A Comparative Study of Different Strategies for Predicting Software Quality

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgoftaar@gmail.com

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

Amri Napolitano
Florida Atlantic University
Boca Raton, Florida 33431
amrifau@gmail.com

Abstract—Various methods have been developed for improving the quality of a software product, especially for high-assurance and mission-critical software systems. One commonly used approach is software quality modeling, in which software practitioners utilize software metrics and defect data collected during the software development process to build defect prediction models that will help to find poor-quality program modules. Those modules predicted to be fault-prone will receive more inspection and testing, thereby improving their quality. Efficacy of defect prediction models is influenced by relevance between software metrics and fault data. Usually not all software metrics in data repositories contribute equally to the occurrence of faults. Choosing the most important metrics (features) prior to the model training process is needed to improve the effectiveness of defect predictors. In this paper, we study 18 filter-based feature selection techniques and evaluate their effectiveness through a case study performed on 16 different software data sets. Among the 18 techniques, six of them are standard filter-based methods, while 11 of them are threshold-based feature selection (TBFS) techniques proposed by our research team recently. The last one is based on signal to noise ratio (S2N), which is a widely used concept in electrical and communication engineering, but which is rarely used in feature selection. The experimental results demonstrate that the TBFS techniques perform similarly to the standard techniques and the S2N technique shows significantly better performance than the other 17 approaches.

Index Terms—threshold-based feature selection, filter-based feature ranking techniques, software defect prediction, software metrics, software quality, signal to noise ratio.

I. INTRODUCTION

One effective method for improving the quality and reliability of a high-assurance software project/product is to detect and correct software defects (bugs) early enough during the software development process and prior to system deployment and operation, since for such high-assurance systems certain defects or failures can have disastrous consequences. Various techniques and approaches have been created for this purpose. Among them, software quality modeling is an effective and attractive approach. In software quality assurance, practitioners often use software metrics (attributes or features) gathered during the software development process and various data mining techniques to build classification models for predicting whether a given program module (instance or example) is in the fault-prone (fp) or not fault-prone (nfp) class [1]. Such a strategy allows practitioners to intelligently allocate project resources and focus more on the potentially problematic modules.

However, in the practice of software quality estimation, we find that not all collected software metrics are useful or make equally important contributions to classification results. Simply training a defect prediction model using the available set of software metrics without regard to the quality of the underlying software measurement data is not recommended. Selecting a subset of features that are most relevant to the class attribute is necessary and may result in better predictions.

In this paper, we investigate 18 filter-based feature ranking techniques to select subsets of features (metrics or predictors). Among the 18 techniques, six of them are commonly used, namely, chi-square (CS), information gain (IG), gain ratio (GR), symmetrical uncertainty (SU), and ReliefF (two types, RF and RFW) [2]. Eleven of them are threshold-based feature selection techniques (TBFS) proposed by our research team; a preliminary investigation of one of these 11 TBFS techniques has been reported in our recent work [3]. The remaining technique is based on signal to noise ratio (S2N), which is a widely used concept in electrical and communication engineering, but which has only been used in data mining research very recently.

Our proposed threshold-based feature selection (TBFS) techniques substantially extend the FAST algorithm proposed by Chen and Wasikowski [4]. FAST is based on the area under a ROC (Receiver Operating Characteristic) curve generated by moving the decision boundary of a single feature classifier with thresholds placed using an even-bin distribution. This means that they calculated a ROC curve by discretizing the distribution, while our technique is much more general than their work. Our technique does not require discretization, making it more precise and avoiding the determination of how wide the bins should be. Further, there are 11 different versions of TBFS which are based on 11 different classifier performance metrics for feature ranking. Another nice property of our technique is that it can be extended to incorporate additional metrics.

In order to evaluate the effectiveness of the 18 filter-based feature selection methods, we perform a case study on 16 software data sets. The case study data consists of software measurement and defect data from three real-world software projects, including four data sets from a very large high-assurance telecommunications software system (LLTS) [5], three data sets from NASA software project KC1 [6], and nine data sets from the Eclipse project [7].

