($n = 208$ in this experiment). For all experiments, we employed 10 runs of 5-fold cross-validation (CV). That is, for each run the data was randomly divided

into five folds, one of which was used as the test data while the other four folds were used as training data. All the preprocessing steps (FS and data sampling) were done on the training dataset. The processed training data was then used to build the classification model and the resulting model was applied to the test fold. This CV was repeated five times (the folds), with each fold used exactly once as the test data. The five results from the five folds then was averaged to produce a single estimation. In order to lower the variance of the CV result, we repeated the CV with new random splits 10 times. The final estimation is the average results over the 10 runs of 5-fold CV. We used AUC for evaluating the performance of learners.

### 5.2. *Results of FS on the RUSBoost method*

The results of the four FS approaches combined with the RUSBoost algorithm averaged over the group of datasets (in terms of AUC) are reported in Table 3. The 'Individual' and 'Repetitive' columns present the respective results of the individual FS and repetitive sampled FS approaches for all 10 base rankers and across all five learners. The sampled ensemble FS and repetitive sampled ensemble FS results are shown at the last row of each block of the table, corresponding to the Individual and Repetitive columns. Each value has a superscript that represents the rank of the filter among the 11 techniques (10 base rankers and the ensemble based on the 10).

We also performed a statistical analysis using the two tailed student's $t$-test to compare the individual FS technique with the repetitive sampled FS method. The $t$-test examines the null hypothesis that the population means related to two independent group samples are equal against the alternative hypothesis that the population means are different. The statistical analysis result for each pairwise comparison is presented in the table via a $p$-value. The significance level was set to 0.05. When the $p$-value is less than 0.05, the two group means are significantly different from each other, and $p$-value is highlighted in bold in the table. As can be seen, the different FS approaches (individual FS versus repetitive sampled FS) did not affect the performance of any base rankers except the RF ranker, where the repetitive sampled FS approach presented much better performance than the individual FS approach for all five learners. In addition, ROC and S2N exhibited better performance than other rankers on average. The table (in the 'NFS' row) also shows the performance of the classification models built based on the RUSBoost algorithm only, in which no external FS step has been performed prior to the learning process. The results demonstrate that FS greatly improved classification performance.

Figure 2 provides the comparison between sampled ensemble filter and the average of the 10 individual filters, and between the repetitive sampled ensemble filter and the average of the 10 repetitive sampled filters. The chart intuitively demonstrates that the ensemble method performed slightly better than or similarly to the average across all five learners.

Table 3. Classification performance of FS on the RUSBoost method.

