Comparing Two Approaches for Adding Feature Ranking to Sampled Ensemble Learning for Software Quality Estimation

Kehan Gao
Eastern Connecticut State University
gaok@easternct.edu

Taghi M. Khoshgoftaar & Amri Napolitano
Florida Atlantic University
khoshgof@fau.edu, amrifau@gmail.com

***Abstract*—High dimensionality and class imbalance are two main problems that affect the quality of training datasets in software defect prediction, resulting in inefficient classification models. Feature selection and data sampling are often used to overcome these problems. Feature selection is a process of choosing the most important attributes from the original data set. Data sampling alters the data set to change its balance level. Another technique, called boosting (building multiple models, with each model tuned to work better on instances misclassified by previous models), is found also effective for resolving the class imbalance problem. In particular, RUSBoost, which integrates random undersampling with AdaBoost, has been shown to improve classification performance for imbalanced training data sets. In this study, we investigated an approach for combining feature selection with this ensemble learning (boosting) process. We focused on two different scenarios: feature selection performed prior to the boosting process and feature selection performed inside the boosting process. Ten base feature ranking techniques and an ensemble ranker based on the ten were examined and compared over the two scenarios. The experimental results demonstrate that the ensemble feature ranking method generally had better or similar performance than the average of the base ranking techniques, and more importantly, the ensemble method exhibited better robustness than any other base ranking technique. As for the two scenarios, the results show that applying feature selection inside boosting performed better than using feature selection prior to boosting.**

***Index Terms*—software defect prediction, feature selection, data sampling, boosting, base ranker, ensemble ranker**

I. Introduction

Software defect prediction is a process of utilizing classification models to classify program modules into quality-based classes, e.g., fault-prone (*fp*) or not-fault-prone (*nfp*) [1]. This kind of estimation can help practitioners effectively allocate limited project resources, making them focus on program modules that are of poor quality or likely to have a high number of faults. Classification models are usually built using software metrics such as code-level measurements and defect data, along with data mining techniques. A considerable amount of research has shown that prediction accuracy of a classification model is affected by the quality of the input (training) data. We are interested in investigating two common problems, high dimensionality and class imbalance, that exist in many software measurement data sets.

High dimensionality occurs when a large number of variables (features or software metrics) are available for building classification models. Several problems may arise due to high dimensionality, including longer learning time, a decline in prediction performance, and difficulty of understanding and interpreting the model. The purpose of feature selection is to choose the most important attributes from the original data set so that prediction performance will be improved, or at least maintained, while learning time is significantly reduced. Recent research [2] has shown that filter-based feature ranking techniques are simple, fast, and effective methods for dealing with this problem. In this study, we examined ten filter-based feature ranking techniques and an ensemble ranker based on the ten.

Class imbalance is a separate problem, wherein the class ratio is especially skewed. In a two-class classification problem, class imbalance manifests as one class (minority) having far fewer instances than the other class (majority). The main disadvantage of such imbalanced training data is that a traditional classification algorithm would tend to classify minority class instances (usually a class of interest, e.g., *fp* modules) as majority class (e.g., *nfp* modules) in order to increase overall prediction accuracy. In software quality engineering, this may result in buggy software in deployment and operation, thereby causing serious consequences and high repair cost.

Many solutions have been proposed to address the class imbalance problem. A simple but effective method is random undersampling (RUS), where instances from the majority class are randomly discarded until a certain balance (ratio) between the majority and the minority classes is achieved. Another technique, called boosting, is also effective to cope with the class imbalance problem. 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 [3]. In this study, we use an ensemble learning approach, called RUSBoost, in which random undersampling (RUS) is integrated into AdaBoost [4].

To deal with both high dimensionality and class imbalance, we studied an approach which combines feature selection with an ensemble learning method (RUSBoost). We investigated two different scenarios: feature selection performed right before RUSBoost and feature selection performed inside RUSBoost. In this study, we are interested in learning the impact of feature selection on the final classification results. More specifically, we would like to investigate ten base feature ranking techniques and the ensemble ranker based on the ten, and compare their behaviors, including prediction performance and stability with respect to different learners and data sets. The experiment was conducted over the two different scenarios discussed. We used two groups of data sets from a real-world software system, all of which exhibit significant class imbalance between the two classes (*fp* and *nfp*). Five different learners were used to build classification models. The experimental results demonstrate that the ensemble ranker resulted in better or similar prediction performance than the average of the ten base rankers. In addition, the ensemble method displayed more stable behavior than most base rankers including the best base ranking technique. As to the two different scenarios of the approach, the result shows that feature selection performed inside RUSBoost provided better classification performance than when it was applied prior to RUSBoost.