Following feature selection, defect prediction models are constructed using the Naïve Bayes classifier with the training data consisting of the software metrics selected by the 18 different approaches. The empirical results demonstrate that our proposed TBFS techniques have similar performance to the six commonly used feature selection approaches. In addition, the S2N technique shows significantly better performance than the other 17 techniques (6 standard methods + 11 TBFS methods). Moreover, some techniques exhibit more stable performance than other techniques with respect to different data sets. One thing we would like to emphasize is that from a software engineering point of view, a practitioner appreciates working with a smaller set of metrics for defect prediction than analyzing a large number of metrics.

The remainder of the paper is organized as follows: Section II summarizes some key related work. Section III describes the 18 filter-based feature ranking techniques and other methodologies used in this study. Section IV presents an empirical case study, including
software measurement data, results and analysis. Section V draws the conclusion of the paper and provides some directions for future work.

II. RELATED WORK

One of the challenging problems in the data mining process is high dimensionality of data, which not only requires extensive computation during the learning process but also may deteriorate learning results. Feature selection (or attribute selection) is an effective method for handling high dimensional data. In the selection process, the most important and relevant features will be selected to build classifiers. Feature selection techniques are broadly categorized into two groups: wrapper-based and filter-based. The wrapper-based approach involves training a learner during the feature selection process, while a filter-based approach uses the intrinsic characteristics of the data for feature selection and does not rely on training a learner. The primary advantage of the filter-based approach over the wrapper-based approach lies in its faster computation.

Hall and Holmes [8] investigated six attribute selection techniques that produce ranked lists of attributes and applied them to several data sets from the UCI machine learning repository. Ilczuk et al. [9] investigated the importance of feature selection in judging the qualification of patients for cardiac pacemaker implantation. Forman [10] studied a few number of filter-based feature ranking techniques in the context of text mining.

Although feature selection has been widely applied in various data mining problems, its application in software quality and reliability engineering has been rather limited. Rodriguez et al. [11] applied feature selection to five software engineering data sets using three filter-based models and two wrapper-based models. The authors stated that the reduced data sets maintained the prediction capability of the original data sets while using fewer attributes. Chen et al. [12] have studied feature selection using wrappers in the context of software cost/effort estimation. In their study, serval COCOMO-I and COCOMO-II data sets were used. They concluded that the reduced data set could improve the estimation and recommended feature selection in cost modeling, particularly when dealing with very small data sets. Pizzi et al. [13] described a stochastic metric selection method to identify the software metric subset that is most effective at predicting software module complexity. This classification method was empirically evaluated and validated against three benchmark approaches. The main advantage of this research, claimed by the authors, is that a project manager or software architect could utilize the predictions to identify highly complex modules for review and possible revision. In a recent study [14], we investigated feature selection techniques using the filter-based method for software defect prediction. It was concluded that the performances of the classification models either improved or were not affected when only 15% of the original features were used.

III. METHODOLOGY

A. Standard Filter-Based Feature Ranking Techniques

The procedure of feature ranking is to score each feature according to a particular method, allowing the selection of the best features. The six commonly used filter-based feature ranking techniques examined in this work include [2]: chi-square (CS), information gain (IG), gain ratio (GR), two types of ReliefF (RF and RFW), and symmetrical uncertainty (SU).

The chi-square ($\chi^2$) statistic is used to evaluate the distribution of the class as it relates to the values of the target feature. The values of the features must be discretized into a number of intervals before performing the test. The $\chi^2$ statistic is defined as follows:

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

where $I$ is the number of different values (or intervals) of the feature, $n_i$ is the number of classes for a binary classification problem, $O_{i,j}$ and $E_{i,j}$ are the observed number and the expected number of instances corresponding to value (or interval) $i$ and class $j$. The larger the $\chi^2$ statistic, the more likely it is that the distribution of values and classes are dependent; that is, the feature is relevant to the class.

Information gain, gain ratio, and symmetrical uncertainty are measures commonly used in the field of information theory [2]. Information gain (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. If one attribute has a larger number of values, it will appear to gain more information than those with fewer values, even if it is actually no more informative. One strategy to solve this problem is to use the gain ratio (GR), which penalizes multiple-valued attributes. Symmetrical uncertainty (SU) is another way to overcome the problem of IG’s bias toward attributes with more values, doing so by dividing by the sum of the entropies of $X$ and $Y$.