| FS approach | NB | | | MLP | | |
|---|---|---|---|---|---|---|
| | Individual | Repetitive | $p$-value | Individual | Repetitive | $p$-value |
| NFS | 0.8166 | — | — | 0.8466 | — | — |
| CS | $0.8000^{11}$ | $0.8217^{10}$ | 0.191 | $0.8847^{8}$ | $0.8833^{8}$ | 0.911 |
| IG | $0.8409^{3}$ | $0.8311^{5}$ | 0.458 | $0.8873^{7}$ | $0.8881^{4}$ | 0.944 |
| RF | $0.8204^{10}$ | $0.8656^{1}$ | **0.000** | $0.8636^{11}$ | $0.8964^{3}$ | **0.000** |
| MI | $0.8246^{7}$ | $0.8228^{9}$ | 0.902 | $0.8900^{5}$ | $0.8798^{9}$ | 0.346 |
| KS | $0.8294^{5}$ | $0.8287^{7}$ | 0.962 | $0.8745^{9}$ | $0.8757^{11}$ | 0.928 |
| Dev | $0.8222^{9}$ | $0.8257^{8}$ | 0.803 | $0.8881^{6}$ | $0.8855^{5}$ | 0.832 |
| GM | $0.8247^{6}$ | $0.8111^{11}$ | 0.382 | $0.8721^{10}$ | $0.8765^{10}$ | 0.739 |
| ROC | $0.8532^{1}$ | $0.8506^{2}$ | 0.777 | $0.8983^{2}$ | $0.8866^{2}$ | 0.242 |
| PRC | $0.8239^{8}$ | $0.8296^{6}$ | 0.660 | $0.8908^{4}$ | $0.8839^{6}$ | 0.532 |
| S2N | $0.8449^{2}$ | $0.8485^{3}$ | 0.708 | $0.9005^{1}$ | $0.8931^{1}$ | 0.300 |
| Ensemble | $0.8399^{4}$ | $0.8379^{4}$ | 0.888 | $0.8908^{3}$ | $0.8835^{7}$ | 0.513 |
| FS approach | KNN | | | SVM | | |
| | Individual | Repetitive | $p$-value | Individual | Repetitive | $p$-value |
| NFS | 0.8022 | — | — | 0.8043 | — | — |
| CS | $0.8768^{8}$ | $0.8807^{7}$ | 0.733 | $0.8980^{8}$ | $0.9012^{7}$ | 0.762 |
| IG | $0.8891^{3}$ | $0.8808^{6}$ | 0.451 | $0.9077^{4}$ | $0.9024^{6}$ | 0.603 |
| RF | $0.8100^{11}$ | $0.8889^{3}$ | **0.000** | $0.8725^{11}$ | $0.9096^{1}$ | **0.000** |
| MI | $0.8821^{7}$ | $0.8725^{9}$ | 0.421 | $0.9045^{7}$ | $0.8998^{9}$ | 0.619 |
| KS | $0.8674^{9}$ | $0.8706^{10}$ | 0.820 | $0.8953^{9}$ | $0.8953^{10}$ | 1.000 |
| Dev | $0.8839^{6}$ | $0.8763^{8}$ | 0.498 | $0.9054^{5}$ | $0.8998^{8}$ | 0.575 |
| GM | $0.8673^{10}$ | $0.8672^{11}$ | 0.998 | $0.8927^{10}$ | $0.8939^{11}$ | 0.912 |
| ROC | $0.8985^{1}$ | $0.8921^{1}$ | 0.474 | $0.9106^{1}$ | $0.9094^{2}$ | 0.876 |
| PRC | $0.8922^{2}$ | $0.8870^{4}$ | 0.604 | $0.9085^{3}$ | $0.9081^{4}$ | 0.958 |
| S2N | $0.8887^{4}$ | $0.8895^{2}$ | 0.922 | $0.9089^{2}$ | $0.9089^{3}$ | 0.999 |
| Ensemble | $0.8846^{5}$ | $0.8855^{5}$ | 0.936 | $0.9045^{6}$ | $0.9036^{5}$ | 0.921 |
| FS approach | LR | | | | | |
| | Individual | Repetitive | $p$-value | | | |
| NFS | 0.7908 | — | — | | | |
| CS | $0.8598^{11}$ | $0.8743^{3}$ | 0.361 | | | |
| IG | $0.8850^{3}$ | $0.8681^{10}$ | 0.208 | | | |
| RF | $0.8723^{7}$ | $0.8937^{1}$ | **0.004** | | | |
| MI | $0.8853^{2}$ | $0.8669^{11}$ | 0.108 | | | |
| KS | $0.8613^{10}$ | $0.8716^{6}$ | 0.482 | | | |
| Dev | $0.8710^{8}$ | $0.8737^{5}$ | 0.839 | | | |
| GM | $0.8655^{9}$ | $0.8699^{9}$ | 0.736 | | | |
| ROC | $0.8759^{4}$ | $0.8741^{4}$ | 0.880 | | | |
| PRC | $0.8732^{5}$ | $0.8710^{7}$ | 0.855 | | | |
| S2N | $0.8955^{1}$ | $0.8927^{2}$ | 0.715 | | | |
| Ensemble | $0.8729^{6}$ | $0.8700^{8}$ | 0.805 | | | |

Afterwards, we conducted a three-way Analysis Of Variance (ANOVA) $F$-test on the classification performance to examine if the performance difference (better/worse) is statistically significant or not. This test gives us results across all factors on a per-factor basis, rather than simply looking at one factor for each

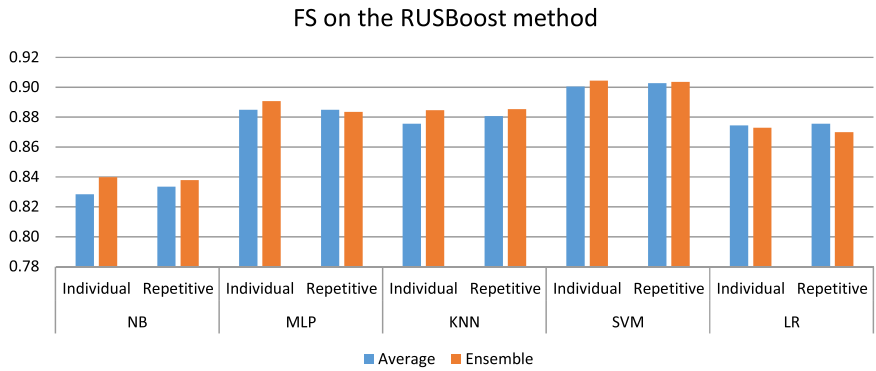


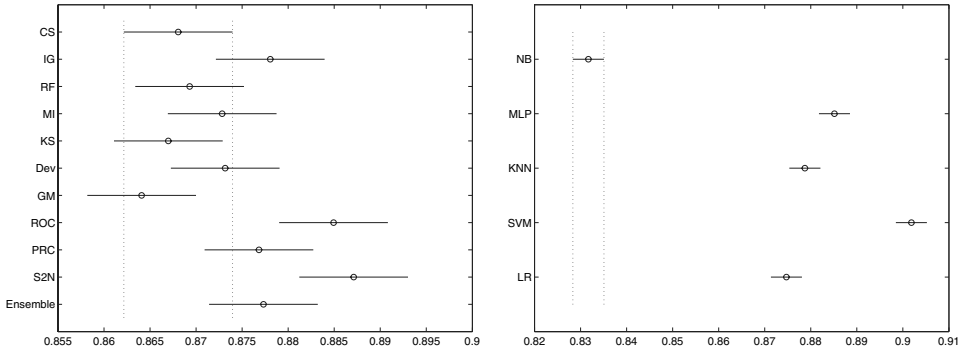

Fig. 2. Comparison between the ensemble and average of the 10 base filters for the individual FS and the repetitive sampled FS approaches (for FS on the RUSBoost method).

