An Empirical Study of Predictive Modeling Techniques of Software Quality

Taghi M. Khoshgoftaar[1], Kehan Gao[2], and Amri Napolitano[1]

[1] Florida Atlantic University, Boca Raton, FL 33431, USA
taghi@cse.fau.edu, amrifau@gmail.com
[2] Eastern Connecticut State University, Willimantic, CT 06226, USA
gaok@easternct.edu

Abstract. The primary goal of software quality engineering is to apply various techniques and processes to produce a high quality software product. One strategy is applying data mining techniques to software metrics and defect data collected during the software development process to identify the potential low-quality program modules. In this paper, we investigate the use of feature selection in the context of software quality estimation (also referred to as software defect prediction), where a classification model is used to predict program modules (instances) as fault-prone or not-fault-prone. Seven filter-based feature ranking techniques are examined. Among them, six are commonly used, and the other one, named *signal to noise ratio* (SNR), is rarely employed. The objective of the paper is to compare these seven techniques for various software data sets and assess their effectiveness for software quality modeling. A case study is performed on 16 software data sets and classification models are built with five different learners. Our experimental results are summarized based on statistical tests for significance. The main conclusion is that the SNR technique performs better than or similar to the best performer of the six commonly used techniques.

Keywords: filter-based feature ranking techniques, software defect prediction, software metrics, software quality.

1 Introduction

The quality of a software product is a key factor to that product's success or failure, and must be monitored and managed throughout the software development process. Extensive studies have been dedicated towards improving software quality. One frequently used method is software defect prediction, in which practitioners utilize software metrics (attributes or features) gathered during the software development life cycle, along with 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 [18]. The primary benefit of such a strategy is that it facilitates intelligent project resource allocation for the potentially problematic modules. For example, modules predicted to be fault-prone receive more inspection and testing, resulting in better quality.

However, in the practice of software defect prediction, we find that superfluous software metrics often exist in data repositories. Moreover, the collected software metrics may not be complete, consistent, or useful; some metrics may be redundant or irrelevant to classification results. Therefore, using all available software metrics to train a defect prediction model without considering the quality of the underlying software measurement data is often not the best strategy. Selecting a subset of features that are most important to the class attribute is needed prior to the model training process.

In this paper, we investigate seven filter-based feature ranking techniques to choose subsets of features (metrics). Among the seven techniques, six of them are commonly used methods, chi-square (CS), information gain (IG), gain ratio (GR), symmetrical uncertainty (SU), and ReliefF (two types, RF and RFW) [25]; the remaining technique is based on ***signal to noise ratio*** (SNR), which is a widely used concept in electrical and communication engineering but which has only started being used in data mining research very recently [17]. The idea of the filter-based feature ranking techniques is as follows: (1) each independent attribute is individually paired with the class attribute; (2) an intrinsic characteristic (score) of each attribute is evaluated; and (3) attributes are ordered according to the calculated scores. Note that no classifiers are built during the filter-based feature ranking process.

The main objective of this paper is to evaluate the effectiveness of seven filter-based feature selection methods in the context of software quality estimation. 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 telecommunications software system (LLTS) [12], nine data sets from the Eclipse project [28], and three data sets from NASA software project KC1 [16]. In the case study, feature selection is performed first, then defect prediction models are constructed using five different classifiers (naïve Bayes, multilayer perceptron, support vector machine, logistic regression, and K-nearest-neighbors) with the training data consisting of the software metrics selected by the seven different approaches. The empirical results demonstrate that the SNR technique has better performance than the other six commonly used feature selection approaches on average. Moreover, SNR exhibits more stable performance than some standard techniques such as RF and RFW with respect to different learners. From a software practice point of view, researchers and practitioners would like to work with a smaller set of metrics for defect prediction rather than analyze a large number of metrics.

The remainder of the paper is organized as follows: Section 2 describes some related work. Section 3 presents seven filter-based feature ranking techniques, five classifiers, and the performance metric used in this study. Section 4 describes the empirical case study, including software measurement data, results, analysis, and threats to empirical validity. Section 5 summarizes the conclusions of the paper and some directions for future work.

2 Related Work

Feature selection is the process of choosing some subset of the features and building a classifier based solely on those. It can remove as many unnecessary features as possible, leaving only those features which are useful in building a classifier. Feature selection has been extensively studied for many years in the data mining and machine learning community. A good overview on various aspects of the attribute selection problem was done by Guyon and Elisseeff [7]. They outlined the key techniques and approaches used in attribute selection including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. Liu and Yu [19] provided a comprehensive survey of feature selection algorithms and presented an integrated approach to intelligent feature selection.