Relief is an instance-based feature ranking technique introduced by Kira and Rendell [15]. It measures the importance of features by considering how much their values change when comparing a randomly chosen instance with its nearest hit (an instance from the same class) and its nearest miss (one from a different class). ReliefF is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA tool [2]. When the WeightByDistance (weight nearest neighbors by their distance) parameter is set as default (false), the algorithm is referred to as RF; when the parameter is set to ‘true’, the algorithm is referred to as RFW.

B. 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 that two-attribute data set is evaluated using different performance metrics. More specifically,

**Algorithm 1: Threshold-Based Feature Selection**

**input**:  
1. Data set $D$ with features $F^j$, $j = 1, \ldots, m$;  
2. Each instance $x \in D$ is assigned to one of two classes $c(x) \in \{P, N\}$ where $P = fp$ and $N = fn$;  
3. The value of attribute $F^j$ for instance $x$ is denoted $F^j(x)$;  
4. Metric $\omega \in \{FM, OR, PO, PR, GI, MI, KS, DEV, GM, AUC, PRC\}$.

**output**: Ranking $R = \{r^1, \ldots, r^m\}$ 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  
1. Normalize $F^j \Rightarrow \hat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;  
2. Calculate metric $\omega$ using attribute $\hat{F}^j$ and class attribute $\{c(x)|x \in D\}$, $\omega(\hat{F}^j)$: (The detailed formula of each metric $\omega$ is provided in Sections III-B1 through III-B11.)

Create attribute ranking $R$ using $\omega(\hat{F}^j) \forall j$.
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. Note that no classifiers were built during the feature selection process.

Analogous to the procedure for calculating rates in a classification setting with a posterior probability, the true positive (TPR), true negative (TNR), false positive (FPR), and false negative (FNR) rates can be calculated at each threshold \( t \in [0,1] \) relative to the normalized attribute \( \hat{F}^j \). Precision \( PRE(t) \) is defined as the number of positive examples with \( \hat{F}^j > t \) divided by the total number of examples with \( \hat{F}^j > t \). The feature rankers we propose utilize these five rates as described below. 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. Each of the 11 metrics are calculated for each attribute individually, and attributes with higher values for FM, GM, PR, PO, AUC, PRC, MI, KS and OR and lower values for GI and DEV are determined to better predict the class attribute. In this manner, the attributes can be ranked from most to least predictive based on each of the 11 metrics.

1) **F-Measure**: The F-measure (FM) is derived from recall (or true positive rate) and precision.

\[
FM = \max_{t \in [0,1]} \frac{2 \times TPR(t) \times PRE(t)}{TPR(t) + PRE(t)}.
\]

Recall and precision are calculated at each point along the normalized attribute range of 0 to 1. The maximum F-measure obtained by each attribute represents how strongly that particular attribute relates to the class, according to the F-measure.

2) **Odds Ratio**: The odds ratio (OR) is defined as:

\[
OR = \max_{t \in [0,1]} \frac{TPR(t)(1 - FPR(t))}{(1 - TPR(t))FPR(t)} = \max_{t \in [0,1]} \left( TPR(t) \left( TNR(t) - FNR(t) \right) \right)
\]

OR is the maximum value of the ratio of the product of correct to incorrect predictions.

3) **Power**: Power (PO) is defined as:

\[
PO = \max_{t \in [0,1]} \left( 1 - FPR(t) \right)^k - \left( 1 - TPR(t) \right)^k
\]

\[
= \max_{t \in [0,1]} \left( TNR(t) - FNR(t) \right)^k
\]

for some integer \( k \geq 1 \). Note that if \( k = 1 \), Power is equivalent to KS (described in Section III-B7). In this work, we use \( k = 5 \) as in Forman [10].