choice of ranker, learner, or FS technique (as is given by the two tailed $t$-tests). The three factors include A (isRepetitive): whether repetition is involved in the external FS step, B: 11 rankers, including 10 base rankers and the ensemble filter based on the 10, and C: five learners. The ANOVA model can test the null hypothesis that all the group population means are equal against the alternative hypothesis that at least one group mean is different. If the alternative hypothesis is accepted, multiple comparisons can be used to determine which of the means are different from the others. Table 4 shows the ANOVA results. The $p$-value is less than the cutoff 0.05 for the ranker and learner factors, meaning that for each factor at least two group means were significantly different from each other. The $p$-value of the isRepetitive factor is greater than 0.05, implying that for the external FS, whether repetition was involved did not affect the classification performance.

We further carried out a multiple comparison test on each main factor that has $p$-value less than 0.05 with Tukey's Honestly Significant Difference (HSD) criterion. Note that for all the ANOVA and multiple comparison tests, the significance level was set to 0.05. Figure 3 shows the multiple comparisons for Factors B and C. The figures display graphs with each group mean represented by a symbol (∘) and 95% confidence interval as a line around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The assumptions for constructing ANOVA and Tukey's HSD models were validated. From these figures we can see the following points:

Table 4. ANOVA for FS on the RUSBoost method.

| Source | Sum sq. | d.f. | Mean sq. | $F$ | $p$ |
|---|---|---|---|---|---|
| isRepetitive | 0.0041 | 1 | 0.0041 | 2.05 | 0.153 |
| Ranker | 0.1595 | 10 | 0.0160 | 7.91 | 0.000 |
| Learner | 1.7893 | 4 | 0.4473 | 221.73 | 0.000 |
| Error | 6.6251 | 3,284 | 0.0020 | | |
| Total | 8.5780 | 3,299 | | | |




(a) (b)

Fig. 3. Multiple comparison for FS on the RUSBoost method. (a) Factor B: Ranker and (b) Factor C: Learner.

- Among the 11 feature ranking techniques, some base rankers, like ROC and S2N, demonstrated better or significantly better performance than other rankers; ensemble, IG, and PRC ranked right behind and had similar performances; then followed Dev, MI, RF, CS, KS, and GM.
- Among the five learners, SVM performed the best, followed by MLP, KNN, and LR; NB had the worst performance.

### 5.3. *Results of FS on the SMOTEBoost method*

The results of the four FS approaches combined with the SMOTEBoost algorithm averaged over the three datasets are presented in Table 5. The 'Individual' and 'Repetitive' columns show the respective results of the individual FS approach and repetitive sampled FS approach for all 10 base rankers and ensemble rankers as well. The superscript number next to each AUC value represents the rank of the filter among the 11 techniques (10 base rankers and the ensemble based on the 10). We also compared the individual FS technique with the repetitive sampled FS method using the two tailed Student's $t$-test and reported the results (via $p$-values) in the table. The significance level was still set to 0.05. When the $p$-value is less than 0.05 (highlighted in bold), the two group means are different from each other, and no obvious difference otherwise. As can be observed, the different FS approaches (individual FS versus repetitive sampled FS) did not affect the performance of most base rankers. However, they did affect some rankers, like RF and PRC in the NB learner, and KS and GM in the MLP and KNN learners, where the repetitive sampled FS approach displayed significantly better performance than the individual FS approach. Only for the IG ranker in the KNN learner did the individual FS approach exhibit significantly better classification performance than the repetitive sampled FS approach. The table also shows the performance of the classification models built using the SMOTEBoost algorithm only (denoted as 'NFS' in the table),

Table 5. Classification performance of FS on the SMOTEBoost method.

