SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2011; **41**:579–606
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1043

Choosing software metrics for defect prediction: an investigation on feature selection techniques

Kehan Gao[1], Taghi M. Khoshgoftaar[2,*,†], Huanjing Wang[3] and Naeem Seliya[4]

[1]*Eastern Connecticut State University, Willimantic, CT, U.S.A.*
[2]*Florida Atlantic University, Boca Raton, FL, U.S.A.*
[3]*Western Kentucky University, Bowling Green, KY, U.S.A.*
[4]*University of Michigan-Dearborn, Dearborn, MI, U.S.A.*

SUMMARY

The selection of software metrics for building software quality prediction models is a search-based software engineering problem. An exhaustive search for such metrics is usually not feasible due to limited project resources, especially if the number of available metrics is large. Defect prediction models are necessary in aiding project managers for better utilizing valuable project resources for software quality improvement. The efficacy and usefulness of a fault-proneness prediction model is only as good as the quality of the software measurement data. This study focuses on the problem of attribute selection in the context of software quality estimation. A comparative investigation is presented for evaluating our proposed hybrid attribute selection approach, in which feature ranking is first used to reduce the search space, followed by a feature subset selection. A total of seven different feature ranking techniques are evaluated, while four different feature subset selection approaches are considered. The models are trained using five commonly used classification algorithms. The case study is based on software metrics and defect data collected from multiple releases of a large real-world software system. The results demonstrate that while some feature ranking techniques performed similarly, the automatic hybrid search algorithm performed the best among the feature subset selection methods. Moreover, performances of the defect prediction models either improved or remained unchanged when over 85% of the software metrics were eliminated. Copyright © 2011 John Wiley & Sons, Ltd.

Received 7 July 2009; Revised 26 July 2010; Accepted 15 October 2010

KEY WORDS: software metric; defect prediction; attribute selection; feature ranking; feature subset selection; search-based software engineering; software quality

1. INTRODUCTION

Defect prediction is an important process activity in software engineering practice. A verified strategy used for this task is building software quality prediction models that estimate the quality of program modules, e.g. fault-prone (*fp*) or not-fault-prone (*nfp*) [1–5]. A practitioner can apply such a model toward implementing a targeted and prioritized software quality improvement activity. This approach allows for a more cost-effective usage of the limited project resources.

Software attributes can characterize the software quality of both the product and the process of software development [6]. Attributes of software quality, such as defect density and failure rate, are external measures of the software product and its development process. We focus on utilizing software metrics, such as code-level measurements and defect data, to build defect predictors or software quality models. This is based on the practical assumption that these software metrics will

*Correspondence to: Taghi M. Khoshgoftaar, Florida Atlantic University, Boca Raton, FL, U.S.A.
†E-mail: taghi@cse.fau.edu

Copyright © 2011 John Wiley & Sons, Ltd.

capture the quality of the end product. By exploring knowledge stored in historical project data, characterized in the form of software metrics and defect data, one can build effective software quality prediction models.

Generally, a software quality prediction model is built using software metrics and defect data collected from a previously developed system release or similar software project. Upon validation of such a model, it is ready for predicting the fault-proneness of program modules that are currently under development. A low quality or *fp* prediction can justify the application of available quality improvement resources to those programs. In contrast, an *nfp* prediction can justify non-application of the limited resources to these already high-quality programs. The goal is to achieve high software reliability and quality with an effective use of the available resources.

Since a software quality prediction model is based on the knowledge stored in the known software metrics, the selection of the specific set of metrics becomes an integral component of the model-building process [6]. In other words, the quality of the software measurement data plays an important role in defect prediction. While not the focus of this study, reducing the different types of noise in software measurement data has been addressed in the literature [1, 4, 7]. The primary focus of this paper is on feature selection (or attribute selection) of software metrics for defect prediction, and thereby, improving the quality of the software measurement data.

We approach the problem from a search-based perspective, i.e. explore the available software metrics' space and derive a useful set of software metrics. This search is a preprocessing step to building the software defect prediction models. Feature selection algorithms can be categorized into two groups, *feature ranking* and *feature subset selection* [8]. Feature ranking assesses each individual feature (software metrics) according to some criterion and then the analyst selects some of the features that are appropriate for a given data set. In contrast, feature subset selection searches for subsets of attributes that collectively have good predictive capability. This is based on the assumption that a given attribute may have better predictability when combined with some other attributes, as compared to when used by itself. However, one of the problems for subset selection is that the number of candidate subsets may be prohibitively large and evaluating all of these is impractical. In this paper, we propose a hybrid attribute selection approach consisting of feature ranking followed by feature subset selection. Feature ranking prioritizes features and selects a pre-specified number of relevant features to reduce the search space. Subsequently, a search algorithm is applied to the reduced search space and feature subset selection is used to choose the final attribute subset.