4) **Probability Ratio**: The probability ratio (PR) is defined as:

\[
PR = \max_{t \in [0,1]} \frac{TPR(t)}{FPR(t)}
\]

5) **Gini Index**: The Gini index (GI) was first introduced by Breiman et al. [16] within the CART algorithm. For a given threshold \( t \), let \( S_t = \{ x : \hat{F}^j(x) > t \} \) and \( \bar{S}_t = \{ x : \hat{F}^j(x) \leq t \} \). The Gini index is calculated as:

\[
GI = \min_{t \in [0,1]} \left[ 1 - \left( P^2(TP(t) \mid S_t) + P^2(FP(t) \mid \bar{S}_t) \right) \right]
\]

\[
+ \left[ 1 - \left( P^2(TN(t) \mid \bar{S}_t) + P^2(FN(t) \mid S_t) \right) \right]
\]

\[
= \min_{t \in [0,1]} \left[ 2PRE(t)(1 - PRE(t)) + 2NPV(t)(1 - NPV(t)) \right]
\]

where \( TP(t) \) is the number of true positives given threshold \( t \) (and similarly for \( TN(t), FP(t) \) and \( FN(t) \)). \( NPV \) or negative predictive value represents the percentage of examples predicted to be negative that are actually negative and is very similar to the precision — in fact, it is often thought of as the precision of instances predicted to be in the negative class. The Gini index for the attribute is then the minimum Gini index at all decision thresholds \( t \in [0,1] \).

6) **Mutual Information**: Let \( c(x) \in \{P, N\} \) denote the actual class of instance \( x \), and let \( \hat{c}(x) \) denote the predicted class based on the value of the attribute \( \hat{F}^j \) and a given threshold \( t \). The mutual information (MI) computes the 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:

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

where

\[
p(c, p) = \frac{\left| \{ x : \hat{c}(x) = c \} \cap \{ P \} \right|}{\left| P \right| + \left| N \right|}
\]

\[
p(c, p) = \frac{\left| \{ x : \hat{c}(x) = c \} \right|}{\left| P \right| + \left| N \right|}
\]

\[
p(p) = \frac{\left| \{ x : c(x) = p \} \right|}{\left| P \right| + \left| N \right|}
\]

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

7) **Kolmogorov-Smirnov Statistic**: The Kolmogorov-Smirnov Statistic (KS) measures the maximum difference between the cumulative distribution functions of examples in each class based on the normalized attribute \( \hat{F}^j \). The distribution function \( F_{c}(t) \) for a class \( c \) is estimated by the proportion of examples \( x \) from class \( c \) with \( \hat{F}^j(x) \leq t \), \( t \in [0,1] \). In a two class setting with \( c \in \{N, P\} \), KS is computed as:

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

The larger the KS value, the better the attribute is able to separate the two classes, and hence the more significant the attribute is. The range of KS is between 0 and 1.

8) **Deviance**: The deviance (DEV) is the minimum residual sum of squares based on a threshold \( t \). It measures the sum of the squared errors from the mean class given a partitioning of the space based on the threshold \( t \). As it represents to total error found in the partitioning, lower values are preferred.

9) **Geometric Mean**: The geometric mean (GM) is the square root of the product of the true positive rate and true negative rate. GM ranges from 0 to 1, and an attribute that is perfectly correlated to the class provides a value of 1. GM is a useful performance measure since it is inclined to maximize the true positive rate and the true negative rate while keeping them relatively balanced. GM is calculated at each value of the normalized attribute range, and the maximum value of GM is used as a measure of attribute strength.

10) **Area Under the ROC Curve**: 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. In other words, the true positive and false positive rates are calculated as the threshold for the normalized attribute varies from 0 to 1. The area under the ROC curve (AUC) is used to provide a
Signal to Noise Ratio Technique

Signal to noise ratio (S2N) [17] is a simple univariate ranking technique which defines how well a feature discriminates 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 equation to calculate S2N is

$$S2N = \frac{\mu_P - \mu_N}{\sigma_P + \sigma_N}$$

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.