| FS approach | NB | | | MLP | | |
|---|---|---|---|---|---|---|
| | Individual | Repetitive | $p$-value | Individual | Repetitive | $p$-value |
| NFS | 0.8050 | — | — | 0.8306 | — | — |
| CS | $0.8135^{7}$ | $0.8320^{10}$ | 0.124 | $0.8646^{8}$ | $0.8654^{9}$ | 0.927 |
| IG | $0.8285^{6}$ | $0.8389^{7}$ | 0.282 | $0.8733^{5}$ | $0.8645^{10}$ | 0.351 |
| RF | $0.8082^{10}$ | $0.8417^{6}$ | **0.000** | $0.8438^{11}$ | $0.8407^{11}$ | 0.735 |
| MI | $0.8350^{4}$ | $0.8429^{5}$ | 0.512 | $0.8692^{6}$ | $0.8812^{4}$ | 0.140 |
| KS | $0.8121^{8}$ | $0.8324^{9}$ | 0.193 | $0.8567^{10}$ | $0.8799^{6}$ | **0.046** |
| Dev | $0.8310^{5}$ | $0.8340^{8}$ | 0.806 | $0.8689^{7}$ | $0.8782^{7}$ | 0.357 |
| GM | $0.8013^{11}$ | $0.8304^{11}$ | 0.058 | $0.8581^{9}$ | $0.8820^{3}$ | **0.042** |
| ROC | $0.8446^{3}$ | $0.8539^{1}$ | 0.307 | $0.8750^{4}$ | $0.8810^{5}$ | 0.487 |
| PRC | $0.8116^{9}$ | $0.8434^{4}$ | **0.002** | $0.8788^{3}$ | $0.8848^{1}$ | 0.486 |
| S2N | $0.8555^{1}$ | $0.8495^{2}$ | 0.500 | $0.8810^{2}$ | $0.8772^{8}$ | 0.579 |
| Ensemble | $0.8538^{2}$ | $0.8481^{3}$ | 0.566 | $0.8865^{1}$ | $0.8820^{2}$ | 0.456 |
| FS approach | KNN | | | SVM | | |
| | Individual | Repetitive | $p$-value | Individual | Repetitive | $p$-value |
| NFS | 0.7743 | — | | 0.8125 | — | — |
| CS | $0.8581^{8}$ | $0.8478^{9}$ | 0.198 | $0.8880^{8}$ | $0.8810^{9}$ | 0.599 |
| IG | $0.8758^{3}$ | $0.8464^{10}$ | **0.000** | $0.9006^{5}$ | $0.8756^{10}$ | 0.057 |
| RF | $0.7971^{11}$ | $0.7950^{11}$ | 0.864 | $0.8765^{11}$ | $0.8752^{11}$ | 0.885 |
| MI | $0.8671^{7}$ | $0.8761^{5}$ | 0.252 | $0.9032^{4}$ | $0.8992^{5}$ | 0.718 |
| KS | $0.8558^{9}$ | $0.8776^{4}$ | **0.023** | $0.8866^{9}$ | $0.8975^{7}$ | 0.414 |
| Dev | $0.8684^{5}$ | $0.8783^{3}$ | 0.193 | $0.8950^{7}$ | $0.8964^{8}$ | 0.910 |
| GM | $0.8492^{10}$ | $0.8733^{6}$ | **0.019** | $0.8857^{10}$ | $0.8984^{6}$ | 0.341 |
| ROC | $0.8846^{1}$ | $0.8859^{2}$ | 0.776 | $0.9032^{3}$ | $0.9068^{2}$ | 0.697 |
| PRC | $0.8783^{2}$ | $0.8865^{1}$ | 0.240 | $0.8980^{6}$ | $0.9096^{1}$ | 0.221 |
| S2N | $0.8734^{4}$ | $0.8701^{8}$ | 0.484 | $0.9101^{1}$ | $0.9030^{4}$ | 0.375 |
| Ensemble | $0.8682^{6}$ | $0.8705^{7}$ | 0.639 | $0.9039^{2}$ | $0.9048^{3}$ | 0.920 |
| FS approach | LR | | | | | |
| | Individual | Repetitive | $p$-value | | | |
| NFS | 0.7598 | — | — | | | |
| CS | $0.8659^{8}$ | $0.8725^{10}$ | 0.640 | | | |
| IG | $0.8761^{4}$ | $0.8753^{9}$ | 0.950 | | | |
| RF | $0.8590^{11}$ | $0.8639^{11}$ | 0.570 | | | |
| MI | $0.8838^{3}$ | $0.8788^{5}$ | 0.620 | | | |
| KS | $0.8674^{7}$ | $0.8770^{6}$ | 0.472 | | | |
| Dev | $0.8680^{6}$ | $0.8764^{8}$ | 0.524 | | | |
| GM | $0.8644^{9}$ | $0.8769^{7}$ | 0.320 | | | |
| ROC | $0.8693^{5}$ | $0.8858^{2}$ | 0.119 | | | |
| PRC | $0.8617^{10}$ | $0.8853^{4}$ | 0.052 | | | |
| S2N | $0.8964^{1}$ | $0.8963^{1}$ | 0.989 | | | |
| Ensemble | $0.8859^{2}$ | $0.8856^{3}$ | 0.982 | | | |

in which no FS was performed prior to the learning process. The results once again demonstrate that FS does help for improving classification performance.