In the context of software quality prediction, we empirically investigate seven different feature ranking techniques: chi-square (CS), information gain (IG), gain ratio (GR), Kolmogorov–Smirnov statistic (KS), two forms of the Relief algorithm (RLF), and symmetrical uncertainty (SU). The results of feature ranking are then analyzed by three different feature subset selection methods (algorithms): exhaustive search (ES), heuristic search (HS), and automatic hybrid search (AHS). We also consider no subset selection algorithm (referred to as NONE), thus, providing four techniques for attribute subset searching. We note that the KS feature ranking technique and the AHS feature subset selection algorithm were proposed by our team in earlier works [9, 10]. Our approach combines both, feature ranking and feature subset selection, for hybrid search-based approach for selecting the appropriate software metrics. Subsequent to attribute selection, we consider five different learners for training defect predictors [11]: naïve Bayes (NB), multilayer perceptron (MLP), support vector machine (SVM), logistic regression (LR), and k nearest neighbor (KNN).

A comparative analysis based on statistical computations suggests that software quality prediction models based on the various feature ranking techniques have similar performances, except for the CS method. In addition, the AHS feature subset selection algorithm yielded a better predictive performance than the other search algorithms. A further empirical analysis demonstrates that even after removing over 85% of the software metrics from the original data, model performances remained similar or even improving in some cases. It is shown that a reduced search space (of software metrics) does not deteriorate the outcomes of the feature subset selection methods, nor does it adversely affect the predictive performances of the classification models. In fact, the subsets of software attributes obtained from the reduced search spaces are more relevant/associated to the class (fault-proneness label) attribute.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

The key contributions of this research are

- We used seven different feature ranking techniques and three different feature subset selection methods in this study. Such an extensive investigation on attribute selection for software defect prediction is unique to this study.
- A novel hybrid search-based approach to attribute selection that involves feature ranking followed by feature subset selection is presented. Not only does it speed up the overall process, but also provides improved predictive performances.
- A comprehensive empirical study based on real-world software engineering projects is used for our comparative analysis. To our knowledge, no related work on attribute selection in software engineering has used an independent test data for validation purposes. The generalization performance of a software quality prediction model is key to its practicality and usefulness.

The remainder of the paper is organized as follows. Section 2 discusses some related work. The software measurement data used in the empirical studies are described in Section 3. Section 4 provides more detailed information about the seven attribute ranking techniques and three subset selection techniques (search algorithms) used in the study. Section 5 presents the different case studies, including their experimental design, results, analysis, and threats to empirical validity. We summarize the paper in Section 6.

2. RELATED WORK

There has been an increase in work on Search Based Software Engineering (SBSE) techniques for solving optimization problems, including requirements, project planning, maintenance, and re-engineering. Harman *et al.* [12, 13] provide a comprehensive survey on SBSE-related studies. In this paper, we provide a unique insight into feature selection for obtaining a good set of software metrics for defect prediction. To our knowledge, very little work has been done on feature selection for software quality prediction modeling.

Rodríguez *et al.* [4, 14] applied feature selection with three filter models and three wrapper models to five software engineering data sets. They conclude that the reduced data sets maintained the prediction capability with fewer attributes than the original data sets. In addition, while it was stated that the wrapper model was better than the filter model, it came at a high computational cost. In contrast to our study, their conclusions were based on evaluating models using cross validation instead of an independent test data set. It is known in the software engineering and data mining communities that prediction models are best evaluated based on their generalization performance, i.e. using a test data set. This is important for avoiding an overfitting of the prediction model to the training data set. In our case studies, three independent validation (test) data sets are used—as discussed in the later sections. Chen *et al.* [15] have studied feature selection using wrappers in the context of software cost/effort estimation, and conclude that the reduced data set could improve the estimation.

