8-1-2009

An Empirical Investigation of Filter Attribute Selection Techniques for Software Quality Classification

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

Taghi M. Khoshgoftaar
Florida Atlantic University, taghi@cse.fau.edu

Huanjing Wang
Western Kentucky University, huanjing.wang@wku.edu

An Empirical Investigation of Filter Attribute Selection Techniques for Software Quality Classification

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

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

Huanjing Wang
Western Kentucky University
Bowling Green, Kentucky 42101
huanjing.wang@wku.edu

***Abstract*—Attribute selection is an important activity in data preprocessing for software quality modeling and other data mining problems. The software quality models have been used to improve the fault detection process. Finding faulty components in a software system during early stages of software development process can lead to a more reliable final product and can reduce development and maintenance costs. It has been shown in some studies that prediction accuracy of the models improves when irrelevant and redundant features are removed from the original data set. In this study, we investigated four filter attribute selection techniques, Automatic Hybrid Search (AHS), Rough Sets (RS), Kolmogorov-Smirnov (KS) and Probabilistic Search (PS) and conducted the experiments by using them on a very large telecommunications software system. In order to evaluate their classification performance on the smaller subsets of attributes selected using different approaches, we built several classification models using five different classifiers. The empirical results demonstrated that by applying an attribution selection approach we can build classification models with an accuracy comparable to that built with a complete set of attributes. Moreover, the smaller subset of attributes has less than 15 percent of the complete set of attributes. Therefore, the metrics collection, model calibration, model validation, and model evaluation times of future software development efforts of similar systems can be significantly reduced. In addition, we demonstrated that our recently proposed attribute selection technique, KS, outperformed the other three attribute selection techniques.**

I. Introduction

Software quality is an important attribute of software product especially for high-assurance and mission-critical systems. Predicting the quality of software modules in the early stages of software development process is very critical, so that software quality assurance efforts can be prioritized for targeting those modules that are either high-risk, or likely to have a high number of faults. Software quality models are the tools to implement such predictions. A software quality model can use software metrics that are collected prior to software testing and operations to estimate the quality factor of software modules such as number of faults or quality based classes, fault-prone and not fault-prone [1].

Over the last two decades, significant research has been dedicated towards developing methods for improving the predictive accuracy of software quality models [2], [3], [4]. It has been shown in some studies that the performance of these models improves when irrelevant and redundant features are eliminated from the original data set [2], [5], [4]. In addition, attribute selection can reduce the time for the metrics collection, model calibration, model validation, and model evaluation of future software development efforts of similar systems.

In this paper, we investigated four different attribute selection techniques, AHS, PS, KS and RS and applied them to a data set for a very large telecommunications software system. In order to evaluate their classification performance on the smaller subsets of attributes selected using various approaches, we built several classification models using five different classifiers. They are Instance-based learning (IBK), Multilayer perceptron (MLP), Support vector machine (SVM), Naïve Bayes (NB), and Logistic Regression (LR). The experimental results demonstrate that the classification accuracy of the models built with some smaller subsets of attributes is comparable to that built with the complete set of attributes. Moreover, the smaller subsets of attributes include less than 15 percent of the complete set of attributes. In addition, among the four attribute selection approaches, our recently proposed KS method performed better than the other three techniques in terms of two performance metrics (AUC and BGM) for four out of five learners.

The rest of the paper is organized as follows. Section II provides more detail information about the attribute selection techniques, five classifiers used in the paper and performance metrics. The data set used in the experiment is described in Section III. Section IV presents the experimental results. Finally, the conclusion is drawn in Section V.