Typically, feature selection techniques are divided into two types: *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.

In this paper, we examine seven filter-based feature ranking techniques, six of which are commonly used filter-based methods, while the other one, named signal to noise ratio (SNR), has only been applied in data mining research very recently to select relevant features in classification problems. Studies [5] showed that SNR's ability to properly rank the features is affected by the number of features present. If the number of features is too large (very high dimensional data) then ranking performance may be affected by the presence of noise in the data. But in most cases, SNR combined with other approaches shows improved performance. Next, we briefly discuss some of the recent techniques that involve SNR in feature selection.

A PCC (Pearson correlation coefficient) SNR hybrid method is shown to be very efficient in selecting genes in microarray expression data [5]. ANN (artificial neural network) based feature selection using SNR for classification of Doppler ultrasound signals has been studied by Güler and íbeyli [6]. They devised an ANN-SNR based classification method to classify Ophthalmic arterial Doppler signals and internal carotid arterial Doppler signals. Lakshmi et al. [17] suggest a MSNR (maximized signal to noise ratio) technique in the field of text classification. A novel GA-Taguchi based feature selection method applies this SNR based technique on seventeen different real world data sets and shows its superiority in selecting useful features [27]. In this paper, we study the standard SNR technique and apply it to software quality modeling. To our knowledge, no research has done this before.

We also notice that although feature selection has been widely applied in various data mining problems [3,7,8,10,22], its application in software quality and reliability engineering has been rather limited. Rodríguez et al. [21] applied feature selection to five software engineering data sets using three filter-based models and three wrapper-based models. The authors state that the reduced data sets maintained the prediction capability of the original data sets while using fewer attributes. Chen et al. [2] have studied feature selection using wrappers in the context of software cost/effort estimation. In recent studies [24,13], we investigated various feature selection techniques, including filter-based and wrapper-based methods. It was concluded that the performances of the classification models either improved or were not affected when over 85% of the features were eliminated from the original data sets.

3 Methodology

3.1 Standard Filter-Based Feature Ranking Techniques

Feature ranking assigns a score to each feature according to a particular method (metric), allowing the selection of the best set of features. The six commonly used filter-based feature ranking techniques considered in this work include [25]: chi-square (CS), information gain (IG), gain ratio (GR), two types of ReliefF (RF and RFW), and symmetrical uncertainty (SU).

The chi-squared - χ^2 (CS) test 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, i.e., each value is as likely to have instances in any one class as any other class. Given the null hypothesis, the χ^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 (in this work, $n_c = 2$), $O_{i,j}$ is the observed number of instances with value i which are in class j, and $E_{i,j}$ is the expected number of instances with value i and class j. The larger the χ^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 based on the concept of entropy from information theory [25]. 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]. ReliefF is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA[1] tool [25]. 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.

[1] WEKA (Waikato Environment for Knowledge Analysis) 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. In this study, all experiments and algorithms were implemented in the WEKA tool.

3.2 Signal to Noise Ratio

Signal to noise ratio (SNR) [5], in the context of classification problems in data mining, defines how well a feature discriminates two classes in a two-class problem. The equation to calculate SNR is

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

where μ_P and μ_N are mean values of a particular attribute for the samples from class P and class N, and σ_P and σ_N are standard deviations for this attribute from the sample set for class P and class N. Because of its discriminating power among the classes, SNR is highly efficient to properly rank the features in terms of its relation to the output class.

3.3 Classification Algorithms

The software defect prediction models are built using five different classification algorithms, including Naïve Bayes (NB) [25], Multilayer Perceptron (MLP) [9], Support Vector Machine (SVM) [23], Logistic Regression (LR) [25], and K-nearest-neighbors (KNN) [25]. These learners were selected for two key reasons: (1) they do not have a built-in feature selection capability, and (2) they are commonly used in both software engineering and data mining domains [11,18,20].

The WEKA data mining tool [25] 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’, and the kNN parameter was set to ‘5’.
– 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.

3.4 Performance Metric

The Area Under the ROC (Receiver Operating Characteristic) curve, abbreviated as AUC, is used for evaluating the defect prediction models in this study. If the *fp* modules are considered positive and the *nfp* modules are considered negative, then the ROC curve plots the true positive rates versus the false positive rates. An ROC curve illustrates the classifier's performance across all decision thresholds, i.e., a number between 0 and 1 that theoretically separates the *fp* and *nfp* modules. This is in contrast to most defect prediction studies that use performance metrics based on the default decision threshold value of 0.5. The AUC values range from 0 to 1, where a perfect classifier provides an AUC value of 1 [11,25].