Classification Algorithm

The software defect prediction models are built using the naïve Bayes (NB) [2] algorithm. This learner was selected for two key reasons: (1) it does not have a built-in feature selection capability, and (2) it is commonly used in both the software engineering and data mining domains. Prior research has shown that naïve Bayes classifiers often perform well, even on real-world data where the variables are related [18]. The WEKA data mining tool [2] is used to implement the classifier and the default parameter settings are adopted.

Performance Evaluation

The Area Under the ROC (Receiver Operating Characteristic) curve, abbreviated as AUC, is used for evaluating the defect prediction models in this study. The AUC metric is illustrated earlier in the paper. It is worthwhile to mention that AUC is commonly used to evaluate software defect prediction models [1]. An ROC curve illustrates the classifier’s performance across all decision thresholds, i.e., a value between 0 and 1 that theoretically separates the $fp$ and $nfp$ modules. The AUC values range from 0 to 1, where a perfect classifier provides an AUC value of 1 [2].

IV. CASE STUDY

A. Software Measurement Data

Experiments conducted in this study used software metrics and defect data collected from three real-world software projects, including a very large high-assurance telecommunications software system (denoted as LLTS) [5], NASA software project KC1 [6], and the Eclipse project [7].

LLTS consists of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. The dependent variable is the class of the program module. A module with one or more faults is considered $fp$, and $nfp$ otherwise. The LLTS software system consists of four successive releases labeled SP1, SP2, SP3, and SP4, where each release is characterized by the same number and type of software metrics, but has a different number of instances (program modules). The SP1, SP2, SP3, and SP4 data sets consist of 3649, 3981, 3541, and 3978 program modules, respectively.

The original NASA project KC1 [6] includes 145 instances, each containing 94 independent attributes. After removing 32 Halstead derived measures, we have 62 attributes left. We used three different thresholds to define defective instances, thereby obtaining three versions of the preprocessed KC1 data set. The thresholds are 20, 10, and 5, indicating instances with number of defects greater or equal to 20, 10, or 5 belong to the $fp$ class, or the $nfp$ class otherwise.

From the PROMISE data repository [7], we obtained the Eclipse defect counts and complexity metrics data set. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0, respectively. We chose three post-release defects thresholds ($thd$) to determine the defective instances for each release. Similar to the NASA KC1 data sets, a program module with $thd$ or more post-release defects is labeled $fp$, while those with fewer than $thd$ defects are labeled $nfp$. In our study, we use $thd \in \{10, 5, 3\}$ for release 2.0 and 3.0, while we use $thd \in \{5, 4, 2\}$ for release 2.1. All nine derived data sets contain 208 independent attributes. Releases 2.0, 2.1, and 3.0 contain 377, 434, and 661 instances, respectively. The sixteen data sets used in this work reflect software projects of different sizes with different proportions of $fp$ and $nfp$ modules. Table I lists the characteristics of the sixteen data sets utilized in this work.

B. Experiments

Before using a filter-based feature ranking technique, the practitioner must choose how many features to select. To our knowledge, no guidance is provided in related literature on the appropriate number of features to select. A recent study [19] recommended using $\lfloor \log_2 n \rfloor$ features ($n$ is the total number of the independent attributes) to build Random Forests learners for binary classification for imbalanced data sets. Moreover, a preliminary investigation showed that $\lfloor \log_2 n \rfloor$ is also appropriate for various learners. Consequently, we still choose $\lfloor \log_2 n \rfloor$ attributes that have the highest scores. That is, for the four LLTS data sets, $\lfloor \log_2 n \rfloor = 6$, where $n = 42$; for the three NASA KC1 data sets $\lfloor \log_2 n \rfloor = 6$, where $n = 62$; and for the nine Eclipse data sets, $\lfloor \log_2 n \rfloor = 8$, where $n = 208$.

1) Results of the Feature Selection Techniques: Following the feature selection algorithms, the NB classification models are constructed with data sets containing only the selected attributes. The defect prediction models are evaluated with respect to the AUC performance metric.

