Software Defect Prediction for High-Dimensional and Class-Imbalanced Data

Kehan Gao
Eastern Connecticut State University
Willimantic, Connecticut 06226
gaok@easternct.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgo@fau.edu

Abstract—Software quality and reliability can be improved using various techniques during the software development process. One effective method is to utilize software metrics and defect data collected during the software development life cycle and build defect predictors using data mining techniques to estimate the quality of target program modules. Such a strategy allows practitioners to intelligently allocate project resources and focus more on the potentially problematic modules. Effectiveness of a defect predictor is influenced, among other factors, by the quality of input data. Two problems which often arise in the software measurement and defect data are high dimensionality and class imbalance. This paper presents an approach for using feature selection and data sampling together to deal with the problems. Three scenarios are considered: 1) feature selection based on sampled data, and modeling based on original data; 2) feature selection based on sampled data, and modeling based on sampled data; and 3) feature selection based on original data, and modeling based on sampled data. Several software measurement data sets, obtained from the PROMISE repository, are used in the case study. The empirical results demonstrate that classification models built in scenario 1) result in significantly better performance than the models built in the other two scenarios.

Index Terms—software quality classification, high dimensionality, class imbalance, feature selection, data sampling

I. INTRODUCTION

The success or failure of a software project depends on the product’s quality and reliability. Software practitioners collect software metrics and fault data during the software development process, and then analyze the data for defect prediction modeling. Typically, a software quality estimation model is trained using software metrics and defect data collected from prior development experiences of the organization, and then the trained model is applied to the project under development. This helps practitioners strategically allocate project resources, for example by assigning more inspection and testing to the potentially problematic modules.

The effectiveness of software quality estimation models is influenced, among other reasons, by two key quality of data factors: (1) the set of software metrics (predictors or independent attributes) used to build the models, and (2) the proportion of minority (i.e., fault-prone) instances in the software measurement data set. Related literature has shown that an overabundance of features (i.e., attributes) exists in various software project data repositories [1], [2]. In addition, studies also show that not all features make equally important contributions to the dependent variable. Selecting a subset of features that are most relevant to the class attribute is necessary and may result in better predictions [3]. In the context of the software quality classification problem (e.g., classifying program modules into fault-prone (fp) and not-fault-prone (nfp) groups), class imbalance (or skewed data) occurs frequently [4]. The class imbalance problem occurs when, for a given data set, instances (program modules) of one class are clearly outnumbered by the instances of the other class. For software measurement and defect data, the fp modules are often much fewer than the nfp modules. A classic learner trained on such an imbalanced data set is likely to have a large number of misclassifications of the minority class (e.g., fp modules) instances. This is an extremely severe problem in the software quality assurance domain, as the model would lead to missed opportunities to re-inspect and correct a poor quality module prior to system deployment. Data sampling is a proven method for alleviating the problems associated with class imbalance [4].

This study investigates a process that combines feature selection and data sampling to deal with the problems of high dimensionality and class imbalance that exist during software defect prediction. We used nine filter-based feature selection techniques, which come from three different families. Among the nine techniques, three of them are standard feature ranking methods, five of them are threshold-based feature selection techniques we recently proposed [5], and the last one is signal-to-noise, a rarely-used feature ranking method. We used only one data sampling technique, that is random undersampling.

The process of using feature selection and data sampling may lead to several different scenarios depending on whether feature selection takes place before or after data sampling and which data set, sampled or unsampled data, is used to build a classifier. Three scenarios are examined in this study:

- Scenario 0: data sampling takes place before feature selection is performed, and then a classifier is built using the features selected and unsampled (original) data.
- Scenario 1: data sampling takes place before feature selection is performed, and then a classifier is built using the features selected and sampled data.
- Scenario 2: data sampling takes place after feature selection is performed, and then a classifier is built using the features selected and sampled data.

Two more scenarios are also produced, where one technique (feature selection or data sampling) is used alone. However, we ignore these two options in this study, because all software data sets investigated exhibit both the class imbalance and high-dimensionality problems. To our knowledge, this paper is one of the very few studies which have considered both feature selection and data sampling together.