Table 1. Software Data Set Characteristics

	Data	#Attri.	#Inst.	#fp	%fp	#nfp	%nfp
LLTS	SP1	42	3649	229	6%	3420	94%
	SP2	42	3981	189	5%	3792	95%
	SP3	42	3541	47	1%	3494	99%
	SP4	42	3978	92	2%	3886	98%
Eclipse	E2.0-10	208	377	23	6%	354	94%
	E2.0-5	208	377	52	14%	325	86%
	E2.0-3	208	377	101	27%	276	73%
	E2.1-5	208	434	34	8%	400	92%
	E2.1-4	208	434	50	12%	384	88%
	E2.1-2	208	434	125	29%	309	71%
	E3.0-10	208	661	41	6%	620	94%
	E3.0-5	208	661	98	15%	563	85%
	E3.0-3	208	661	157	24%	504	76%
NASA	KC1-5	62	145	36	25%	109	75%
	KC1-10	62	145	21	14%	124	86%
	KC1-20	62	145	10	7%	135	93%

4 Case Study

4.1 Software Measurement Data

Experiments conducted in this study used software metrics and defect data collected from real-world software projects, including a very large telecommunications software system (denoted as LLTS) [12], the Eclipse project [28], and NASA software project KC1 [16].

The software measurement data sets of LLTS consist of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. More details about these software metrics can be seen in [12]. 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.

From the PROMISE data repository [28], 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. A program module with thd or more post-release defects is labeled as *fp*, while those with fewer than thd defects are labeled as *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.

We obtain the class-level data set for the KC1 data set also from the PROMISE repository. This is a publicly available data set that represents one of five NASA projects [16]. The original data contains a set of measures per module, including the number of defects reported for the module. It included 145 instances, each containing 94 independent attributes plus a defect attribute. After removing 32 Halstead derived measures, we have 62 independent attributes left. We used three different thresholds to define defective instances, thereby obtaining three structures 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. The three data sets are named KC1-20, KC1-10, and KC1-5, respectively.

The 16 data sets used in this work reflect software projects of different sizes with different proportions of *fp* and *nfp* modules. Table 1 lists the characteristics of the 16 data sets utilized in this work.

4.2 Results of the Feature Selection Techniques

When using a filter-based feature ranking technique, the number of features (software metrics) that will be selected for modeling must be given in advance for the technique. In this study, we choose $\lceil \log_2 n \rceil$ features that have the highest scores, where n is the total number of the independent features. The reasons we adopt such a strategy include: (1) to our knowledge, related literature does not provide guidance on the appropriate number of features to select in such ranking techniques; (2) a recent study [14] recommended using $\lceil \log_2 n \rceil$ features to build Random Forests learners for binary classification for imbalanced data sets; and (3) a preliminary investigation showed that $\lceil \log_2 n \rceil$ is also appropriate for various learners. Consequently, a feature subset with size of $\lceil \log_2 n \rceil$ is used. That is, for the four LLTS data sets, we select $\lceil \log_2 42 \rceil = 6$ features; for the nine Eclipse data sets, we select $\lceil \log_2 208 \rceil = 8$ features; and for the three NASA KC1 data sets, we select $\lceil \log_2 62 \rceil = 6$ features.

Following the feature selection algorithms, the five different types of classification models are constructed with data sets containing only the selected attributes. The defect prediction models are evaluated in terms of the AUC performance metric, as stated earlier. Due to paper size limitations, we could not present each individual classifier's performance. We only present the results of the NB and LR learners individually and the average performance over five learners, as shown in Tables 2, 3, and 4 respectively. However, the discussions and summaries are made based on the facts observed over all five different classifiers.

In the 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 seven feature selection techniques and over 16 different software data sets are reported. The best feature selection technique in terms of their AUCs for each data set (row) is highlighted in **bold**. We also summarize the average performance (last row of the table) for each feature selection technique across the 16 data sets.