### Table I

<table>
<thead>
<tr>
<th>Data Set</th>
<th>#Attributes</th>
<th>#Instances</th>
<th>#fp</th>
<th>%fp</th>
<th>#nfp</th>
<th>%nfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP1</td>
<td>42</td>
<td>3649</td>
<td>229</td>
<td>6%</td>
<td>3420</td>
<td>94%</td>
</tr>
<tr>
<td>SP2</td>
<td>42</td>
<td>3981</td>
<td>189</td>
<td>5%</td>
<td>3792</td>
<td>95%</td>
</tr>
<tr>
<td>SP3</td>
<td>42</td>
<td>3541</td>
<td>47</td>
<td>1%</td>
<td>3494</td>
<td>99%</td>
</tr>
<tr>
<td>SP4</td>
<td>42</td>
<td>3978</td>
<td>92</td>
<td>2%</td>
<td>3886</td>
<td>98%</td>
</tr>
<tr>
<td>NASA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KC1-5</td>
<td>62</td>
<td>145</td>
<td>35</td>
<td>10%</td>
<td>110</td>
<td>75%</td>
</tr>
<tr>
<td>KC1-10</td>
<td>62</td>
<td>145</td>
<td>21</td>
<td>14%</td>
<td>124</td>
<td>86%</td>
</tr>
<tr>
<td>KC1-20</td>
<td>62</td>
<td>145</td>
<td>5</td>
<td>7%</td>
<td>138</td>
<td>93%</td>
</tr>
</tbody>
</table>

TABLE I

SOFTWARE DATA SET CHARACTERISTICS

<table>
<thead>
<tr>
<th>Data Set</th>
<th>#Attributes</th>
<th>#Instances</th>
<th>#fp</th>
<th>%fp</th>
<th>#nfp</th>
<th>%nfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1</td>
<td>42</td>
<td>3649</td>
<td>229</td>
<td>6%</td>
<td>3420</td>
<td>94%</td>
</tr>
<tr>
<td>SP2</td>
<td>42</td>
<td>3981</td>
<td>189</td>
<td>5%</td>
<td>3792</td>
<td>95%</td>
</tr>
<tr>
<td>SP3</td>
<td>42</td>
<td>3541</td>
<td>47</td>
<td>1%</td>
<td>3494</td>
<td>99%</td>
</tr>
<tr>
<td>SP4</td>
<td>42</td>
<td>3978</td>
<td>92</td>
<td>2%</td>
<td>3886</td>
<td>98%</td>
</tr>
<tr>
<td>NASA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KC1-5</td>
<td>62</td>
<td>145</td>
<td>35</td>
<td>10%</td>
<td>110</td>
<td>75%</td>
</tr>
<tr>
<td>KC1-10</td>
<td>62</td>
<td>145</td>
<td>21</td>
<td>14%</td>
<td>124</td>
<td>86%</td>
</tr>
<tr>
<td>KC1-20</td>
<td>62</td>
<td>145</td>
<td>5</td>
<td>7%</td>
<td>138</td>
<td>93%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eclipse</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E2.0-10</td>
<td>208</td>
<td>377</td>
<td>23</td>
<td>6%</td>
<td>354</td>
<td>94%</td>
</tr>
<tr>
<td>E2.0-5</td>
<td>208</td>
<td>377</td>
<td>52</td>
<td>14%</td>
<td>325</td>
<td>86%</td>
</tr>
<tr>
<td>E2.0-3</td>
<td>208</td>
<td>377</td>
<td>101</td>
<td>27%</td>
<td>276</td>
<td>73%</td>
</tr>
<tr>
<td>E2.1-5</td>
<td>208</td>
<td>434</td>
<td>34</td>
<td>8%</td>
<td>400</td>
<td>92%</td>
</tr>
<tr>
<td>Eclipse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2.1-4</td>
<td>208</td>
<td>434</td>
<td>50</td>
<td>12%</td>
<td>384</td>
<td>88%</td>
</tr>
<tr>
<td>E2.1-2</td>
<td>208</td>
<td>434</td>
<td>129</td>
<td>29%</td>
<td>309</td>
<td>71%</td>
</tr>
<tr>
<td>E3.0-10</td>
<td>208</td>
<td>661</td>
<td>41</td>
<td>6%</td>
<td>620</td>
<td>94%</td>
</tr>
<tr>
<td>E3.0-5</td>
<td>208</td>
<td>661</td>
<td>98</td>
<td>15%</td>
<td>563</td>
<td>85%</td>
</tr>
<tr>
<td>E3.0-3</td>
<td>208</td>
<td>661</td>
<td>157</td>
<td>24%</td>
<td>504</td>
<td>76%</td>
</tr>
</tbody>
</table>
The classifier performance results are presented in Table II. In our experiments, ten runs of five-fold cross-validation were performed for model training. The values presented in the tables represent the average AUC for every classification model constructed over the ten runs of five-fold cross-validation. All the results of the 18 feature selection techniques and over 16 different software data sets are reported. The best feature selection technique in terms of classification performance (AUC) for each data set (column) is highlighted in bold. We also summarize the average AUC (last column of the table) for each feature selection technique across the 16 data sets. The results demonstrate that S2N outperformed the other feature selection techniques for 10 out of 16 data sets and also on average. For the other six data sets the best feature selection techniques are scattered at GR(1), RF(2), RFW(2) and AUC(1), where the value in parenthesis represents the number of best cases.