In order to compare the effectiveness of the three approaches (scenarios), we conducted a case study based on nine software measurement and defect data sets obtained from the PROMISE software project repository [6]. The key contributions of this paper are summarized as follows: (1) Using filter-based feature selection techniques combined with sampling to deal with the common problems of software data sets, i.e., high dimensionality and class imbalance. (2) Exploring three approaches (scenarios) when using feature selection and data sampling simultaneously. (3) Employing nine different feature ranking techniques from three different families to make the results easier to generalize.

The remainder of the paper is organized as follows. Section II describes related work. The nine filter-based feature selection techniques and the random undersampling method, as well as the
classifier and the associated performance metric used in this study are presented in Section III. A case study is described in Section IV. Finally, conclusions and future work are summarized in Section V.

II. RELATED WORK

Feature selection, as an effective method for handling high-dimensional data, has been extensively studied for a long time in the data mining and machine learning community. Generally, feature selection is divided into two categories, wrapper-based feature selection and filter-based feature selection. For wrapper-based techniques, the same classifier or inductive algorithm is used to both select the relevant features and execute the mining process [7]. Therefore, for a given data set, a wrapper-based technique may produce different feature subsets when using different learners. The potential problems of a wrapper-based technique lie in its high computational cost and a risk of overfitting to the model. In contrast, a filter-based technique is learner-independent. Once the data set is given, the filter-based technique will produce a feature subset (or feature subsets) that is (or are) correlated to the dependent attributes irrespective of which learner will be used after. In this study, the nine feature selection techniques used belong to the filter-based category.


Class imbalance, which appears in various domains [12], [13], is another significant problem in data mining. One effective method for alleviating the adverse effect of skewed class distribution is sampling, which will add or remove instances from a data set until it becomes more balanced with respect to its class distribution [14], [15]. In this study, we used random undersampling due to its simplicity and effectiveness [15]. We will leave other sampling techniques as our future work.

While considerable work has been done for feature selection and data sampling separately, limited research can be found on investigating them both together, particularly in the software engineering field. Chen et al. [11] have studied data row pruning (data sampling) and data column pruning (feature selection) in the context of software cost/effort estimation. However, the data sampling in their study was not specific for the class imbalance problem, and unlike this study, the classification models are meant for non-binary problems. Moreover, they focused only on the case in which data sampling is used prior to feature selection.

A recent work of our research team investigated both feature selection and data sampling in the domain of software quality engineering [16]. The paper presented four approaches (scenarios) which include the three scenarios studied in this paper and the scenario where feature selection is used alone. In that paper, six standard filter-based feature ranking techniques were adopted. The conclusion was that data sampling performed prior to feature selection resulted in significantly better performance than data sampling performed after feature selection. The present study is an extension of the earlier work, exploring new ground in three ways: (1) We used nine different feature selection techniques from three different families instead of only six commonly used (standard) filter-based feature ranking techniques, which makes our conclusion more generalized; (2) In addition to considering the different scenarios, this study also looked at the different feature ranking techniques and examined their impact on the classification performance; and (3) The conclusion of this study is different than the previous one, possibly as a result of the aforementioned changes. We try to keep all the experimental settings the same as we did before to ensure effectiveness of the comparisons between the two studies.

III. METHODOLOGY

A. Feature Selection Techniques

In this paper, we investigate nine filter-based feature ranking techniques from three different families, including three standard methods, five threshold-based feature selection techniques, and the signal-to-noise approach. Following is an overview of each ranker family.