The results demonstrate that SNR outperformed the other feature selection techniques on average when the five classifiers are applied to the selected subset of features, since the average AUC of SNR over the five learners is 0.8380 (last row of Table 4), which is the highest score among the seven techniques. For the six commonly used techniques, IG performed best on average; GR, RF, and RFW performed more poorly than the other three techniques (IG, CS and SU). Of course, once the subset of features is set, the classification performance is determined by the learners we select. For instance, the feature subset selected by SNR demonstrated better performance than the

Table 2. Classification Performance in terms of AUC for **NB Classifier**

Data Set	CS	GR	IG	RF	RFW	SU	SNR
SP1	0.7846	0.7346	0.7831	0.7879	0.7882	0.7865	**0.7995**
SP2	0.8108	0.7613	0.8081	0.8053	0.8081	0.7729	**0.8142**
SP3	0.8184	0.7808	0.8118	**0.8305**	0.8190	0.7882	0.8067
SP4	0.7696	0.7519	0.7795	0.7731	0.7735	0.7592	**0.8130**
E2.0-10	0.7904	0.8074	0.8070	0.8455	0.8107	0.8158	**0.8623**
E2.0-5	0.8421	0.8078	0.8562	0.8617	0.8607	0.8464	**0.8767**
E2.0-3	0.7963	0.7458	0.8002	0.7857	**0.8107**	0.7940	0.8003
E2.1-5	0.8419	0.7919	0.8547	0.8022	0.8188	0.8269	**0.8681**
E2.1-4	0.8226	0.7917	0.8281	0.7688	0.7302	0.8170	**0.8560**
E2.1-2	0.7536	0.7614	0.7542	0.7993	0.7966	0.7551	**0.8063**
E3.0-10	0.8742	0.8463	0.8963	0.8044	0.8101	0.8540	**0.9175**
E3.0-5	0.8866	0.8785	0.8851	0.8481	0.8732	0.8817	**0.9099**
E3.0-3	0.8130	0.7974	0.8122	0.7789	0.8024	0.8072	**0.8759**
KC1-5	0.7484	0.7489	0.7438	**0.7990**	0.7832	0.7468	0.7847
KC1-10	0.7513	**0.7729**	0.7546	0.7585	0.7639	0.7719	0.7508
KC1-20	0.8525	0.8669	0.8569	0.8296	**0.8987**	0.8532	0.8646
Average	0.8098	0.7903	0.8145	0.8049	0.8093	0.8048	**0.8379**

Table 3. Classification Performance in terms of AUC for **LR Classifier**

Data Set	CS	GR	IG	RF	RFW	SU	SNR
SP1	0.8021	0.7688	0.8014	0.8103	0.8091	0.7993	**0.8176**
SP2	0.8230	0.7935	0.8176	0.8221	0.8233	0.7909	**0.8279**
SP3	0.8354	0.7805	0.8361	0.8354	**0.8387**	0.8040	0.8336
SP4	0.8153	0.7816	0.8216	0.8118	0.8142	0.7802	**0.8232**
E2.0-10	0.8067	0.8173	0.8367	0.8493	0.8184	0.8214	**0.8595**
E2.0-5	0.8898	0.8695	0.8999	0.8907	0.8923	0.8895	**0.9029**
E2.0-3	**0.8665**	0.8140	0.8658	0.8345	0.8468	0.8600	0.8611
E2.1-5	0.8729	0.8394	0.8765	0.8728	0.8824	0.8655	**0.8887**
E2.1-4	0.8673	0.8587	0.8652	0.8252	0.7582	0.8706	**0.8768**
E2.1-2	0.8852	0.8774	0.8839	0.8607	0.8726	0.8829	**0.8868**
E3.0-10	0.9090	0.8808	0.9015	0.8672	0.8604	0.8850	**0.9152**
E3.0-5	**0.9424**	0.9334	0.9420	0.8974	0.9083	0.9360	0.9418
E3.0-3	0.9096	0.9019	0.9098	0.8267	0.8415	0.9066	**0.9204**
KC1-5	0.7774	0.7669	0.7711	**0.8134**	0.7946	0.7648	0.7990
KC1-10	0.7613	0.7649	**0.7791**	0.7393	0.7072	0.7734	0.7701
KC1-20	0.8202	0.8293	0.8160	0.7433	**0.8420**	0.8336	0.8252
Average	0.8490	0.8299	0.8515	0.8312	0.8319	0.8415	**0.8594**

feature subsets selected by the other approaches for 11 out of 16 cases when NB was applied (see Table 2); 10 out of 16 cases when LR was used (see Table 3); seven out of 16 cases for MLP; six out of 16 for KNN; and four out of 16 for SVM. But on average, SNR performed better than other techniques for eight out of 16 cases (see Table 4).