The rest of the paper is organized as follows. Section II discusses related work. Section III provides methodology, including more detailed information about the feature selection, ensemble-

based boosting technique, learners, and performance metric applied in this study. The data sets used in the experiments are described in Section IV. Section V presents the experimental results. Finally, the conclusion and future work are summarized in Section VI.

II. Related Work

Significant research has been dedicated towards feature selection [5], and applied to a range of fields. Van Hulse et al. [2] proposed 11 new threshold-based feature selection techniques and applied them to 17 different bioinformatics datasets. Yu et al. applied feature selection to gene expression microarray data and studied the stability of feature selection via sample weighting [6].

In addition to excess number of features, many datasets are plagued with the class imbalance problem. Two techniques that have been discussed for addressing this problem are data sampling and boosting. The simplest form of sampling is random sampling. In addition, a few more intelligent algorithms for sampling data have been proposed [7]. The most commonly used boosting algorithm is AdaBoost [4]. Several variations have been proposed to improve AdaBoost's performance on imbalanced data. One promising technique is RUSBoost [3], which is a highly effective hybrid approach to learning from imbalanced data.

While a great deal of effort has been dedicated toward feature selection and data sampling separately for many years, research working on both together is starting to receive more attention [8]. Yang et al. [9] proposed an ensemble-based wrapper approach for feature selection from data with highly imbalanced class distribution. In this work, we create an approach that combines feature selection with RUSBoost and study two different combination scenarios. We also compare different base feature ranking techniques and the ensemble ranker over the two scenarios.

III. Methodology

A. Filter-Based Feature Ranking Techniques

The goal of feature ranking is to score each feature according to a particular method, allowing the selection of the best features. In this study, we investigated ten individual feature ranking techniques from three different categories: three standard methods, six threshold-based feature selection techniques, and the signal to noise ratio approach. In addition, we studied the use of an ensemble of feature ranking techniques.

1) Standard Techniques: The three standard filter-based feature ranking techniques used in this work include: chi-squared, information gain, and ReliefF. All three use the implementation found in the WEKA tool[1] [10] with default parameters. The chi-squared (CS) test evaluates the worth of a feature by computing the value of the chi-squared statistic with respect to the class. The null hypothesis is the assumption that the two features are unrelated, and it is tested by the chi-squared (χ^2) formula: $\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$, where O_{ij} is the observed frequency and E_{ij} is the expected (theoretical) frequency, asserted by the null hypothesis. The greater the value of χ^2, the greater the evidence against the null hypothesis. Information gain (IG) is a measure based on the concept of entropy from information theory. IG 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. ReliefF is an extension of the relief algorithm that can handle noise and multiclass data sets.

[1]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 1: 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 ω.

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 ω 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 ω is provided in Section III-A2.)

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

2) Threshold-Based Feature Selection: The threshold-based feature selection (TBFS) technique was proposed by our research team and implemented within WEKA [2]. The procedure is shown in Algorithm 1. Each independent attribute works individually with the class attribute, and this two-attribute data set is evaluated using different classification performance metrics. More specifically, the TBFS procedure includes two steps: (1) normalizing the 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[2]. 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 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:

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

where

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

[2]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.

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:

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

c. Deviance (Dev). Dev, like GI, is a metric in which it is the minimum value over all the thresholds that is the chosen value for the attribute. Deviance 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 feature selection. The equation for the geometric mean is

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

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 ROC Curve (ROC). Receiver Operating Characteristic, or ROC, curves graph true positive rate on the y-axis versus the false positive rate on the x-axis. The resulting curve illustrates the trade-off between true positive rate and false positive rate. 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 precision-recall curve is generated by varying the decision threshold t from 0 to 1 and plotting the recall (equivalent to true positive rate) 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) Signal to Noise Ratio: Signal to noise ratio (S2N) [11] is a simple univariate ranking technique which defines how well a feature discriminates between two classes in a two class problem. 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 S2N $= \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}$, where μ_P and μ_N are the mean values of a particular attribute for the samples from class P and class N, and σ_P and σ_N are the corresponding standard deviations. The larger the S2N value, the more relevant the feature is to the class attribute.