1) Standard Filter-Based Feature Rankers: Feature ranking assigns a score to each feature according to a particular method (metric), allowing the selection of the best set of features. Due to the space limitation, we only present three filter-based feature ranking techniques [17]: chi-square (CS), information gain (IG), and ReliefF (RF). More details are shown as follows.

i. The chi-square (CS) statistic, denoted as $\chi^2$, is used to examine the distribution of the class as it relates to the values of the target feature. The null hypothesis is that there is no correlation. Given the null hypothesis, the $\chi^2$ statistic measures how far away the actual value is from the expected value:

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

where $r$ is the number of different values of the feature, $n_c$ is the number of classes ($n_c = 2$ in this work), and $O_{i,j}$ and $E_{i,j}$ are the observed and expected number of instances with value $i$ which are in class $j$, respectively. The larger the $\chi^2$ statistic, the more likely the feature is relevant to the class.

ii. Information gain (IG) is a measure based on the concept of entropy from information theory [17]. 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 to the impurity of the complete set of data. IG tends to prefer attributes with a larger number of possible values.

iii. Relief is an instance-based feature ranking technique introduced by Kira and Rendell [18]. ReliefF (RF) is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA tool [17].

2) Threshold-Based Feature Rankers: The family of threshold-based feature selection (TBFS) techniques was developed and implemented by our research group within WEKA. A full discussion of this approach (although limited to cover only the AUC performance metric) can be found in [5]. In TBFS, each independent attribute is paired individually with the class attribute, and that two-attribute data set is evaluated using a number of different performance metrics. This feature ranking framework includes normalizing the attribute values (so that they fall between 0 and 1) and treating those values as the posterior probabilities from which to calculate performance metrics. In fact, this allows the use of performance metrics to describe how well the feature correlated with the class. Note that no classifiers were
Built during the feature selection process. Five performance metrics used in TBFS are listed below.

i. Mutual Information (MI). Let \( c(x) \) denote the actual class of instance \( x \), and let \( \hat{c}(x) \) denote the predicted class based on the value of the attribute \( F^j \) and a given threshold \( t \). Then MI is defined as:

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

where

\[
p(c, i) = \frac{|\{x \mid \hat{c}(x) = \alpha \cap (c(x) = \beta)\}|}{|P| + |N|},
\]

\[
p(c) = \log \frac{p(c, i)}{p(c)}.
\]

\[\alpha, \beta \in \{P, N\}.\]

ii. Kolmogorov-Smirnov (KS) statistic measures the maximum difference between the cumulative distribution functions of examples in each class based on the normalized attribute \( F^j \).

\[
F_P(t) = \left| \frac{|\{x \mid (\hat{c}(x) \leq t) \cap (c(x) = P)\}|}{|\{x \mid (\hat{c}(x) \leq t) \cap (c(x) = N)\}|} \right|
\]

\[
F_N(t) = \left| \frac{|\{x \mid (\hat{c}(x) \leq t) \cap (c(x) = N)\}|}{|\{x \mid (\hat{c}(x) \leq t) \cap (c(x) = P)\}|} \right|
\]

KS is computed as

\[
KS = \max_{t \in [0,1]} |F_P(t) - F_N(t)|.
\]

With respect to KS, an attribute provides the best performance at a specific \( t \) value when the distance between the two distribution functions is maximized. The larger the KS value, the better the attribute is able to separate the two classes. The range of KS is between 0 and 1.

iii. Deviance (DV). For a given threshold \( t \), define \( v(x) = 1 \) if example \( x \) belongs to the positive class, otherwise, \( v(x) = 0 \).

Then DV is defined as:

\[
DV = \min_{t \in [0,1]} \left[ \sum_{x \in S_1} (v(x) - v(S^j))^2 + \sum_{x \in S_2} (v(x) - v(S^j))^2 \right]
\]

where \( v(S^j) = |S_1|^{-1} \sum_{x \in S_1} v(x) \). In other words, the equation measures the sum of the squared errors from the mean class given a partitioning of the space based on each possible threshold \( t \). The minimum value is preferred.

iv. Area Under the ROC Curve (AUC). The receiver operating characteristic [19], or ROC, curve graphs true positive rate on the \( y \)-axis versus the false positive rate on the \( x \)-axis. ROC curves are generated by varying the decision threshold \( t \) (between 0 and 1) used to transform the normalized attribute values into a predicted class. AUC is used to provide a single numerical metric for comparing the predictive power of each attribute. The range of AUC is between 0 and 1, and larger value is preferred.