We also performed a one-way ANalysis Of VAriance (ANOVA) F-test [1] on the classification performance for each technique across all the data sets to examine the significance level of the performance differences. The ANOVA tests were performed on the five classifiers individually. Once again, due to space limitation, we only present the test results for NB, LR, and the average of all five classifiers. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The main factor of our ANOVA experiment is the seven 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 4. Classification Performance in terms of AUC for **Five Classifiers**

Data Set	CS	GR	IG	RF	RFW	SU	SNR
SP1	0.7556	0.7236	0.7579	0.7627	0.7557	0.7593	**0.7678**
SP2	0.7863	0.7437	0.7741	0.7594	0.7619	0.7570	**0.7906**
SP3	0.7800	0.7379	0.7792	0.7818	**0.7839**	0.7593	0.7752
SP4	0.7641	0.7180	0.7722	0.7469	0.7512	0.7346	**0.7781**
E2.0-10	0.8172	0.8092	0.8404	0.8499	0.8178	0.8188	**0.8693**
E2.0-5	0.8758	0.8440	0.8850	0.8699	0.8674	0.8754	**0.8905**
E2.0-3	0.8333	0.7850	**0.8358**	0.8023	0.8158	0.8322	0.8300
E2.1-5	0.8767	0.8252	**0.8882**	0.8162	0.8299	0.8695	0.8800
E2.1-4	0.8664	0.8310	**0.8680**	0.7875	0.7207	0.8636	0.8654
E2.1-2	0.8512	0.8460	0.8500	0.8243	0.8328	0.8482	**0.8598**
E3.0-10	0.9018	0.8644	**0.9128**	0.8435	0.8304	0.8744	0.9065
E3.0-5	0.9228	0.9144	0.9226	0.8633	0.8760	0.9175	**0.9247**
E3.0-3	0.8822	0.8693	0.8812	0.7962	0.8134	0.8772	**0.8964**
KC1-5	0.7774	0.7589	0.7676	**0.8049**	0.7895	0.7634	0.7837
KC1-10	0.7714	0.7731	0.7760	0.7358	0.7084	**0.7803**	0.7641
KC1-20	0.8288	0.8418	0.8353	0.7989	**0.8664**	0.8355	0.8261
Average	0.8307	0.8053	0.8341	0.8027	0.8013	0.8229	**0.8380**

Table 5. One-way ANOVA

(a) **NB**

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Techniques	0.1982	6	0.0330	16.61	0
Error	2.2129	1113	0.0020		
Total	2.4111	1119			

(b) **LR**

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Techniques	0.1289	6	0.0215	7.59	0
Error	3.1523	1113	0.0028		
Total	3.2812	1119			

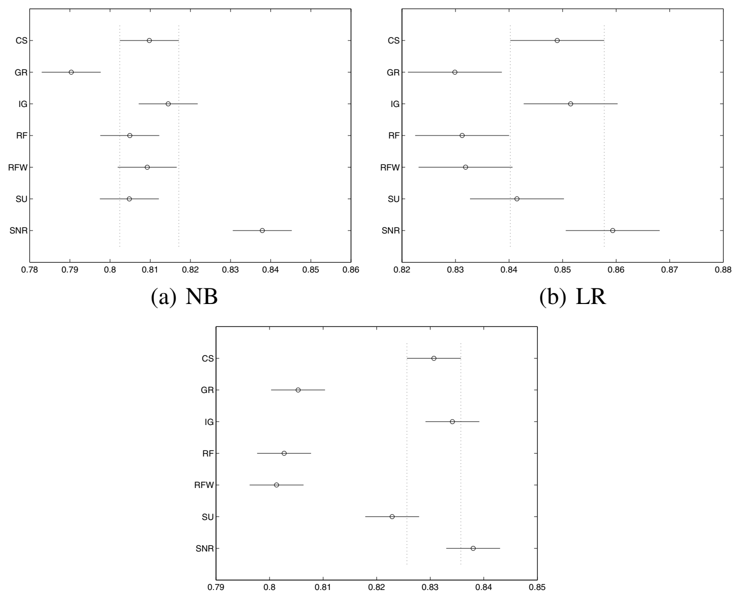
(c) **Five Classifiers**

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Techniques	1.2057	6	0.2010	43.11	0
Error	26.0682	5593	0.0047		
Total	27.2739	5599			

Table 5 shows the ANOVA results. It includes three subtables, each representing the result for each individual case (NB, LR, and five classifiers). All the p-values are less than the typical cutoff 0.05, indicating that for the main factor (Techniques), 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 [1]. Note that for both ANOVA and multiple comparison tests, the significance level α was set to 0.05.

The multiple comparison results are shown in Figure 1, displaying graphs with each group mean represented by a symbol ($\circ$) and the 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 perform the ANOVA and multiple comparisons presented in this work. Based on the multiple comparison results, we can conclude the following points:

– Among the six commonly used filter-based feature selection techniques, IG performed best. This is true irrespective of what classifier is used to build classification models. CS and SU performed averagely and GR, RF, and RFW performed poorly on average.