II. Methodology

A. Attribute Selection Techniques

Attribute selection is a process of reducing data dimension. It is one of the frequently used techniques in data preprocessing for data mining. Attribute selection process consists of four basic steps [6]:

1) **Subset generation**. It produces candidate attribute subset based on a certain search strategy. In this step, two problems need to be solved. First, where to start. According to the different strategies, we can divide them into two categories, *forward* (search starts with an empty set and inserts attributes subsequently) and *backward* (search starts with a full set and deletes attributes subsequently). Second, how to produce next candidate subsets. Based on different strategies that would be used, we divide search into *complete search* also called exhaustive search and *sequential search*. A complete search examines all the attribute subsets of a given data set. It is able to find the optimal result at the expense of $O(2^N)$ computational complexity, where N is the number of attributes in the data set. In contrast, a sequential search gives up completeness and thus it cannot guarantee to find the optimal subsets. There are many variations to the greedy hill-climbing approach, such as sequential forward selection and sequential backward elimination [7].
2) **Subset evaluation**. During the search process, each generated subset need to be assessed by an evaluation criterion. If the new subset turn out to be better, it substitutes the previous subset. The process of subset generation and evaluation is an iterative activity until a stoping criterion is met. Subset evaluation can be divided into two categories, *filter* and *wrapper* based on their dependency on a data mining algorithm. The filter model evaluates the subset of attributes by examining the intrinsic

IEEE IRI 2009, July 10-12, 2009, Las Vegas, Nevada, USA

978-1-4244-4116-7/09/$25.00 ©2009 IEEE

characteristic of the data without involving any data mining algorithm. In contrast, the wrapper model evaluates the goodness of the subset of attributes by applying a predetermined data mining algorithm on the selected subset of attributes. It tends to be computationally expensive.

3) **Stopping criterion**. It determines when to stop search algorithm. Various stoping criteria are frequently used.
4) **Result validation**. It is used to assess the effectiveness of an attribution selection method. Various performance metrics can be used to evaluate the prediction models before and after the attribution selection was made.

For both AHS and PS methods, we used a modified version of consistency rate as the evaluation criterion [8]. Consistency rate (CR) has monotonic property. The property has the following facts:

- The complete attribute set has the highest consistency rate δ. In other words, the consistency rate of any attribute subset is less than or equal to δ;
- The superset of a consistent attribute subset is also consistent;
- If $\mathrm{CR}(S_i, D) \leq \mathrm{CR}(S_j, D)$, then $\mathrm{CR}(S_i \cap f, D) \leq \mathrm{CR}(S_j \cap f, D)$, where f is an attribute not in S_i and S_j.

The **Probabilistic Search** (PS) algorithm makes probabilistic choices of subsets in searching. PS employs two possible stopping criteria, generation time and when the best subset is found. The probabilistic search [8] implemented in this study using generation time as the stopping criterion proceeds as follows: attribute subsets are randomly generated with equal probability and evaluated; the algorithm will not stop until reaching the specified generation time. The smallest consistent attribute subset is selected. If an algorithm's purpose is to find a result with a certain amount of tolerance, setting the running time as the main stopping criterion is the most reasonable method. LVF is a probabilistic search algorithm [8]. A modified LVF implemented in this study is illustrated in Figure 1.

```
Algorithm: Probability search
Input:
    D, dataset
    S, full feature set of D
    Max: the maximum number of iterations
Output:
    L, consistent feature subset
Method:
(1)  L = S
(2)  δ = conCal(S, D)
(3)  for j=1 to Max
(4)      randomly choose a feature subset S_j
(5)      tempCon = conCal(S_j ,D)
(6)      if |S_j| < |L| and tempCon ≥ δ
(7)          L = S
(8)  return L
```