v. Area Under the Precision-Recall Curve (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 \( (y\)-axis) and precision \( (x\)-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: Signal to noise (S2N) [20], in the context of the feature ranking problem in data mining, defines how well a feature discriminates two classes in a two class problem. The equation to calculate S2N is

\[
S2N = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}
\]

where \( \mu_P \) and \( \mu_N \) are mean values of a particular attribute for the samples from class \( P \) and class \( N \), and \( \sigma_P \) and \( \sigma_N \) are standard deviations of this attribute from the sample set for class \( P \) and class \( N \). Because of its discriminating power among the classes, S2N is highly efficient to properly order the features in terms of their relation to the output class. But this technique has not been used as often for feature selection.

B. Data Sampling Techniques

A number of data sampling techniques have been studied in the literature, including both majority undersampling and minority oversampling techniques [14], [15]. We consider random undersampling (RUS) as the data sampling technique in this study. Random undersampling is a simple, yet effective, data sampling technique that achieves more balance in a given data set by randomly removing instances from the majority (\( npf \)) class. The post-sampling class distribution is a parameter for any data sampling technique. In our study, the post-sampling distribution of the \( npf \) and \( fp \) instances is 65% and 35% respectively. Other settings such as 50:50 were also considered, but those results are not presented due to similarity of conclusions.

C. Classification Performance Metric

The effectiveness of each approach is assessed by evaluating the classification performance of the models subsequently trained and tested with that particular approach. In our experiments, we use AUC as the classification performance metric. Our selection of AUC is based on one of its characteristics, namely its invariance to a priori class probability distributions. AUC does not emphasize one class over the other as may be the case in some other performance metrics, thus it is not biased against the positive (\( fp \)) class. Given the imbalanced nature of our data sets (see in later section), AUC is an appropriate measure for comparing the classification performance of the learners. In fact, AUC serves as an aid to both feature ranking and final classification evaluation in this study.

D. Classifier

In this study, the software quality prediction models are built with support vector machine (SVM) [21]. This learner was selected because of its common use in the software engineering domain and data mining, and also because it does not have a built-in feature selection capability. SVM, also called SMO in WEKA, had two changes to the default parameters: the complexity constant \( c \) was set to ‘5.0’ and build Logistic Models was set to ‘true’. By default, a linear kernel was used.

IV. A Case Study

A. Data Sets

In our experiments, we use publicly available data, namely the Eclipse defect counts and complexity metrics data set obtained from the PROMISE data repository [6]. In particular, we use 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,
TABLE I

<table>
<thead>
<tr>
<th>Data#</th>
<th>Rel.</th>
<th>thd</th>
<th>#Attr.</th>
<th>#Inst.</th>
<th>#fp</th>
<th>%fp</th>
<th>#nfp</th>
<th>%nfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>10</td>
<td>209</td>
<td>377</td>
<td>23</td>
<td>6.1</td>
<td>354</td>
<td>93.9</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
<td>5</td>
<td>209</td>
<td>377</td>
<td>52</td>
<td>13.8</td>
<td>325</td>
<td>86.2</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>3</td>
<td>209</td>
<td>377</td>
<td>101</td>
<td>26.8</td>
<td>276</td>
<td>73.2</td>
</tr>
<tr>
<td>4</td>
<td>2.1</td>
<td>5</td>
<td>209</td>
<td>434</td>
<td>34</td>
<td>7.8</td>
<td>400</td>
<td>92.2</td>
</tr>
<tr>
<td>5</td>
<td>2.1</td>
<td>4</td>
<td>209</td>
<td>434</td>
<td>50</td>
<td>11.5</td>
<td>384</td>
<td>88.5</td>
</tr>
<tr>
<td>6</td>
<td>2.1</td>
<td>2</td>
<td>209</td>
<td>434</td>
<td>125</td>
<td>28.8</td>
<td>309</td>
<td>71.2</td>
</tr>
<tr>
<td>7</td>
<td>3.0</td>
<td>10</td>
<td>209</td>
<td>661</td>
<td>41</td>
<td>6.2</td>
<td>620</td>
<td>93.8</td>
</tr>
<tr>
<td>8</td>
<td>3.0</td>
<td>5</td>
<td>209</td>
<td>661</td>
<td>98</td>
<td>14.8</td>
<td>563</td>
<td>85.2</td>
</tr>
<tr>
<td>9</td>
<td>3.0</td>
<td>3</td>
<td>209</td>
<td>661</td>
<td>157</td>
<td>23.8</td>
<td>504</td>
<td>76.2</td>
</tr>
</tbody>
</table>