(a) NB (b) LR

(c) Five Classifiers

Fig. 1. Multiple Comparison

- SNR performed better than or similar to the best performer of the six commonly used techniques (i.e., IG) across 16 data sets on average.
- Some techniques demonstrate relatively stable performance with respect to different learners such as SNR and IG, while other techniques, such as RF and RFW, show more fluctuational performance with respect to different learners.
- Overall, IG and SNR demonstrated better and more stable performance than other approaches, and therefore are recommended by this study.

4.3 Discussion on Selected Software Metrics

From a software engineering point of view, a discussion on which software metrics were selected is warranted. Due to paper size limitations, we only present the results on LLTS SP1 as shown in Table 6. The table presents the selected subsets of software metrics for the seven techniques. For example, for SP1, the CS technique produces a subset of software metrics $\{5, 11, 15, 24, 33, 41\}$, where the values represent the software metric ID numbers. The 42 software metrics are labeled with an ID number ranging from 1 to 42. From the table, we can see that 27 metrics were selected by at least one feature selection technique and 15 metrics have never been selected by any techniques such as metric 3, 4, 10, and so on. The second column of the table indicates the number of times the metric is selected. All the detailed information related to the 42 software metrics are listed in Table 7.

Our recent work [4] has shown that classification models built on smaller subsets of features via the six commonly used filter-based feature selection techniques had similar

Table 6. Software Metrics Selected for SP1

Metric	Total #	Filter-Based Techniques						
ID	selected	CS	GR	IG	RF	RFW	SU	SNR
1	2		x				x	
2	1		x					
5	2	x					x	
6	1				x			
7	1					x		
8	1		x					
9	1				x			
11	2	x			x			
12	1					x		
15	3	x				x	x	
17	1			x				
19	2		x					x
20	1			x				
21	1							x
24	2	x				x		
27	2				x			x
28	1		x					
29	2			x			x	
30	2				x		x	
31	1							x
32	1							x
33	2	x		x				
34	2			x		x		
35	2					x	x	
36	3		x		x			x
41	1	x						
42	1			x				

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.

4.4 Threats to Validity

A typical software development project is very human intensive, which can affect many aspects of the development process including software quality and defect occurrence. Consequently, software engineering research that utilizes controlled experiments for evaluating the usefulness of empirical models is not practical. The case study presented in this paper is an empirical software engineering effort, for which the software engineering community demands that its subject have the following characteristics [26]: (1) developed by a group, and not by an individual; (2) be a large, industry-sized project, and not a toy problem; (3) developed by professionals, and not by students; and (4) developed in an industry/government organization setting, and not in a laboratory.

The software systems that are used in our case study were developed by professionals in large software development organizations using established software development processes and management practices. The software was developed to address real-world problems. We note that our case studies fulfill all of the above criteria specified by the software engineering community. The rest of this section discusses threats to external validity and threats to internal validity.

Threats to external validity are conditions that limit generalization of case study results. The analysis and conclusion presented in this article are based upon the metrics and defect data obtained from 16 data sets of three software projects. The same analysis

Table 7. Software Metrics for LLTS Data Sets

Metric ID	Symbol	Description
1	DES_PR	Number of problems found by designers during development of the current release.
2	BETA_PR	Number of problems found during beta testing of the current release.
3	DES_FIX	Number of problems fixed that were found by designers in the prior release.
4	BETA_FIX	Number of problems fixed that were found by beta testing in the prior release.
5	CUST_FIX	Number of problems fixed that were found by customers in the prior release.
6	REQ_UPD	Number of changes to the code due to new requirements.
7	TOT_UPD	Total number of changes to the code for any reason.
8	REQ	Number of distinct requirements that caused changes to the module.
9	SRC_GRO	Net increase in lines of code.
10	SRC_MOD	Net new and changed lines of code.
11	UNQ_DES	Number of different designers making changes.
12	VLO_UPD	Number of updates to this module by designers who had 10 or less total updates in entire company career.
13	LO_UPD	Number of updates to this module by designers who had between 11 and 20 total updates in entire company career.
14	UPD_CAR	Number of updates that designers had in their company careers.
15	USAGE	Deployment percentage of the module.
16	CALUNQ	Number of distinct procedure calls to others.
17	CAL2	Number of second and following calls to others. CAL2 = CAL - CALUNQ where CAL is the total number of calls.
18	CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs.
19	CNDSPNMX	Maximum span of branches of conditional arcs.
20	CTRNSTMX	Maximum control structure nesting.
21	FILINCUQ	Number of distinct include files.
22	KNT	Number of knots. A “knot” in a control flow graph is where arcs cross due to a violation of structured programming principles.
23	LOC	Number of lines of code.
24	CNDNOT	Number of arcs that are not conditional arcs.
25	IFTH	Number of non-loop conditional arcs (i.e., if-then constructs).
26	LOP	Number of loop constructs.
27	NDSENT	Number of entry nodes.
28	NDSEXT	Number of exit nodes.
29	NDSPND	Number of pending nodes (i.e., dead code segments).
30	NDSINT	Number of internal nodes (i.e., not an entry, exit, or pending node).
31	LGPATH	Base 2 logarithm of the number of independent paths.
32	STMCTL	Number of control statements.
33	STMDEC	Number of declarative statements.
34	STMEXE	Number of executable statements.
35	VARGLBUS	Number of global variables used.
36	VARSPNSM	Total span of variables.
37	VARSPNMX	Maximum span of variables.
38	VARUSDUQ	Number of distinct variables used.
39	VARUSD2	Number of second and following uses of variables. VARUSD2 = VARUSD - VARUSDUQ where VARUSD is the total number of variable uses.
40	RESCPU	Execution time (microseconds) of an average transaction on a system serving consumers.
41	BUSCPU	Execution time (microseconds) of an average transaction on a system serving businesses.
42	TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