Feature selection is an important data preprocessing task in many data mining and machine learning applications. Liu and Yu [16] provide a survey of feature selection algorithms and present an integrated approach to intelligent feature selection. Guyon and Elisseeff [8] outline the key approaches used for attribute selection, including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. Hall and Holmes [7] investigated six attribute selection techniques that produce ranked lists of attributes and applied them to 15 data sets from the UCI repository. They conclude that while the ranking techniques yielded similar results, a wrapper/filter approach to attribute selection was the best.

Many applications of feature selection in various fields have been reported [17–19]. Jong *et al.* [20] introduced methods for feature selection based on SVMs. Ilczuk *et al.* [21] highlighted the importance of attribute selection in judging the qualification of patients for cardiac pacemaker implantation. In the context of text mining, where attributes are binary in value, Forman [18] investigates multiple filter-based feature ranking techniques.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

The limitations of the studies discussed above include that no comparative studies were performed between the performance of the classification models using the reduced data set with the selected features and the original data set with all the features. One of the possible reasons is that the characteristics of data in some specific domains restrict such comparisons. For example, the high number of available features found in bioinformatics data sets makes it difficult to build classification models using all the attributes, preventing further comparison. In addition, the pattern of attributes found in some studies may not appear in other domains. For instance, binary attributes are often seen in the domain of text classification, but may not be expected in other fields like software metrics. In our study, we investigate attributes with numeric values in the software engineering domain. In order to validate the effectiveness of our proposed feature selection methods, we compare the classification models built with the selected attribute subsets with those built with the original data set and report the results.

3. SOFTWARE MEASUREMENT DATA

The software metrics and fault data for this case study were collected over four historical releases from a very Large Legacy Telecommunications software System (denoted as LLTS), using the procedural paradigm and the PROTEL[‡] language, and maintained by professional programmers in a large organization. We labeled the four system releases 1 through 4. These releases were the last four releases of the legacy system. The telecommunications system had over 10 million lines of code and included numerous finite-state machines and interfaces to various kinds of equipment.

A software module was considered as a set of functionally related source-code files according to the system's architecture. A module was considered *fp* if any faults were discovered during operations, and *nfp* otherwise. A software fault for a program module was recorded only when the problems discovered by customers resulted in changes to the module's source code. Faults in deployed telecommunications systems are extremely expensive because, in addition to down-time due to failures, visits to remote sites are usually necessary to repair them. Preventing customer-discovered faults was a very high priority for the developers of this system, and thus, they were interested in timely software quality predictions.

Fault data, collected at the module level by the problem reporting system, consisted of faults discovered during post unit testing phases, and were recorded before and after the product was released to customers. It was observed that over 99% of the unchanged modules had no faults. Consequently, this case study considered modules that were new, or had at least one source code update since the prior release. Configuration management data analysis identified software modules that were unchanged from the prior release. Table I presents the distribution of the faults discovered in the updated modules of the four system releases by the customers.

Each release had approximately 3500 to 4000 updated software modules. The number of modules considered in Releases 1, 2, 3, and 4 (also referred to as Fit, Test 1, Test 2, and Test 3) were 3649, 3981, 3541, and 3978, respectively. According to the definitions of *fp* and *nfp* modules for this case study, the numbers of *fp* and *nfp* modules for the four releases are as follows: Releases 1–4 have 229, 189, 47, and 92 *fp* modules, and 3420, 3792, 3494, and 3886 *nfp* modules, respectively. The proportion of modules with no faults among the updated modules of the first release (fit data set) was 0.937, and the proportion with at least one fault was 0.063.

The set of available software metrics is usually determined by pragmatic considerations. A broad set of metrics are analyzed rather than limiting data collection according to a predetermined set of research questions. Data collection for this case study involved extracting source code from the configuration management system. The available data collection tools selected the software metrics. Software measurements were recorded using the EMERALD (Enhanced Measurement

[‡]PROTEL stands for 'Procedure Oriented Type Enforcing Language'. It is a programming language used on telecommunications switching systems by Nortel Networks.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table I. LLTS fault distribution.

	Percentage of updated modules			
Faults	Rel. 1 (Fit)	Rel. 2 (Test 1)	Rel. 3 (Test 2)	Rel. 4 (Test 3)
0	93.7%	95.3%	98.7%	97.7%
1	5.1%	3.9%	1.0%	2.1%
2	0.7%	0.7%	0.2%	0.2%
3	0.3%	0.1%	0.1%	0.1%
4	0.1%	*		
6	*			
9	*			

*One module.

Table II. Software product metrics.