Figure 4 provides the comparison between sampled ensemble filter and the average of the 10 individual filters, and between the repetitive sampled ensemble filter

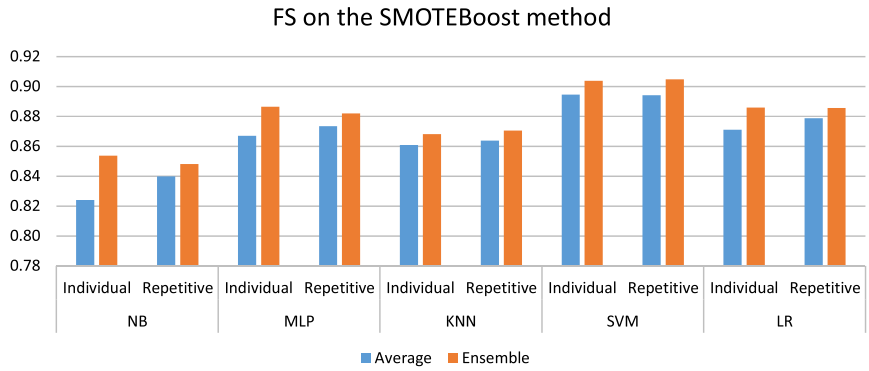



Fig. 4. Comparison between the ensemble and average of the 10 base filters for the individual FS and the repetitive sampled FS approaches (for FS on the SMOTEBoost method).

and the average of the 10 repetitive sampled filters. The chart demonstrates that unlike the result from FS on the RUSBoost method, the ensemble technique here always performed better than the average of the base rankers irrespective of which FS approach, an individual filter or a repetitive filter, was selected. This is true for all five learners.

Like the previous experiment, we conducted a three-way ANOVA test on the classification performance to examine if the performance difference is statistically significant or not. The three factors used are the same as we did previously, including A (isRepetitive): whether repetition is involved in the FS step, B: 11 rankers, and C: five learners. Table 6 shows the ANOVA results. The $p$-value is less than the cutoff 0.05 for all three factors, meaning that for each factor at least two group means were significantly different from each other. If the alternative hypothesis is accepted, multiple comparisons can be used to determine which of the means are different from others.

We therefore carried out a multiple comparison test on each main factor with Tukey's HSD criterion. Note that for all the ANOVA and multiple comparison tests, the significance level was set to 0.05. Figure 5 shows the multiple comparisons for Factors A, B, and C. The assumptions for constructing ANOVA and Tukey's HSD models were also validated. From these figures we can see the following points:

- The repetitive FS methods showed significantly better performance than the individual FS methods.

Table 6. ANOVA for FS on the SMOTEBoost method.

| Source | Sum sq. | d.f. | Mean sq. | $F$ | $p$ |
|---|---|---|---|---|---|
| isRepetitive | 0.0274 | 1 | 0.0274 | 16.76 | 0.000 |
| Ranker | 0.4199 | 10 | 0.0420 | 25.65 | 0.000 |
| Learner | 1.3355 | 4 | 0.3339 | 204.01 | 0.000 |
| Error | 5.3747 | 3,284 | 0.0016 | | |
| Total | 7.1575 | 3,299 | | | |

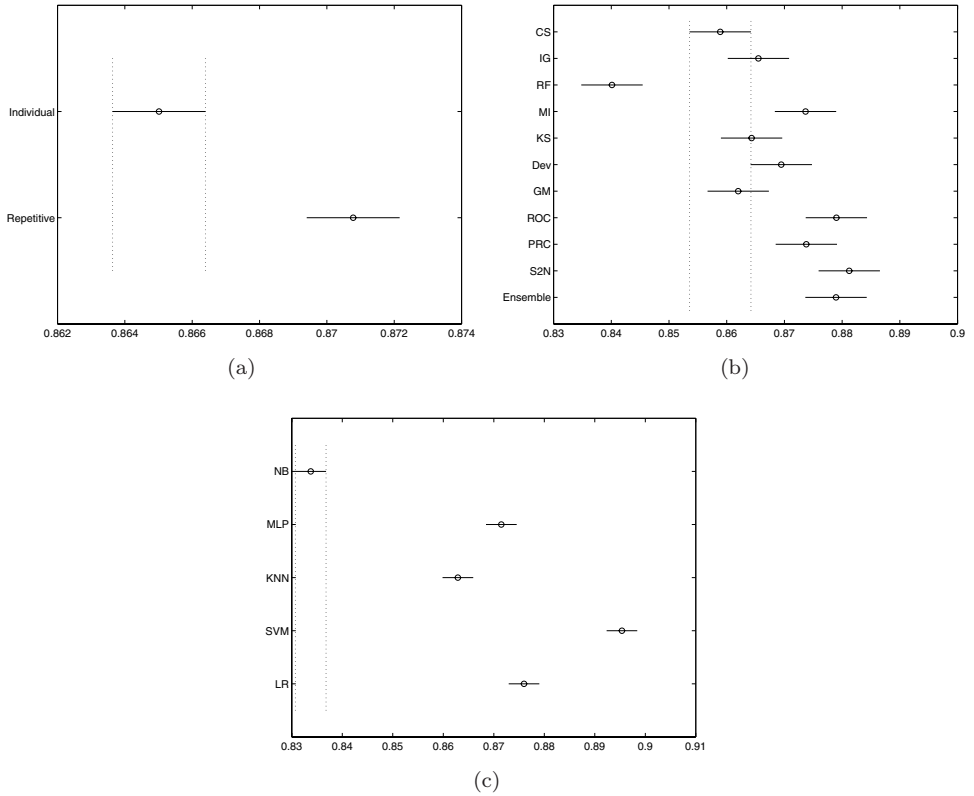



Fig. 5. Multiple comparison for FS on the SMOTEBoost method. (a) Factor A: isRepetitive, (b) Factor B: Ranker and (c) Factor C: Learner.