for another software system may provide different results which is a likely threat in all empirical software engineering research. However, we place our emphasis on the process of comparing the different feature selection techniques considered in this study. Our comparative analysis can easily be applied to another software system. Moreover, since all our final conclusions are based on ten runs of five-fold cross-validation and statistical tests for significance, our findings are grounded in using sound methods.

Threats to internal validity are unaccounted for influences on the experiments that may affect case study results. Poor fault proneness estimates can be caused by a wide variety of factors, including measurement errors while collecting and recording software metrics, modeling errors due to the unskilled use of software applications, errors in model-selection during the modeling process, and the presence of outliers and noise in the training data set. Measurement errors are inherent to the data collection effort. In our comparative study, a common model-building and model-evaluation approach is used for all feature selection techniques and classifiers considered. Moreover, the experiments and statistical analysis were performed by only one skilled person in order to keep modeling errors to a minimum.

5 Conclusion

In the software quality modeling process, many practitioners often ignore a fact that excessive metrics exist in data repositories. They directly use the available set of software metrics to build classification models without regard to the quality of the underlying software measurement data, leading to inferior prediction accuracy and extension of training time. Selecting software metrics that are important for defect prediction is needed and critical before the model training process.

In this study, we investigated seven filter-based feature ranking techniques to select the most important software metrics. Among the seven techniques, six of them are standard filter-based techniques that are commonly used, while the remaining one is the signal to noise ratio (SNR) technique rarely used in feature selection. The main purpose of this paper is to examine and compare all seven filter-based feature selection techniques and evaluate their effectiveness in the context of software defect prediction. The experiments were conducted on 16 different data sets obtained from three different software projects. In order to alleviate the problem of potentially biased results generated by a specific classifier, we used five different learners to build classification models with data sets containing only the selected attributes. Moreover, ten runs of five-fold cross-validation were adopted to make the conclusions more persuasive.

The key conclusions are summarized as follows:

- For the six standard filter-based feature selection techniques, IG performed better than the other five techniques on average across all data sets. This is true no matter what classifier is used to build classification models. CS and SU performed slightly worse than IG, but better than GR, RF, and RFW.
- The SNR technique showed better or similar performance to the best performer of the six commonly used techniques, i.e., IG.
- Some techniques (such as SNR and IG) demonstrated more stable performance than other techniques (RF and RFW) with respect to different learners.
- IG and SNR exhibit overall better and more stable performance than other approaches, and therefore are recommended by this study.
- Selecting fewer software metrics for defect prediction is very important to the software quality assurance team, for instance, in our case study, working with six metrics is much easier and more practical than dealing with 42 metrics.

Future work will include more case studies with software measurement data sets of other software systems. In addition, as very few research works utilize SNR to select features, this concept presents a significant research area for computer scientists. More research works that use SNR to rank features are expected.