Fig. 1. PS Algorithm

The **Automatic Hybrid Search** (AHS), an attribute subset selection method recently proposed by us [9], also uses the consistency rate properties. It works as follows: the consistency rate of complete attribute set is computed first and is used as the stopping criterion. Starting with size 1 of any attribute, attribute subsets that have the locally highest consistency rate are selected. These selected attribute subsets will be used to generate superset. Repeat the process until finding the attribute subsets that have the same consistency rate or the complete attribute set is reached. If more than one attribute subsets are generated, a classifier called C4.5 [10] will be used to decide which attribute subset is selected based on an error rate. C4.5 is an algorithm for inducing classification rules in the form of a decision tree from a given data set. For this case study, one attribute subset (with six attributes) was produced without using C4.5. The AHS algorithm is illustrated in Figure 2.

```
Algorithm: Automatic hybrid search
Input:
    D, dataset
    S, full feature set of D
Output:
    L, consistent feature subsets
Method:
(1)  L = S
(2)  δ = conCal(S, D)
(3)  T = all feature subset S' in S where |S'| = 1
(4)  max = -∞
(5)  while the size of any set in T < |S|
(6)      tempSet = φ
(7)      for each set T' in T
(8)          tempCon = conCal(T', D)
(9)          if max < tempCon
(10)             max = tempCon
(11)             tempSet = φ
(12)             tempSet = append(tempSet, T')
(13)         else if max = tempCon
(14)             tempSet = append(tempSet, T')
(15)     if max ≥ δ
(16)         L = tempSet
(17)         return L
(18)     else if |tempSet| = |T|
(19)         T = combinationSet(T, size + 1)
(20)     else
(21)         for any set tempSet' in tempSet
(22)             append tempSet' with f where f is any feature in S
                     not in tempSet'
(23)         T = tempSet
(24) return L
Step 2:
Input:
    L, consistent feature subsets from step 1
Output:
    T, selected feature subset
Method:
(1)  min = ∞
(2)  T = φ
(3)  for each set L' in L
(4)      calculate error rate r using C4.5 with L'
(5)      if r < min
(6)          min = r
(7)          T = L'
(8)  return T
```

Fig. 2. AHS Algorithm

The **K-S Method** is an attribute selection method recently proposed by our research group [11]. It utilizes the Kolmogorov-Smirnov (KS) statistic to measure the maximum differences between the empirical distribution function of the posterior probabilities of instances in each class. The larger the distance between the distribution functions, the better the attribute is able to distinguish between the two classes. The attributes can be ranked based on their K-S scores and be selected according to their K-S scores and the number of attributes needed. The more detail information about the calculation of K-S score is described below.

For the jth independent variable, the data is composed of two independent samples, fault-prone (*fp*) and not fault-prone (*nfp*) samples. The *fp* sample has a size of n_{fp}, and its components are referenced as $x_j^{(k)}$, $k = 1, \ldots, n_{fp}$. The *nfp* sample contains n_{nfp} software modules, and it is composed of $x_j^{(l)}$, $l = 1, \ldots, n_{nfp}$.

Let $F_{X_j^{fp}}(x_j)$ and $F_{X_j^{nfp}}(x_j)$ represent the cumulative distribution functions of *fp* and *nfp* samples, respectively, for the jth independent variable. $S_{X_j^{fp}}(x_j)$ is an empirical cumulative distribution function of *fp* sample for the jth independent variable, that is defined as the percentage of X_j^{fp} which are less than or equal to x_j. $S_{X_j^{fp}}(x_j)$ is calculated by:

$$S_{X_j^{fp}}(x_j) = \frac{N_{fp}(x_j)}{n_{fp}} \tag{1}$$

where $N_{fp}(x_j)$ is the number of elements that are less than or equal to x_j, which correspond to the set of $\{x_j^{(k)} \mid x_j^{(k)} \leq x_j, k = 1, \ldots, n_{fp}\}$.

Similarly, $S_{X_j^{nfp}}(x_j)$ is defined as the empirical cumulative distribution function of the *nfp* sample for the jth independent variable. $S_{X_j^{nfp}}(x_j)$ is computed by the following formula:

$$S_{X_j^{nfp}}(x_j) = \frac{N_{nfp}(x_j)}{n_{nfp}} \tag{2}$$

where $N_{nfp}(x_j)$ is the number of elements of X_j^{nfp} that are less than or equal to x_j. In other words, it consists of the set $\{x_j^{(l)} \mid x_j^{(l)} \leq x_j, l = 1, \ldots, n_{nfp}\}$.

The greatest vertical distance, T_{ks}, for the jth independent variable is computed using the formula below.

$$T_{ks} = \max_{x_j} |S_{X_j^{fp}}(x_j) - S_{X_j^{nfp}}(x_j)| \tag{3}$$

T_{ks} is K-S score for attribute j.