- Between the individual FS and repetitive sampled FS approaches, the repetitive method showed better performance than the individual method.
- Among the 11 feature ranking techniques, some base rankers like ROC, S2N, and Ensemble demonstrated better or significantly better performance than other rankers; MI, PRC, and Dev ranked right behind and had similar performances; then followed IG, KS, GM, and CS. RF showed the worst performance.
- Among the five learners, SVM once again performed the best, followed by LR, MLP, and KNN; NB still had the worst performance.

## 6. Threats to Validity

A typical software development project is very human intensive, which can affect many areas of the development process, including software quality and defect occurrence. Therefore, software engineering research that utilizes controlled experiments for evaluating the usefulness of empirical models is not practical. The case study presented is an empirical software engineering effort, for which the software

engineering community demands that its subject has the following characteristics[40,41]: (1) developed by a group, and not by an individual; (2) be as large as industry size projects, and not a toy problem; (3) developed by professionals, and not by amateur programmers; and (4) developed in an industry/government organization setting, and not in a laboratory.

The software systems that are used in our case study were developed by professionals in a large software development organization using an established software development process and management practices. The software was developed to address real-world problems. We note that our case studies fulfill all of the above criteria specified by the software engineering community.

Threats to external validity are conditions that limit generalization of case study results. The analysis and conclusion presented in this paper are based upon the metrics and defect data obtained from a Java-based Eclipse software project, in which three datasets from three different releases were used. These strengthen the generalization of our empirical results. But it must be recognized that the same (as our approach) analysis for another software system, especially from a different application domain, may provide relatively different results — a likely threat in all empirical software engineering research. However, a software quality practitioner would appreciate the process of developing a useful defect predictor in the presence of the problem of class imbalance and/or when there are a large number of software metrics to work with. The proposed strategy of integrating data sampling, FS, and ensemble learning (boosting) can be extended to any software system, especially high-assurance and mission-critical software. Moreover, as all our final conclusions are based on 10 runs of five-fold CV and statistical tests for significance, our findings are grounded in using sound methods. Finally, it is observed that the type of classifier used for software quality prediction greatly affected the prediction results for the case studies. Hence, software quality practitioners should not rule out considering other classification algorithms when applying the proposed approach. Also, the selection of specific parameter settings for a given classifier is likely to cause the classifier to analyze the training data differently.

Threats to internal validity for a software engineering experiment are unaccounted for influences that may affect case study results. In the context of this study, poor fault-proneness estimates can be caused by a wide variety of factors, including measurement errors while collecting and recording software metrics, modeling errors due to the unskilled use of software applications, errors in model-selection during the modeling process, and the presence of outliers and noise in the training dataset. Measurement errors are inherent to the data collection effort of a given software project. In our study, common model-building and model-evaluation approaches that are used for all combinations of data sampling technique, FS method, and classifier have been adopted. Moreover, the experiments and statistical analysis were performed by only one skilled person in order to keep modeling errors to a minimum.

A software engineering domain expert is consulted to set the number of software metrics to select from the original set after ranking the software metrics. This

number may be different for another project. However, based on our extensive prior work in software quality estimation and quantitative software engineering, we are confident the selection of $\lceil \log_2 n \rceil$ metrics in the attribute subset is large enough to capture the quality-based characteristics of most software projects. We have found very often that only a handful of project-specific metrics are relevant for defect prediction, and that in many cases, very few metrics (among the available set) are selected by the learner in the final software quality prediction model.

## 7. Conclusion

In order to overcome the high dimensionality and class imbalance problems that affect software quality classification, we proposed a technique that uses FS followed by a hybrid boosting and data sampling approach. We studied four FS approaches: individual FS, repetitive sampled FS, sampled ensemble FS, and repetitive sampled ensemble FS. Following FS, models were built using the sampled ensemble learning algorithm (RUSBoost or SMOTEBoost), where AdaBoost incorporates RUS or SMOTE. We were interested in studying the effect of various FS methods on software quality prediction.

In the empirical study, we applied these techniques to three datasets from a real-world software system. We employed 10 base feature ranking techniques (rankers) and apply them to the individual FS and repetitive sampled FS approaches. We also implemented the sampled ensemble FS and the repetitive sampled ensemble FS based on the 10 rankers. We built classification models using five learners. The results demonstrate that the repetitive sampled FS technique generally had similar performance as the individual FS method for most rankers, however, the repetitive method can achieve greater performance than individual FS for some specific ranker in some particular environment, like RF working along with RUSBoost. Therefore, we recommend the use of nonrepetitive method in the external FS step when FS works along with the ensemble learning algorithm unless you know that some specific ranker (like RF) will get benefit from the repetitive process. The results also show that some individual ranking techniques, such as ROC, S2N, and PRC, resulted in relatively better performance than other rankers and consequently are recommended in this study. The ensemble filter also exhibited competitive performance on average, and more importantly, it has stable and consistent performance with respect to various learners and datasets; thus, we recommend using the ensemble technique when each individual ranker's characteristics are unknown and a relatively large computational cost for building models is allowed. Finally, we constructed classification models using the RUSBoost and SMOTEBoost algorithms without FS and compared the results to those with FS. The empirical results show that FS is a very important and necessary step prior to the learning process for both algorithms. Of the five learners, SVM performed the best, followed by MLP, KNN, and LR; NB had the worst prediction.