and 3.0 respectively. Each release as reported by Zimmermann et al. [22] contains the following information: the name of the package for which the metrics are collected (name), the number of defects reported six months prior to release (pre-release defects), the number of defects reported six months after release (post-release defects), a set of complexity metrics computed for classes or methods and aggregated in terms of average, maximum, and total (complexity metrics), and the abstract syntax tree of the package consisting of the node size, type, and frequency (structure of abstract syntax tree(s)).

For our study we transform the original data by: (1) removing all non-node size, type, and frequency (structure of abstract syntax tree(s)) metrics), and the abstract syntax tree of the package consisting of the aggregated in terms of average, maximum, and total (complexity metrics). This results in three groups. Each group contains three data sets, for releases 2.0 and 3.0 while we use Scenario 1. This results in three groups. Each group contains three data sets, one for each release. The reason why a different set of thresholds is 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. These data sets exhibit different distribution of class skew (i.e., the percentage of fp examples).

### B. Design

The primary objective of the experiments is to evaluate the effectiveness of feature selection techniques when combined with data sampling. Different scenarios may be produced depending on whether sampling is performed prior to or after feature selection and which data set, sampled or unsampled data, is used to build a classifier. The three different scenarios examined are described as follows:

- **Scenario 0**: Data sampling is performed before feature selection and the selected features are applied to the unsampled (original) data to form the training data set.
- **Scenario 1**: Data sampling is performed before feature selection and the selected features are applied to the sampled data to form the training data set.
- **Scenario 2**: Data sampling is performed after feature selection and the selected features are applied to the sampled data to form the training data set.

### C. Results & Analysis

The experiments were performed on the three groups of Eclipse data sets. Nine feature ranking techniques from three different families were used to rank the attributes according to their respective scores. Then, we selected \([\log_2 n]\) attributes that had the highest scores, where \(n\) is the number of the independent attributes in the original data set. In this study, \(n = 208\), so \([\log_2 208]\) = 8. We choose \([\log_2 n]\) attributes because 1) related literature does not provide guidance on the appropriate number of features to select; and 2) one of our recent empirical studies [23] showed that it was appropriate to use \([\log_2 n]\) features when using WEKA to build random forests learners for binary classification in general and imbalanced data sets in particular. Although we used a different learner here, a preliminary study showed that \([\log_2 n]\) is still appropriate for various learners.

After feature selection and data sampling were implemented in the different scenarios, we built SVM models using the training data sets with the selected attributes, and we used AUC to evaluate the performance of the classifications. All results are reported in Table II. In the experiments, ten runs of five-fold cross-validation were performed. The values in the table represent the average AUC for the classification models constructed over the ten runs of five-fold cross-validation. For each data set, the average performance (Avg. column) for each feature selection method across three different scenarios, and the average performance (Avg. row) of each scenario over nine feature selection techniques are also presented. The best feature selection technique (based on their average performance) for each data set is highlighted with **underline**, and the best scenario (based on their average performance) is highlighted with *bold*. The results demonstrate that the classification models have better performance when using AUC, PRC, IG and MI feature selection techniques and also that scenario 0 shows better performance than the other two scenarios.