The **Rough Sets** (RS) theory is based on classical set theory [12]. In this study, we are interested in the partitions that are constructed from groups of attributes, called equivalence classes. Using the concept of *equivalence* from classical set theory, we can eliminate redundant data and insignificant attributes. A *reduct* is a minimal set of attributes that preserves the discrimination power and the ability to perform classifications as if we are using the whole attribute set. There exist a number of genetic algorithms [13] that can compute a sufficient number of reducts for a given attribute set. Genetic algorithms use heuristic techniques to search the attibute subset space. For this case study, we used RSES tools to generate reducts. RSES is an abbreviation for the Rough Set Exploration System, which is a set of software tools that are used for rough set computations in data mining [14], [2].

The AHS, PS and RS methods need the input data to be discretized before using them. We used the WEKA tool to discretize the data for both AHS and PS with the *equal frequency* strategy [15]. The number of bins was set to 10. The RS method was implemented by the RSES tools which include the function of discretizing data.

B. Classifiers

The five classifiers used in this case study are instance-based learning, multilayer perceptron, support vector machine, naïve Bayes, and logistic regression [16], [17], [18], [19], [20]. All these five learners themselves do not have the attribute selection capability. Every classifier was implemented in the WEKA tool [15]. Default parameter changes were done only when classifier performance improved significantly.

IBK [16], also called *k* nearest neighbors (*k*NN) classifier, was built with changes to two parameters. The 'distanceWeighting' parameter was set to 'Weight by 1/distance', the '*k*NN' parameter was set to '30', and the 'crossValidate' parameter was turned on (set to 'True'). 'crossValidate' tells the classifier to try each *k* between 1 and the value of the *k*NN parameter, picking the one which performs best on the training data and using that for the actual classification. By default this process chooses the *k* which optimizes overall accuracy, however we modified the code slightly so that it chooses the *k* which produces the highest mean of the accuracies for each class (i.e., the arithmetic mean between the true positive rate and true negative rate).

TABLE I
CONFUSION MATRIX FOR A BINARY CLASSIFICATION

		Correct Result	
		+	-
Obtained	+	TP	FP
Result	-	FN	TN

The support vector machine (SVM) classifier [18] called SMO in WEKA had two changes to the default parameters: the complexity constant c was set to 5.0 and buildLogisticModels was set to true. By default, a linear kernel was used.

For a multilayer perceptrons (MLP) classifier [17] (a type of neural network), the 'hiddenLayers' parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process.

Naïve Bayes (NB) utilizes Bayess rule of conditional probability and is termed 'naïve' because it assumes conditional independence of the features [19].

Logistic regression (LR) [20] is a statistical regression model for categorical prediction.

C. Performance Metrics

In a binary (positive and negative[1]) classification problem, there can be four possible outcomes of classifier prediction: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). A two-by-two confusion matrix is described in Table I. The four values provided by the confusion matrix form the basis for several other performance metrics that are well known and commonly used within the data mining and machine learning community.

The **Area Under the ROC** (receiver operating characteristic) curve (i.e., AUC) is a single-value measurement that originated from the field of signal detection. The value of the AUC ranges from 0 to 1. The ROC curve is used to characterize the trade-off between hit (true positive) rate and false alarm (false positive) rate [21]. It depicts the performance of a classifier without taking class distribution or error costs into consideration. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. A perfect classifier provides an AUC that equals 1.

The **Geometric Mean** (GM) is a single-value performance measure that ranges from 0 to 1, and a perfect classifier provides a value of 1. It 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. Such error rates are often preferred, depending on the misclassification costs and the application domain. The threshold $t = 0.5$ is used for the Default Geometric Mean (DGM). The Best Geometric Mean (BGM) is the maximum Geometric Mean value that

[1]positive and negative refer to *fp* and *nfp* modules respectively.

is obtained for $0 < t < 1$. The default decision threshold usually causes the classifier to perform suboptimally when the given data set has an imbalanced class distribution or when the costs for both types of misclassification are not equal. The decision threshold can be changed to account for class imbalance in the data set and for unequal costs of misclassification. Lowering the decision threshold causes the classifier to assign more instances to the positive class, and thus increase the number of FP errors while reducing the number of FN errors. The best geometric mean is implemented by this adjustment.