4) Ensemble Filter Technique: This approach combines different ranking techniques to yield more stable and robust results. The procedure of the ensemble technique includes two essential steps. First, a set of different ranking lists is created using corresponding filter-based rankers and input to the next combining step; second, these ranking lists are integrated using an aggregation technique. Diversity can be achieved by using various rankers [12]. In this study, we use the ten filters discussed above to form the ensemble filter. The aggregation method used is arithmetic mean, where each feature's score is determined by the average of the ranking scores of the feature in each ranking list. Finally, the highest ranked attributes are selected to be used.

B. The RUSBoost Technique

RUSBoost combines random undersampling (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 study uses AdaBoost [4], a well-known boosting algorithm shown to improve the classification performance of weak classifiers. Initially, all examples in the training data set 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, random undersampling is applied to the training data to achieve a more balanced class distribution. The base learners used in this work are NB, MLP, KNN, SVM, and LR, and these will be discussed in the next section. The procedure of RUSBoost is described in part of Figure 1. More details about the algorithm can be found in [3].

C. Learners

The software defect prediction models in this study are built using five different classification algorithms, including Naïve Bayes (NB) [10], MultiLayer Perceptron (MLP) [13], K Nearest Neighbors (KNN) [10], Support Vector Machine (SVM) [14], and Logistic Regression (LR) [10]. Due to space limitations, we refer interested readers to these references to understand how these commonly-used learners function. The WEKA tool is used to instantiate the different classifiers. Generally, the default parameter settings for the different learners are used (for NB and LR), except for the below-mentioned changes. A preliminary investigation in the context of this study indicated that the modified parameter settings are appropriate.

In the case of MLP, 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 for use as a validation set to determine when to stop the iterative training process. For the KNN learner, the `distanceWeighting` parameter was set to 'Weight by 1/distance', the `kNN` parameter was set to '5', and the `crossValidate` parameter was turned on (set to 'true'). In the case of SVM, two changes were made: the `complexity constant c` was set to '5.0', and `build Logistic Models` was set to 'true'. A linear kernel was used by default.

D. Performance Metric

One of the most popular methods for evaluating the performance of learners built using imbalanced data is *receiver operating characteristic*, or ROC, curves. The ROC curve illustrates the performance of a classifier across the complete range of possible decision thresholds, and accordingly does not assume any particular misclassification costs or class prior probabilities. The area under the ROC curve (AUC) is used to provide a single numerical metric for comparing model performances. The AUC value ranges from 0 to 1.

IV. Datasets

In our experiments, we used publicly available data, namely the Eclipse defect counts and complexity metrics data set obtained from

TABLE I
DATA CHARACTERISTICS

Dataset	Rel.	thd	#Attr.	#Inst.	#fp	%fp	#nfp	%nfp
Eclipse 1	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
Eclipse 2	2.0	5	209	377	52	13.8	325	86.2
	2.1	4	209	434	50	11.5	384	88.5
	3.0	5	209	661	98	14.8	563	85.2

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 three releases denoted 2.0, 2.1, and 3.0, each release reported by Zimmerman et al. [15]. Membership in each class is determined by a post-release defects threshold thd, which separates *fp* from *nfp* packages by classifying packages with thd or more post-release defects as *fp* and the remaining as *nfp*. In our study, we used $thd = \{10, 5\}$ for releases 2.0 and 3.0 and $thd = \{5, 4\}$ for release 2.1. This resulted in two groups. Each group contains three data sets, one for each release. The reason why a different set of thresholds was chosen for release 2.1 is that we would like to keep similar class distributions for the data sets in the same group. All data sets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table I shows the characteristics of the data sets after transformation for each group.