References

1. Berenson, M.L., Goldstein, M., Levine, D.: Intermediate Statistical Methods and Applications: A Computer Package Approach, 2nd edn. Prentice-Hall (1983)
2. Chen, Z., Menzies, T., Port, D., Boehm, B.: Finding the right data for software cost modeling. IEEE Software 22(6), 38–46 (2005)
3. Forman, G.: An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, 1289–1305 (2003)
4. Gao, K., Khoshgoftaar, T.M., Wang, H., Seliya, N.: Choosing software metrics for defect prediction: An investigation on feature selection techniques. Software: Practice and Experience. Special Issue: Practical Aspects of Search-Based Software Engineering 41(5), 579–606 (2011), doi:10.1002/spe.1043
5. Goh, L., Song, Q., Kasabov, N.: A novel feature selection method to improve classification of gene expression data. In: Chen, Y.P. (ed.) Proceedings of the Second Conference on Asia-Pacific Bioinformatics, pp. 161–166. Australian Computer Society, Darlinghurst (2004)
6. Güler, İ., íbeyli, E.D.: Feature saliency using signal-to-noise ratios in automated diagnostic systems developed for doppler ultrasound signals. Engineering Applications of Artificial Intelligence 19(1), 53–63 (2006)
7. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (2003)
8. Hall, M.A., Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining. IEEE Transaction on Knowledge and Data Engineering 15(6), 1437–1447 (2003)
9. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice-Hall (1998)
10. Ilczuk, G., Mlynarski, R.: W Kargul, and A Wakulicz-Deja. New feature selection methods for qualification of the patients for cardiac pacemaker implantation. Computers in Cardiology 34(2-3), 423–426 (2007)
11. Jiang, Y., Lin, J., Cukic, B., Menzies, T.: Variance analysis in software fault prediction models. In: Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering, Bangalore, Mysore, India, November 16-19, pp. 99–108 (2009)
12. Khoshgoftaar, T.M., Bullard, L.A., Gao, K.: Attribute selection using rough sets in software quality classification. International Journal of Reliability, Quality and Safty Engineering 16(1), 73–89 (2009)
13. Khoshgoftaar, T.M., Gao, K.: Feature selection with imbalanced data for software defect prediction. In: Proceedings of the 8th International Conference on Machine Learning and Applications, Miami, Florida, USA, December 13-15, pp. 235–240 (2009)
14. Khoshgoftaar, T.M., Golawala, M., Van Hulse, J.: An empirical study of learning from imbalanced data using random forest. In: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, USA, vol. 2, pp. 310–317 (2007)
15. Kira, K., Rendell, L.A.: A practical approach to feature selection. In: Proceedings of 9th International Workshop on Machine Learning, pp. 249–256 (1992)
16. Koru, A.G., Zhang, D., El Emam, K., Liu, H.: An investigation into the functional form of the size-defect relationship for software modules. IEEE Transactions on Software Engineering 35(2), 293–304 (2009)
17. Lakshmi, K., Mukherjee, D.S.: An improved feature selection using maximized signal to noise ratio technique for tc. In: Proceedings of the Third international Conference on information Technology: New Generations, pp. 541–546. IEEE Computer Society Press, Washington, DC (2006)
18. Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 34(4), 485–496 (2008)

19. Liu, H., Yu, L.: Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and Data Engineering 17(4), 491–502 (2005)
20. Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33(1), 2–13 (2007)
21. Rodriguez, D., Ruiz, R., Cuadrado-Gallego, J., Aguilar-Ruiz, J.: Detecting fault modules applying feature selection to classifiers. In: Proceedings of 8th IEEE International Conference on Information Reuse and Integration, Las Vegas, Nevada, August 13-15, pp. 667–672 (2007)
22. Saeys, Y., Abeel, T., Van de Peer, Y.: Robust Feature Selection Using Ensemble Feature Selection Techniques. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 313–325. Springer, Heidelberg (2008)
23. Shawe-Taylor, J., Cristianini, N.: Support Vector Machines, 2nd edn. Cambridge University Press (2000)
24. Wang, H., Khoshgoftaar, T.M., Gao, K., Seliya, N.: Mining data from multiple software development projects. In: Proceedings of 2009 IEEE International Conference on Data Mining Workshops, Miami, Florida, USA, December 6, pp. 551–557 (2009)
25. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann (2005)
26. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: An Introduction. Kluwer International Series in Software Engineering. Software Engineering. Kluwer Academic Publishers, Boston (2000)
27. Yang, C.-H., Huang, C.-C., Wu, K.-C., Chang, H.-Y.: A Novel GA-Taguchi-Based Feature Selection Method. In: Fyfe, C., Kim, D., Lee, S.-Y., Yin, H. (eds.) IDEAL 2008. LNCS, vol. 5326, pp. 112–119. Springer, Heidelberg (2008)
28. Zimmermann, T., Premraj, R., Zeller, A.: Predicting defects for eclipse. In: Proceedings of the 29th International Conference on Software Engineering Workshops, p. 76. IEEE Computer Society Press, Washington, DC (2007)