III. Data Set Description

The software metrics and fault data for this case study (denoted as LLTS) was collected from a very large legacy telecommunications software system. The LLTS software system was developed in a large organization by professional programmers using a proprietary high level procedural language, PROTEL. The system was comprised of several million lines of code. The data collection effort used the Enhanced Measurement for Early Risk Assessment of Latent Defect (EMERALD) system [22]. A decision support system for software measurements and software quality modeling, EMERALD periodically measures the static attributes of the most recent version of the software code. We used the data set collected from the LLTS software system to conduct our experiments. This data set contains 3649 program modules. Each set of associated source code files was considered as a module. The LLTS data set consists of 42 software metrics including 24 product metrics, 14 process metrics, and 4 execution metrics shown in Table II. The dependent variable is the class of the software module, fault-prone or not fault-prone. The fault-proneness is based on a selected threshold, i.e., modules with one or more faults were considered as fault-prone, not fault-prone otherwise.

IV. Experiments

A. Selected Attributes

The experiments were conducted with the four attribute selection techniques. It is worthwhile to note that we restricted the maximum number of selected attributes to six which is obtained from $\lceil \log_2 42 \rceil$, where 42 is the number of attributes of the original data set [23]. The reason is that usually the performance of a model will be getting better when the number of attributes involved in model construction is increasing. Different attribute selection approaches may adopt different stoping criteria so that they may produce different sizes of attribute subsets. In order to remove the effect of the number of selected attributes on the performance of classification models, we used this criterion for selecting the number of attributes.

Table III lists the attributes selected by the four methods. For AHS and KS, it is relatively easy to control the number of selected attributes. For example, the KS method ranks the attributes based on their K-S scores. The larger the score, the better the attribute is able to distinguish between the two classes. So the top six attributes were selected. For the AHS method, the restriction for the number of the selected attributes can be implemented by adjusting the consistency rate requirement.

The RS method used three different genetic filter algorithms described in [24] to determine the most significant subsets of attributes(s). The training data set was discretized by the algorithm implemented in RSES [14] before using the filter algorithms. To determine significant attribute subsets, we choose the subsets that were generated by all three algorithms. Each algorithm had to generate at least 20 subsets, for a total of 60 attribute subsets, before there were two subsets of attributes that were common to all three algorithms. After reviewing the reducts, we observed that of the 60

TABLE II
Software Metrics

Symbol	Description
Product Metrics	
CALUNQ	Number of distinct procedure calls to others.
CAL2	Number of second and following calls to others. CAL2 = CAL - CALUNQ where CAL is the total number of calls.
CNDNOT	Number of arcs that are not conditional arcs.
IFTH	Number of non-loop conditional arcs (i.e., if-then constructs).
LOP	Number of loop constructs.
CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs.
CNDSPNMX	Maximum span of branches of conditional arcs.
CTRNSTMX	Maximum control structure nesting.
KNT	Number of knots. A "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles.
NDSINT	Number of internal nodes (i.e., not an entry, exit, or pending node).
NDSENT	Number of entry nodes.
NDSEXT	Number of exit nodes.
NDSPND	Number of pending nodes (i.e., dead code segments).
LGPATH	Base 2 logarithm of the number of independent paths.
FILINCUQ	Number of distinct include files.
LOC	Number of lines of code.
STMCTL	Number of control statements.
STMDEC	Number of declarative statements.
STMEXE	Number of executable statements.
VARGLBUS	Number of global variables used.
VARSPNSM	Total span of variables.
VARSPNMX	Maximum span of variables.
VARUSDUQ	Number of distinct variables used.
VARUSD2	Number of second and following uses of variables. VARUSD2 = VARUSD - VARUSDUQ where VARUSD is the total number of variable uses.
Process Metrics	
DES_PR	Number of problems found by designers during development of the current release.
BETA_PR	Number of problems found during beta testing of the current release.
DES_FIX	Number of problems fixed that were found by designers in the prior release.
BETA_FIX	Number of problems fixed that were found by beta testing in the prior release.
CUST_FIX	Number of problems fixed that were found by customers in the prior release.
REQ_UPD	Number of changes to the code due to new requirements.
TOT_UPD	Total number of changes to the code for any reason.
REQ	Number of distinct requirements that caused changes to the module.
SRC_GRO	Net increase in lines of code.
SRC_MOD	Net new and changed lines of code.
UNQ_DES	Number of different designers making changes.
VLO_UPD	Number of updates to this module by designers who had 10 or less total updates in entire company career.
LO_UPD	Number of updates to this module by designers who had between 11 and 20 total updates in entire company career.
UPD_CAR	Number of updates that designers had in their company careers.
Execution Metrics	
USAGE	Deployment percentage of the module.
RESCPU	Execution time (microseconds) of an average transaction on a system serving consumers.
BUSCPU	Execution time (microseconds) of an average transaction on a system serving businesses.
TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