V. EXPERIMENTS

A. Design

The main objective of this study is to evaluate the individual filters and the ensemble filter. More specifically, we would like to investigate the ten individual filters (as discussed in Section III-A) with the ensemble filter based on those ten. This comparison was carried out in two different scenarios of feature selection combined with the ensemble learning (boosting) approach. The process of the ensemble learning approach is presented in Figure 1. The two different scenarios are highlighted at top.

- **Scenario 1**: External Feature Selection (EFS), i.e., feature selection performed **prior to** the ensemble learning process.
- **Scenario 2**: Internal Feature Selection (IFS), i.e., feature selection performed **inside** the ensemble learning process.

In the experiment, 11 feature selection methods, including ten base rankers (CS, IG, RF, MI, KS, Dev, GM, ROC, PRC and S2N) and an ensemble ranker based on the ten, were applied. The sampling technique used was the random undersampling (RUS) approach. 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 $\lceil \log_2 n \rceil = 8$, where n is the number of independent attributes in the original data set ($n = 208$ in this experiment). For all experiments, we employed ten runs of five-fold cross-validation. 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 (feature selection and data sampling) were done on the training data set. The processed training data was then used to build the classification model and the resulting model was applied to the test fold. This cross-validation 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.

TABLE II
CLASSIFICATION PERFORMANCE FOR EXTERNAL FEATURE SELECTION

(a) Eclipse 1

Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8399^{4}	0.8908^{3}	0.8846^{5}	0.9045^{6}	0.8729^{6}
CS	0.8000^{11}	0.8847^{8}	0.8768^{8}	0.8980^{8}	0.8598^{11}
IG	0.8409^{3}	0.8873^{7}	0.8891^{3}	0.9077^{4}	0.8850^{3}
RF	0.8204^{10}	0.8636^{11}	0.8100^{11}	0.8725^{11}	0.8723^{7}
MI	0.8246^{7}	0.8900^{5}	0.8821^{7}	0.9045^{7}	0.8853^{2}
KS	0.8294^{5}	0.8745^{9}	0.8674^{9}	0.8953^{9}	0.8613^{10}
Dev	0.8222^{9}	0.8881^{6}	0.8839^{6}	0.9054^{5}	0.8710^{8}
GM	0.8247^{6}	0.8721^{10}	0.8673^{10}	0.8927^{10}	0.8655^{9}
ROC	0.8532^{1}	0.8983^{2}	0.8985^{1}	0.9106^{1}	0.8759^{4}
PRC	0.8239^{8}	0.8908^{4}	0.8922^{2}	0.9085^{3}	0.8732^{5}
S2N	0.8449^{2}	0.9005^{1}	0.8887^{4}	0.9089^{2}	0.8955^{1}

(b) Eclipse 2

Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8364^{7}	0.9084^{5}	0.8962^{6}	0.9183^{7}	0.9108^{2}
CS	0.8236^{11}	0.9076^{6}	0.8942^{7}	0.9183^{8}	0.9047^{9}
IG	0.8275^{10}	0.9090^{4}	0.8971^{3}	0.9210^{2}	0.9061^{7}
RF	0.8319^{9}	0.8587^{11}	0.8056^{11}	0.8752^{11}	0.8747^{11}
MI	0.8424^{3}	0.9058^{8}	0.8963^{5}	0.9210^{1}	0.9090^{4}
KS	0.8346^{8}	0.9040^{9}	0.8925^{9}	0.9185^{6}	0.9069^{6}
Dev	0.8395^{5}	0.9101^{2}	0.8986^{2}	0.9193^{5}	0.9089^{5}
GM	0.8414^{4}	0.9015^{10}	0.8905^{10}	0.9169^{9}	0.9044^{10}
ROC	0.8427^{2}	0.9125^{1}	0.8971^{1}	0.9209^{3}	0.9120^{1}
PRC	0.8390^{6}	0.9091^{3}	0.8970^{4}	0.9194^{4}	0.9060^{8}
S2N	0.8533^{1}	0.9066^{7}	0.8927^{8}	0.9136^{10}	0.9098^{3}