Future work will consider subset selection combined with these data-sampling-based boosting algorithms. In addition, more empirical studies with software measurement and defect data from other software projects will be conducted in the future.

## References


1. Q. Song, Z. Jia, M. Shepperd, S. Ying and J. Liu, A general software defect-proneness prediction framework, *IEEE Trans. Softw. Eng.* **37**(3) (2011) 356–370.
2. A. K. Pandey and N. K. Goyal, Predicting fault-prone software module using data mining technique and fuzzy logic, *Int. J. Comp. Comm. Technol.* **2** (2–4) (2010), pp. 56–63.
3. K. Gao, T. M. Khoshgoftaar and N. Seliya, Predicting high-risk program modules by selecting the right software measurements, *Softw. Qual. J.* **20**(1) (2012) 3–42.
4. S. Maldonado, R. Weber and F. Famili, Feature selection for high-dimensional class-imbalanced data sets using support vector machines, *Inf. Sci.* **286**(1) (2014) 228–246.
5. H. Liu, H. Motoda, R. Setiono and Z. Zhao, Feature selection: An ever evolving frontier in data mining, in *Proc. Fourth Int. Workshop on Feature Selection in Data Mining*, Hyderabad, India (2010), pp. 4–13.
6. S. S. Rathore and A. Gupta, A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction, in *Proc. 7th India Software Engineering Conf.*, Chennai, India, 19–21 February 2014.
7. 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.* **22**(2) (2012) 161–183.
8. C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse and A. Napolitano, Rusboost: A hybrid approach to alleviating class imbalance, *IEEE Trans. Syst., Man, Cybern. A, Syst. Hum.* **40**(1) (2010) 185–197.
9. N. V. Chawla, K. W. Bowyer, L. O. Hall and P. W. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, *J. Artif. Intell. Res.* **16** (2002) 321–357.
10. Y. Freund and R. E. Schapire, Experiments with a new boosting algorithm, in *Proc. 13th Int. Conf. Machine Learning* (1996), pp. 148–156.
11. I. Guyon and A. Elisseeff, An introduction to variable and feature selection, *J. Mach. Learn. Res.* **3** (2003) 1157–1182.
12. V. Kumar and S. Minz, Feature selection: A literature review, *Smart Comput. Rev.* **4**(3) (2014) 211–229.
13. G. Ilczuk, R. Mlynarski, W. Kargul and A. Wakulicz-Deja, New feature selection methods for qualification of the patients for cardiac pacemaker implantation, *Computers in Cardiology*, 2007, Durham, NC, USA (2007), pp. 423–426.
14. R. Wald, T. M. Khoshgoftaar, A. Napolitano and C. Sumner, Predicting susceptibility to social bots on twitter, in *Proc. 14th IEEE Int. Conf. Information Reuse and Integration* (*IRI2013*), San Francisco, CA, USA (2013), pp. 6–13.
15. C. Akalya devi, K. E. Kannammal and B. Surendiran, A hybrid feature selection model for software fault prediction, *Int. J. Comput. Sci. Appl.* **2**(2) (2012) 25–35.
16. G. M. Weiss, Mining with rarity: A unifying framework, *SIGKDD Explorations* **6**(1) (2004) 7–19.
17. H. Han, W. Y. Wang and B. H. Mao, Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning, *Int. Conf. Intelligent Computing* (*ICIC*'05), Lecture Notes in Computer Science, Vol. 3644 (Springer-Verlag, 2005), pp. 878–887.