TABLE III
SELECTED ATTRIBUTES

ID	Metrics	AHS	KS	RS 1	RS 2	PSES 1	PSES 2	PSES 3	PSES 4
1	DES_PR								
2	BETA_PR								
3	DES_FIX						x		x
4	BETA_FIX								
5	CUST_FIX	x							
6	REQ_UPD								x
7	TOT_UPD		x						
8	REQ					x	x		
9	SRC_GRO					x			
10	SRC_MOD	x							x
11	UNQ_DES	x		x					
12	VLO_UPD								
13	LO_UPD							x	
14	UPD_CAR	x				x			
15	USAGE	x				x	x	x	x
16	CALUNQ				x				
17	CAL3								
18	CNDSPNSM	x							x
19	CNDSPNMX					x			
20	CTRNSTMX							x	
21	FILINCUQ		x		x	x	x		
22	KNT						x	x	
23	LOC								
24	CNDNOT								
25	IFTH		x						
26	LOP								
27	NDSENT								
28	NDSEXT								
29	NDSPND						x		
30	NDSINT		x						
31	LGPATH				x				
32	STMCTL								
33	STMDEC								
34	STMEXE							x	
35	VARGLBUS								
36	VARSPNSM		x						
37	VARSPNMX		x	x				x	
38	VARUSDUQ			x					
39	VARUSD2								x
40	RESCPU								
41	BUSCPU								
42	TANCPU								

AHS: Automatic Hybrid Search method
KS: Kolmogorov-Smirnov statistic method
RS: Rough Sets method
PSES: combination of probabilistic search and exhaustive search method

reducts generated, only 50 were unique. We choose the reducts that had the most votes. Each algorithm was able to cast one vote for a particular reduct. If an algorithm generated a particular reduct, then that event would be counted as a vote for the reduct. We have two subsets that got votes from all three algorithms.

For the PS algorithm, feature subsets are randomly generated in the iterative process. Therefore, for a given data set, you may get different results with different runs. In this experiment, after running the PS algorithm with 50,000 iterations, we obtained four different attribute subsets each with 8 attributes. Then, we employed the exhaustive search on each of these four reduced subsets. The exhaustive search is able to find the optimal result. However, the search space is $O(2^N)$, where N represents the number of the attributes in the data set. The exhaustive search is practical when N is small. For example, for the reduced attribute subsets (with 8 attributes), the search space is 2^8 which is 256. But it becomes computationally prohibitive when you try to apply it to the original data with 42 attributes. PSES in Table III indicates the combination of the probabilistic search method and (followed by) the exhaustive search method. We present all the four results for PSES in the last four columns of the table.

B. Performance Results

For this study, we assessed the classification performance by applying five learners (IBK, MLP, SVM, NB and LR) to the attribute subsets selected by the four attribute selection approaches. We also applied the five learners to the original data set and made the results as the base for comparison. For each learner, we had 10 runs each with a 10-fold cross-validation. The result for each leaner was summarized across a total of 100 outcomes. We used the WEKA tool [15] to build all classification models. The classification performance was evaluated in terms of the two performance metrics AUC and BGM. All the results are reported in Table IV. It can be seen that for both RS and PSES methods, the results presented are the average on their own selected attribute subsets. From the table, we can summarize the following facts:

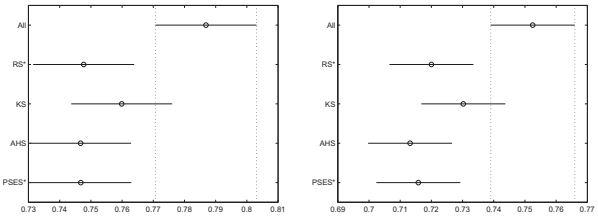
1) The classification accuracy in terms of both AUC and BGM for all five learners on the complete data set (with 42 attributes) outperforms those on the attribute subsets (with six or less attributes) selected by any attribute selection method as it was expected.
2) Among the four attribute selection techniques, KS performs better than the other three techniques in terms of AUC and BGM for four out of five learners. The four learners are MLP, SVM, Naive Bayes, and Logistic Regression. The best performances (excluding the complete attribute set) are highlighted with **bold** in Table IV.
3) Among the four attribute selection techniques, RS performs worse than the other three techniques in terms of AUC for three out of five learners, and in terms of BGM for one out of five learners. The reason is probably the smaller size of the attribute subsets, each with three attributes, selected by the RS method. The worst performances are highlighted with *italic* in Table IV.
4) The AHS method shows the best performance in terms of AUC and BGM among the four attribute selection techniques when using the IBK learner but shows the worst performance when using the SVM learner.

We also carried out a two-way Analysis of Variance (ANOVA) F test [25] on the performance metrics, AUC and BGM, to examine if the performance difference (better/worse) is significant or not. The two-way ANOVA test was designed as follows. Factor A represents the results from four different attribute selection techniques and the result without using any selection technique. Factor B represents the results from the five different classification models or learners. The test results indicate that the classification performances in terms of AUC/BGM were significantly different from each other for Factor A and also for Factor B (all $p < 0.01$). We further conducted the multiple comparison test [25] on Factor A with Tukey's honestly significant difference criterion, since this study mainly focuses on the attribute selection techniques and their classification performance. The multiple comparison test is shown in Figure 3, (a) AUC and (b) BGM. These figures show that the three attribute selection methods, PS, AHS and RS performed significantly worse than the complete data set (with 42 attributes) over the five learners ($p = 0.05$); while the predictive accuracy of the KS method was insignificantly lower than that of models built on the complete data set. Also, there were no significant differences in terms of AUC and BGM among all four attribute selection techniques. Both ANOVA and multiple comparison tests were implemented in MATLAB.

TABLE IV
PERFORMANCE METRICS

		AUC		BGM	
	Learner	mean	stdev	mean	stdev
1. All	IBK	0.7658	0.0434	0.7419	0.0387
	MLP	0.8128	0.0492	0.7688	0.0519
	SVM	0.7451	0.0592	0.7214	0.0483
	NB	0.7925	0.0449	0.7569	0.0416
	LR	0.8182	0.0502	0.7734	0.0476
2. RS*	IBK	0.7298	0.0476	0.7105	0.0426
	MLP	*0.7803*	0.0534	0.7393	0.0482
	SVM	0.6724	0.1146	0.6722	0.0817
	NB	*0.7748*	0.0532	*0.7383*	0.0507
	LR	*0.7811*	0.0537	*0.7398*	0.0491
3. KS	IBK	0.7239	0.0524	0.7068	0.0467
	MLP	**0.8022**	0.0493	**0.7588**	0.0474
	SVM	**0.6804**	0.1038	**0.6772**	0.0721
	NB	**0.7896**	0.0500	**0.7476**	0.0454
	LR	**0.8033**	0.0503	**0.7608**	0.0474
4. AHS	IBK	**0.7427**	0.0507	**0.7164**	0.0446
	MLP	0.7826	0.0491	*0.7371*	0.0488
	SVM	*0.6327*	0.0992	*0.6269*	0.0678
	NB	0.7840	0.0451	0.7424	0.0411
	LR	0.7915	0.0480	0.7431	0.0504
5. PSES*	IBK	0.7164	0.0527	0.6955	0.0471
	MLP	0.7893	0.0495	0.7461	0.0460
	SVM	0.6528	0.1096	0.6417	0.0789
	NB	0.7827	0.0476	0.7456	0.0453
	LR	0.7926	0.0474	0.7502	0.0450

* the performance metrics presented are the average for RS and PSES.