B. Results and Analysis

The results of the two scenarios (EFS and IFS) averaged over the respective group of data sets (in terms of AUC) are reported in Tables II and III, each containing the results for all five learners. The tables show the classification performance of the ten individual filters as well as the ensemble filter based on the ten. Each value has a superscript that represents the rank of the filter among the 11 techniques. For instance, the ensemble filter ranks #4 among all the 11 ranking techniques when NB was employed on Eclipse 1 for the EFS scenario. From the results, we can see that the performance of the rankers were related to learners, training data set, and working scenarios. There was no method that always performed the best in all circumstances. For example, ROC performed relatively better than other rankers for the EFS scenario, while it performed averagely for the IFS scenario. IG performed best for Eclipse 1 in IFS when the MLP and SVM learners were employed, however showed the worst prediction among the 11 rankers when NB was used. The ensemble filter generally performed better than average of the individual base rankers and also displayed more stable performance than most base rankers (as the rank deviation of the ensemble filter is the second smallest among 11, GM had the smallest rank deviation, however, GM performed below the average). Figure 2 provides the comparison between the ensemble filter and the average of the ten base rankers for a given scenario and a data set across all five learners. The charts intuitively demonstrate that the ensemble method performed better than or similarly to the average.

We further carried out 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. The three factors include: A representing two different scenarios of the approach, B representing the five learners, and C representing the ten base rankers and the ensemble method. The null hypothesis for the ANOVA test is that all the group population means are the same,

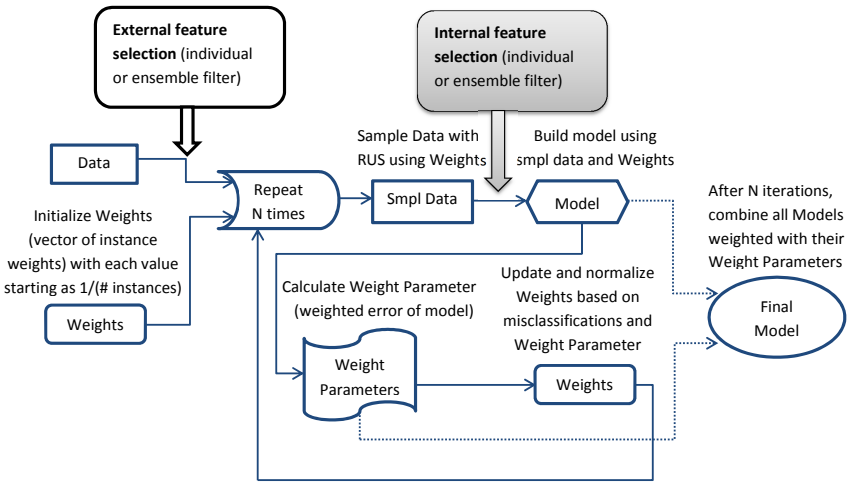


Fig. 1. Two scenarios of feature selection combined with the ensemble learning approach

TABLE III
CLASSIFICATION PERFORMANCE FOR INTERNAL FEATURE SELECTION

(a) Eclipse 1

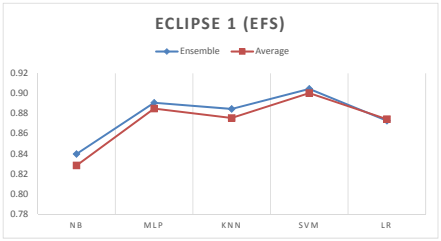
Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8561^{3}	0.8985^{5}	0.9033^{2}	0.9105^{2}	0.8803^{5}
CS	0.8398^{10}	0.8931^{9}	0.8935^{10}	0.9096^{3}	0.8746^{11}
IG	0.8385^{11}	0.9042^{1}	0.9031^{3}	0.9159^{1}	0.8840^{2}
RF	0.8563^{2}	0.8874^{11}	0.8919^{11}	0.8975^{11}	0.8784^{8}
MI	0.8469^{8}	0.8935^{8}	0.8953^{9}	0.9078^{6}	0.8834^{3}
KS	0.8508^{7}	0.8966^{7}	0.8966^{7}	0.9087^{5}	0.8802^{6}
Dev	0.8551^{4}	0.8988^{4}	0.8988^{6}	0.9094^{4}	0.8798^{7}
GM	0.8511^{6}	0.8922^{10}	0.8956^{8}	0.9054^{8}	0.8755^{10}
ROC	0.8543^{5}	0.9006^{3}	0.8998^{5}	0.9076^{7}	0.8762^{9}
PRC	0.8669^{1}	0.8967^{6}	0.9035^{1}	0.9044^{9}	0.8856^{1}
S2N	0.8452^{9}	0.9018^{2}	0.9012^{4}	0.9044^{10}	0.8803^{4}