18. R. Barandela, R. M. Valdovinos, J. S. Sanchez and F. J. Ferri, The imbalanced training sample problem: Under or over sampling? *Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition* (*SSPR/SPR'*04), Lecture Notes in Computer Science, Vol. 3138 (2004), pp. 806–814.
19. R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*, Adaptive Computation and Machine Learning Series (The MIT Press, 2012).
20. C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse and A. Napolitano, Building useful models from imbalanced data with sampling and boosting, in *Proc.* 21*st Int. Florida Artificial Intelligence Research Society Conference* (*FLAIRS*-2008), Coconut Grove, Florida, May 2008 (AAAI Press), pp. 306–311.
21. N. V. Chawla, A. Lazarevic, L. O. Hall and K. Bowyer, SMOTEBoost: Improving prediction of the minority class in boosting, in *Proc. Principles of Knowledge Discovery in Databases* (2003), pp. 107–119.
22. R. S. Wahono, N. Suryana and S. Ahmad, Metaheuristic optimization based feature selection for software defect prediction, *J. Softw.* **9**(5) (2014) 1324–1333.
23. K. Gao, T. M. Khoshgoftaar and R. Wald, The use of under- and oversampling within ensemble feature selection and classification for software quality prediction, *Int. J. Reliab. Qual. Saf. Eng.* **21**(1) (2014), doi:10.1142/S0218539314500041.
24. 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.* **16**(5) (2014) 801–822.
25. A. A. Shanab, T. M. Khoshgoftaar and R. Wald, Robustness of threshold-based feature rankers with data sampling on noisy and imbalanced data, *FLAIRS Conf.* (2012), pp. 92–97.
26. H. Wang, T. M. Khoshgoftaar, R. Wald and A. Napolitano, A study on first order statistics-based feature selection techniques on software metric data, in *Proc.* 25*th Int. Conf. Software Engineering and Knowledge Engineering*, Boston, MA, USA, 27–29 June 2013, pp. 467–472.
27. J. S. Olsson and D. W. Oard, Combining feature selectors for text classification, in *Proc.* 15*th ACM Int. Conf. Information and Knowledge Management* (2006), pp. 798–799.
28. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. (Morgan Kaufmann, 2005).
29. J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano and R. Wald, Threshold-based feature selection techniques for high-dimensional bioinformatics data, *Netw. Model. Anal. Health Inform. Bioinform.* **1**(1–2) (2012) 47–61.
30. L. Goh, Q. Song and N. Kasabov, A novel feature selection method to improve classification of gene expression data, in *Proc. Second Conf. Asia-Pacific Bioinformatics*, Dunedin, New Zealand (2004), pp. 161–166.
31. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd edn. (Prentice-Hall, 1998).
32. J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd edn. (Cambridge University Press, 2000).
33. S. Le Cessie and J. C. Van Houwelingen, Ridge estimators in logistic regression, *Appl. Stat.* **41**(1) (1992) 191–201.
34. 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.* **34**(4) (2008) 485–496.
35. Y. Jiang, J. Lin, B. Cukic and T. Menzies, Variance analysis in software fault prediction models, in *Proc. 20th IEEE Int. Symp. Software Reliability Engineering*, Bangalore-Mysore, India, 16–19 November 2009, pp. 99–108.

36. J. Souza, N. Japkowicz and S. Matwin, Stochfs: A framework for combining feature selection outcomes through a stochastic process, in *Knowledge Discovery in Databases*: *PKDD* 2005, Vol. 3721 (2005), pp. 667–674.
37. T.-Y. Liu, Easy ensemble and feature selection for imbalance data sets, in *Proc.* 2009 *Int. Joint Conf. Bioinformatics*, *Systems Biology and Intelligent Computing* (IEEE Computer Society, Washington, DC, USA, 2009), pp. 517–520.
38. T. Zimmermann, R. Premraj and A. Zeller, Predicting defects for eclipse, in *Proc.* 29*th Int. Conf. Software Engineering Workshops* (IEEE Computer Society, Washington, DC, USA, 2007), p. 76.
39. K. Gao, T. M. Khoshgoftaar, H. Wang and N. Seliya, Choosing software metrics for defect prediction: An investigation on feature selection techniques, software: Practice and experience, *Pract. Aspects Search-Based Softw. Eng.* **41** (2011) 579–606.
40. C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wesslen, *Experimentation in Software Engineering*: *An Introduction*, Kluwer International Series in Software Engineering (Kluwer Academic Publishers, Boston, MA, 2000).
41. L. G. Votta and A. A. Porter, Experimental software engineering: A report on the state of the art, in *Proc.* 17*th Int. Conf. Software Engineering*, April 2005 (IEEE Computer Society, Seattle, WA, USA), pp. 277–279.

## About the Authors

Taghi M. Khoshgoftaar is Motorola Endowed Chair Professor of the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University and the Director of the Data Mining and Machine Learning Laboratory, and Empirical Software Engineering Laboratory. His research interests are in Big Data Analytics, Data Mining and Machine Learning, Health Informatics and Bioinformatics, and Software Engineering. He has published more than 500 refereed journal and conference papers in these areas. He was the conference chair of the *IEEE International Conference on Bioinformatics and Bioengineering* (*BIBE* 2014). He is the workshop chair of the *IEEE IRI Health Informatics Workshop* (2015). Also, he is the Co-Editor-in-Chief of the *Big Data* journal.

Kehan Gao is an Associate Professor at the Department of Mathematics and Computer Science, Eastern Connecticut State University. Her research interests include Software Engineering, Software Metrics, Software Reliability, and Quality Engineering, Computational Intelligence, Data Mining and Machine Learning, and Big Data. She is a member of the IEEE and IEEE Computer Society.

Ye Chen is a Software Architect at InfoCore LLC, China. He received his M.S. degree in Computer Engineering from Florida Atlantic University, Boca Raton, FL, USA in 2000. His research interests include Software Engineering, Cloud Computing, Big Data, and Data Mining.

Amri Napolitano received the Ph.D. and M.S. degrees in Computer Science from Florida Atlantic University in 2006 and 2009, respectively, and the B.S. degree in Computer and Information Science from the University of Florida in 2004. His research interests include Data Mining and Machine Learning, Evolutionary Computation, and Artificial Intelligence.