We also conducted a two-way ANalysis Of VAriance (ANOVA) F test on the classification performance over the nine data sets to examine whether the performance difference (better/worse) is statistically significant or not. The two factors considered in the test are: Factor A, representing three scenarios, and Factor B, representing nine feature ranking techniques. The null hypothesis for the ANOVA test is that all the group population means are the same and the alternate hypothesis is that at least one pair of means is different. In addition, the interaction between Factor A and Factor B is also taken into account in the test. Table III shows the ANOVA results. The p-value is zero or close to zero for each main factor (Factor A and Factor B). This means that at least two scenarios performed significantly differently from each other and at least two feature selection techniques present significantly different performances. The p-value of the interaction term is greater than the typical cutoff 0.05, meaning that the interaction does not significantly affect classification performance. In other words, changing the value of Factor A will not significantly influence the value of Factor B, and vice versa.

We further carried out a multiple comparison test [24] on each main factor and their interaction with Tukey’s honestly significant difference criterion. Figure 1, including three subfigures, shows the multiple comparisons for Factor A, Factor B, and interaction A×B, respectively. 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. Matlab was used to construct the ANOVA models and perform the multiple comparisons presented in this work, and the assumptions for constructing ANOVA models were validated. From these figures, we can see the following points:

- Of the three scenarios, Scenario 0 performs significantly better than the other two scenarios, followed by Scenario 2, then Scenario 1.
- Among the nine feature selection methods, AUC performs
### TABLE II
Classification Performance of SVM in terms of AUC

<table>
<thead>
<tr>
<th>Data# Filter</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>Avg.</th>
<th>Data# Filter</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>Avg.</th>
<th>Data# Filter</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>0.8662</td>
<td>0.8567</td>
<td>0.8409</td>
<td>0.8546</td>
<td>CS</td>
<td>0.9174</td>
<td>0.9111</td>
<td>0.9034</td>
<td>0.9016</td>
<td>CS</td>
<td>0.9329</td>
<td>0.9248</td>
<td>0.9253</td>
<td>0.9264</td>
</tr>
<tr>
<td>IG</td>
<td>0.8665</td>
<td>0.8555</td>
<td>0.8587</td>
<td>0.8602</td>
<td>IG</td>
<td>0.9135</td>
<td>0.9135</td>
<td>0.9137</td>
<td>0.9157</td>
<td>IG</td>
<td>0.9325</td>
<td>0.9290</td>
<td>0.9320</td>
<td>0.9312</td>
</tr>
<tr>
<td>RF</td>
<td>0.8753</td>
<td>0.8602</td>
<td>0.8619</td>
<td>0.8658</td>
<td>RF</td>
<td>0.8484</td>
<td>0.8462</td>
<td>0.8463</td>
<td>0.8470</td>
<td>RF</td>
<td>0.8925</td>
<td>0.8914</td>
<td>0.8766</td>
<td>0.8868</td>
</tr>
<tr>
<td>MI</td>
<td>0.8564</td>
<td>0.8436</td>
<td>0.8675</td>
<td>0.8558</td>
<td>MI</td>
<td>0.9160</td>
<td>0.9128</td>
<td>0.9228</td>
<td>0.9179</td>
<td>MI</td>
<td>0.9277</td>
<td>0.9218</td>
<td>0.9204</td>
<td>0.9233</td>
</tr>
<tr>
<td>KS</td>
<td>0.8471</td>
<td>0.8347</td>
<td>0.8268</td>
<td>0.8362</td>
<td>4</td>
<td>KS</td>
<td>0.9178</td>
<td>0.9128</td>
<td>0.9223</td>
<td>0.9176</td>
<td>7</td>
<td>KS</td>
<td>0.9249</td>
<td>0.9204</td>
</tr>
<tr>
<td>DV</td>
<td>0.8656</td>
<td>0.8565</td>
<td>0.8613</td>
<td>0.8611</td>
<td>DV</td>
<td>0.9179</td>
<td>0.9106</td>
<td>0.9174</td>
<td>0.9153</td>
<td>DV</td>
<td>0.9232</td>
<td>0.9201</td>
<td>0.9307</td>
<td>0.9247</td>
</tr>
<tr>
<td>AUC</td>
<td>0.8817</td>
<td>0.8641</td>
<td>0.8883</td>
<td>0.8757</td>
<td>AUC</td>
<td>0.9197</td>
<td>0.9144</td>
<td>0.9236</td>
<td>0.9193</td>
<td>AUC</td>
<td>0.9338</td>
<td>0.9286</td>
<td>0.9322</td>
<td>0.9315</td>
</tr>
<tr>
<td>PRC</td>
<td>0.8712</td>
<td>0.8638</td>
<td>0.8624</td>
<td>0.8658</td>
<td>PRC</td>
<td>0.9132</td>
<td>0.9134</td>
<td>0.9134</td>
<td>0.9133</td>
<td>PRC</td>
<td>0.9300</td>
<td>0.9245</td>
<td>0.9315</td>
<td>0.9287</td>
</tr>
<tr>
<td>S2N</td>
<td>0.8757</td>
<td>0.8652</td>
<td>0.8673</td>
<td>0.8694</td>
<td>S2N</td>
<td>0.9037</td>
<td>0.8962</td>
<td>0.8979</td>
<td>0.8993</td>
<td>S2N</td>
<td>0.9161</td>
<td>0.9179</td>
<td>0.9214</td>
<td>0.9185</td>
</tr>
<tr>
<td>Avg</td>
<td>0.8908</td>
<td>0.8956</td>
<td>0.8937</td>
<td>0.8907</td>
<td>Avg</td>
<td>0.9013</td>
<td>0.8987</td>
<td>0.8967</td>
<td>0.8936</td>
<td>Avg</td>
<td>0.9255</td>
<td>0.9178</td>
<td>0.9240</td>
<td>0.9220</td>
</tr>
</tbody>
</table>