(b) Eclipse 2

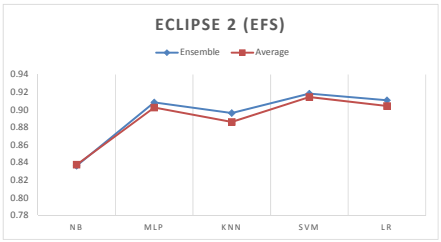
Ranker	NB	MLP	KNN	SVM	LR
Ensemble	0.8565^{2}	0.9055^{8}	0.9065^{4}	0.9147^{8}	0.9118^{1}
CS	0.8204^{11}	0.9085^{4}	0.9052^{7}	0.9175^{1}	0.9110^{2}
IG	0.8274^{10}	0.9103^{1}	0.9054^{6}	0.9169^{2}	0.9099^{6}
RF	0.8585^{1}	0.8923^{11}	0.8936^{11}	0.9032^{11}	0.9014^{11}
MI	0.8491^{5}	0.9059^{7}	0.9065^{3}	0.9164^{3}	0.9102^{5}
KS	0.8486^{6}	0.9095^{2}	0.9055^{5}	0.9162^{4}	0.9095^{9}
Dev	0.8420^{8}	0.9073^{5}	0.9043^{9}	0.9161^{5}	0.9105^{4}
GM	0.8519^{4}	0.9068^{6}	0.9047^{8}	0.9153^{6}	0.9097^{7}
ROC	0.8483^{7}	0.9090^{3}	0.9076^{2}	0.9139^{9}	0.9095^{8}
PRC	0.8543^{3}	0.9043^{9}	0.9079^{1}	0.9151^{7}	0.9108^{3}
S2N	0.8385^{9}	0.9030^{10}	0.9040^{10}	0.9103^{10}	0.9061^{10}

TABLE IV
ANOVA FOR THE ECLIPSE DATA SETS

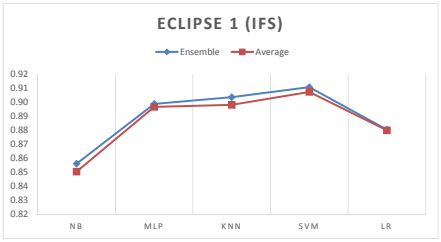
Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Scenario	0.1591	1	0.1591	133.27	0.000
Learner	3.6895	4	0.9224	772.49	0.000
Ranker	0.2963	10	0.0296	24.82	0.000
Error	7.8616	6584	0.0012		
Total	12.0065	6599			



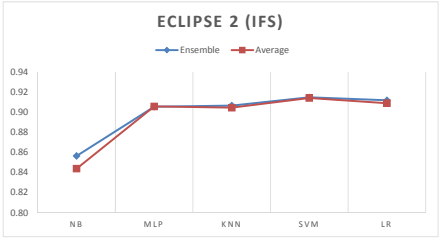
(a) Eclipse 1 (EFS)



(b) Eclipse 2 (EFS)

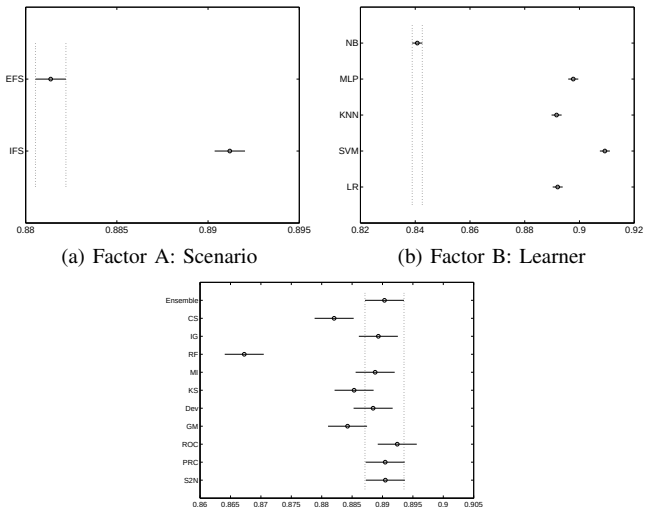


(c) Eclipse 1 (IFS)