We also performed a one-way Analysis Of Variance (ANOVA) F-test on the classification performance for each technique across all the data sets to examine the significance level of the performance differences. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The main factor of our ANOVA experiment is the 18 feature ranking techniques. The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different. Table III shows the ANOVA results. The $p$-value is less than the typical cutoff of 0.05, implying that for the main factor, the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other. We continued our statistical validation by performing a multiple comparison test on the main factor with Tukey’s Honestly Significant Difference (HSD) criterion. Note that for both ANOVA and multiple comparison tests, the significance level was set to 0.05.

The multiple comparison results, as shown in Figure 1, display a significant difference in performance with respect to different data sets such as MI, KS, S2N and SU, while other techniques, such as RF and RFW, showed more fluctuational performance with respect to different data sets.

2) Discussion on Selected Software Metrics: From a software engineering point of view, a discussion on which software metrics are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. Based on the multiple comparison results, we can conclude the following points:

- For the six standard filter-based feature selection techniques, GR performed worst, IG performed best and the other four methods sit in between.
- For the 11 proposed threshold-based feature selection techniques, PR and GI performed poorly, DEV, PO, FM, MI, AUC, and PRC performed relatively better than other techniques, and OR, KS, and GM performed moderately.
- Overall, the proposed threshold-based feature selection methods showed significantly better performance than the other 17 techniques.
- Moreover, some techniques demonstrated relatively stable performance with respect to different data sets such as MI, KS, S2N and SU, while other techniques, such as RF and RFW, showed more fluctuational performance with respect to different data sets.
attributes were spread over 15 different software metrics, and there were very few overlaps of the selected metrics between the top three methods, only three metrics \[11, 32, 36\] selected twice. Table V lists more detailed information of the 15 metrics.

Our recent work [20] has shown that classification models built on smaller subsets of attributes via the six commonly used filter-based feature selection techniques had similar or better performances than those built with a complete set of attributes. Thus, we did not present the results for full data sets in this paper.

### V. Conclusion

In the software quality modeling process, one problem often encountered by software practitioners is the presence of excessive metrics in a training data set. In this study, we presented 18 filter-based feature ranking techniques to choose the most important software metrics. Among the 18 techniques, six are standard filter-based techniques, while 11 are the threshold-based techniques we proposed, and the remaining one is the signal to noise ratio (S2N) technique rarely used in feature selection. We used the naïve Bayes classifier to build classification models with data sets containing only the selected attributes. The experiments were performed on 16 different data sets obtained from three types of software projects. The key conclusions include: (1) our proposed threshold-based feature selection techniques performed similarly to the standard feature selection techniques; (2) the S2N technique showed significantly better performance than the other 17 techniques; and (3) some techniques (such as MI, KS, S2N, and SU) demonstrated more stable performance than other techniques (such as RF and RFW) with respect to different data sets. Future work will include more case studies with software measurement data sets of other software systems. In addition, evaluation of the classification models using different learners such as multilayer perceptron, support vector machine, and k-nearest-neighbors will be examined.

### References