### TABLE III
Two-way ANOVA Table

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum Sq</th>
<th>d.f.</th>
<th>Mean Sq</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0117</td>
<td>2</td>
<td>0.0059</td>
<td>7.23</td>
<td>0.0007</td>
</tr>
<tr>
<td>B</td>
<td>0.2794</td>
<td>8</td>
<td>0.0349</td>
<td>43.05</td>
<td>0.0000</td>
</tr>
<tr>
<td>A × B</td>
<td>0.0152</td>
<td>16</td>
<td>0.0009</td>
<td>1.17</td>
<td>0.2864</td>
</tr>
<tr>
<td>Error</td>
<td>1.4991</td>
<td>2403</td>
<td>0.0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1.5553</td>
<td>2429</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Multiple Comparisons
performances of the SVM learner using the original data with 208 software metrics. Each value presented in the table is the average AUC over the ten runs of five-fold cross-validation outcomes. The results demonstrate that all three strategies (scenarios) perform better or significantly better than the case where no feature selection or sampling is used at all, except for the 9th data set when RF is used.

V. Conclusion

In the context of software defect prediction, two key problems often faced by software practitioners are the presence of excessive metrics in training data sets and a relatively small proportion of fault-prone modules to learn from.

In order to overcome the problems, we studied a process using feature selection and data sampling together to modify the training data to improve software defect prediction models. Three scenarios are investigated: (1) feature selection based on sampled data, and training data based on original data; (2) feature selection based on sampled data, and training data based on sampled data; and (3) feature selection based on original data, and training data based on sampled data. The main objective of this paper is to examine and compare all three scenarios and assess their effectiveness in the context of software quality prediction. We used nine different filter-based feature ranking techniques to select the software metrics and random undersampling to deal with class imbalance. The SVM classifier was employed to build classification models. The experiments were performed on nine software measurement data sets obtained from the PROMISE repository. The results show that sampling performed prior to feature selection and training data based on original data resulted in significantly better performance than the other two approaches (scenarios). In addition, among the nine filter-based feature ranking techniques, AUC performed better than other methods. Moreover, all three strategies performed significantly better than the case where no feature selection or sampling is used. The above result is of particular importance to a software quality analyst, since a useful model can be built using only selected software metrics. This provides a less cumbersome and more insightful model for analyzing the software quality trends of the target project, as compared to analyzing the model with respect to a large set of metrics.

Future work will include experiments using data sets from different software projects. Various sampling techniques, learners, and performance metrics will also be examined in future work.

References