(d) Eclipse 2 (IFS)

Fig. 2. Comparison between the ensemble and average of the ten individual filters in EFS and IFS scenarios



(a) Factor A: Scenario (b) Factor B: Learner

(c) Factor C: Ranker

Fig. 3. Multiple comparison for Eclipse 1 and 2 data sets

while the alternate hypothesis is that at least one pair of means is different. Note that the two groups of data sets were analyzed together. Table IV shows the ANOVA results. The p-value is less than the cutoff 0.05 for all factors, meaning that for each main factor the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other.

We further conducted a multiple comparison test on each main factor 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 A, B, and C. The figures display graphs with each group mean represented by a symbol ($\circ$) 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:

- Feature selection performed inside RUSBoost (IFS) resulted in better classification performance than when it is applied prior to RUSBoost (EFS).
- For the five learners, SVM presented the best performance, followed by MLP, then KNN and LR; NB performed the worst.
- Among the 11 feature ranking techniques, the ensemble method demonstrated competitive performance, ranking right behind the best performer, ROC, and had the same performance as PRC and S2N; IG, MI, and Dev were in the following group, then KS and GM, although the ensemble ranker did not show statistical difference between itself and those methods; CS and RF had the worst performance.

VI. Conclusion

In this study, we proposed feature selection combined with an ensemble learning (boosting) technique to overcome the high dimensionality and class imbalance problems that often affect software quality classification. Two different scenarios (IFS and EFS) of the technique that combines feature selection with boosting were investigated. IFS (internal feature selection) refers to when feature selection is applied inside the boosting process, while EFS (external feature selection) refers to when feature selection takes place prior to the boosting process. In this research, we were interested in investigating ten individual feature ranking techniques and the ensemble ranker based on the ten, and comparing them for the two scenarios. In the experiment, we applied these techniques to two groups of data sets from a real-world software system. We built classification models using five learners. The results demonstrate that the ensemble technique performed better than or similarly to the average of the ten base rankers, and more importantly, the ensemble ranker yielded more stable and robust performance than any other competing base rankers. In addition, the results show that performing feature selection inside of boosting generally performed better than using feature selection prior to boosting. Of the five learners, support vector machine performed the best, followed by multilayer perceptron, k nearest neighbors, and logistic regression; naïve Bayes had the worst prediction performance. Future work will involve conducting additional empirical studies with software measurement and defect data from other software projects.

References

[1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, May/June 2011.

[2] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, and R. Wald, "Threshold-based feature selection techniques for high-dimensional bioinformatics data," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 1, no. 1-2, pp. 47–61, June 2012.

[3] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, Jan. 2010.

[4] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the 13th International Conference on Machine Learning*, 1996, pp. 148–156.

[5] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, Hyderabad, India, 2010, pp. 4–13.

[6] L. Yu, Y. Han, and M. E. Berens, "Stable gene selection from microarray data via sample weighting," *IEEE/ACM Transactions On Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 262–272, Jan/Feb 2012.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[8] 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," *Information Systems Frontiers*, pp. 1–22, April 2013.

[9] P. Yang, W. Liu, B. B. Zhou, S. Chawla, and A. Y. Zomaya, "Ensemble-based wrapper methods for feature selection and class imbalance learning," in *Proceedings of the 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, Part I, Lecture Notes in Computer Science 7818*. Springer-Verlag, April 14-17 2013, pp. 544–555.

[10] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[11] L. Goh, Q. Song, and N. Kasabov, "A novel feature selection method to improve classification of gene expression data," in *Proceedings of the Second Conference on Asia-Pacific Bioinformatics*, Dunedin, New Zealand, 2004, pp. 161–166.

[12] J. S. Olsson and D. W. Oard, "Combining feature selectors for text classification," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, 2006, pp. 798–799.

[13] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice-Hall, 1998.

[14] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd ed. Cambridge University Press, 2000.

[15] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.