(a) AUC (b) BGM

Fig. 3. Multiple Comparison Test

V. CONCLUSION

Attribute selection plays an important role in data preprocessing. By removing irrelevant and redundant features from a training data set, software quality estimation based on some classification models may improve.

In this paper, we present four different attribute selection techniques and their applications to a very large telecommunications software system. The classification accuracy was evaluated in terms of two performance metrics AUC and BGM. The experimental results demonstrate that the KS method is better than the other three techniques, PS, AHS and RS. Also, the classification model built on the smaller subset of attributes via the KS method has a comparable (no significant difference) performance to that built with a complete set of attributes. Moreover, the smaller subset of attributes has less than 15 percent of number of the attributes in original data set. This would benefit the metrics collection, model calibration, model validation, and model evaluation times of future software project development efforts of similar systems. Future research may involve conducting more experiments, including other filter attribute selection techniques and more datasets from other software projects.

REFERENCES

[1] T. M. Khoshgoftaar and N. Seliya, "Comparitive assessment of software quality classification technique," *Empirical Sofware Engineering Journal*, vol. 9, no. 3, pp. 229–257, 2004.

[2] T. M. Khoshgoftaar, L. A. Bullard, and K. Gao, "Attribute selection using rough sets in software quality classification," *International Journal of Reliability, Quality and Safty Engineering*, vol. 16, no. 1, pp. 73–89, 2009.

[3] T. M. Khoshgoftaar, Y. Xiao, and K. Gao, "Assessment of a multi-strategy classifier for an embedded software system," in *Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence*, Washing D. C., November 13-15 2006, pp. 651–658.

[4] D. Rodriguez, R.Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, Las Vegas, IL, August 13-15 2007, pp. 667–672.

[5] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437 – 1447, Nov/Dec 2003.

[6] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[7] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Boston, MA: Springer, 1998.

[8] M. Dash and H. Liu, "Consistency-based search in feature selection," *Artificial Intelligence*, vol. 151, no. 1-2, pp. 151–176, 2003.

[9] P. Lin, H. Wang, and T. M. Khoshgoftaar, "A novel hybrid search algorithm for feature selection," in *Proceedings of 21st International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, July 1-3 2009, in press.

[10] J. R. Quinlan, *C4.5: programs for machine learning*. Los Altos, California: Morgan Kaufmann, 1993.

[11] T. M. Khoshgoftaar, L. Nguyen, K. Gao, and J. Rajeevalochanam, "Application of an attribute selection method to cbr-based software quality classification," in *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence*. Sacramento, California: IEEE, November 3-5 2003, pp. 47–52.

[12] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning About Data*. Kluwer Academic, 1992.

[13] J. Wróblewski, "Finding minimal reducts using genetic algorithms," in *Proceedings of the International Workshop on Rough Sets Soft Computing at Second Annual Joint Conference on Information Sciences*, 1995, pp. 186–189.

[14] J. G. Bazan, M. S. Szczuka, and J. Wróblewski, "A new version of rough set exploration system," in *Proceedings of 3rd International Conference, RSCTC2002*, 2002, pp. 397–404.

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

[16] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 1573–0565, January 1991.

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

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

[19] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, vol. 2, San Mateo, 1995, pp. 338–345.

[20] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.

[21] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, June 2006.

[22] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand, "EMERALD: Software metrics and models on the desktop," *IEEE Software*, vol. 13, no. 5, pp. 56–60, September 1996.

[23] T. M. Khoshgoftaar, , M. Golawala, and J. V. Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2. Washington DC, USA: IEEE Computer Society, 2007, pp. 310–317.

[24] J. Wróblewski, "Genetic algorithms in decomposition and classification problems," in *Rough Sets in Knowledge Discovery 2*, L. Polkowski and A. Skowron, Eds. Physica-Verlag, 1998, pp. 471–487.

[25] M. L. Berenson, M. Goldstein, and D. Levine, *Intermediate Statistical Methods and Applications: A Computer Package Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1983.