Symbol	Description
Call graph 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.
Control flow graph metrics	
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.
Statement metrics	
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 whereVARUSD is the total number of variable uses.

for Early Risk Assessment of Latent Defects) software metrics analysis tool, which includes software-measurement facilities and software quality models [22].

Preliminary data analysis selected metrics (aggregated to the module level) that were appropriate for our modeling purposes. The software metrics considered included 24 product metrics, 14 process metrics, and 4 execution metrics. Consequently, this case study consists of 42 independent variables that are used to predict the response variable, i.e. *Class*. It should be noted that the sets of software metrics used for this case study may not be universally appropriate for all software systems. Another project might collect (depending on availability) and use a different set of software metrics.

The software product metrics in Table II are based on call graphs, control flow graphs, and statement metrics. The number of procedure calls by each module (CALUNQ and CAL2) is

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table III. Software process metrics.

Symbol	Description
DES_PR	Number of problems found by designers during development of the current release.
BETA_PR	Number of problems found during beta testing ofthe 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 their entire company career.
LO_UPD	Number of updates to this module by designers who had between 11 and 20 total updates in their entire company career.
UPD_CAR	Number of updates that designers had in their company careers.

Table IV. Software execution metrics.

Symbol	Description
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.

derived from a call graph depicting calling relationships among procedures. A module's control flow graph consists of nodes and arcs depicting the flow of control of the program. Statement metrics are measurements of the program statements without expressing the meaning or logic of the statements.

Process metrics in Table III may be associated with either the likelihood of inserting a fault during development or the likelihood of discovering and fixing a fault prior to product release. The configuration management systems tracked each change to source code files, including the identity of the designer and the reason for the change, e.g. a change to fix a problem or to implement a new requirement. The problem reporting system maintained records of past problems.

Execution metrics listed in Table IV are associated with the likelihood of executing a module, i.e. operational use. The proportion of installations that had a module, USAGE, was approximated by deployment data on a prior release. Execution times were measured in a laboratory setting with different simulated workloads.

4. METHODOLOGY

Feature ranking techniques assess attributes individually based on a given criterion and order them accordingly. However, it was found that sometimes an attribute may be less useful by itself but can make a significant contribution when combined with other attributes. Feature subset selection approaches consider this issue by searching and selecting subsets of attributes that collectively have good performance. Attribute selection can also be divided into *wrappers* and *filters* [16]. Wrappers are algorithms that use feedback from a learning algorithm to determine which attributes to use in building a predictive model. In filters, the training data are analyzed using a method that does not require a learning algorithm to determine which attribute(s) are most relevant. In this paper, we only consider filters instead of wrappers for feature selection. The reasons include that

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

(1) the use of wrappers would be too complex for inclusion in our case study; (2) the complexity of the problem was also one of the reasons why we intelligently reduced the attribute space during feature selection; and (3) the dependency of wrappers on a specific learner makes it difficult to find the best suitable wrapper since there are many learners available to choose from.

4.1. Feature ranking techniques

A variety of feature ranking techniques have been developed. All these studies include single attribute classifiers [8]. The idea of this method is using a classifier (learner) with a single independent attribute (as well as a class attribute) to build a classification model and selecting attributes according to their individual predictive capability. The predictive capability of the attribute can be evaluated in terms of different performance metrics, such as accuracy, AUC (Area Under the ROC Curve), and F-measure [11]. This single attribute classifier method is also known as *wrapper-based feature ranking technique*. For this study, we used different feature ranking techniques in which no learners were involved. All these techniques, also called *filter-based feature ranking techniques*, will be introduced in the next following subsections.

4.1.1. CS method. CS can evaluate the worth of an attribute by computing the value of the CS statistic with respect to the class. The CS statistic (also denoted as χ^2) is a nonparametric statistical technique used to determine if a distribution of observed frequencies differs from the theoretical expected frequencies. CS statistics use nominal (categorical) or ordinal level data, thus instead of using means and variances, this test uses frequencies. The value of the CS statistic (χ^2) is given by [23]

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

where χ^2 is the test statistic that asymptotically approaches an χ^2 distribution, O_i is the observed frequency, and E_i is the expected frequency. n is the number of possible outcomes of each event.

There are two types of CS tests: (1) the test for goodness of fit which compares the expected and observed values to determine how well an experimenter's predictions fit the data and (2) the test for independence which compares two sets of categories to determine whether the two groups are distributed differently among the categories. The CS feature ranking technique takes advantage of the second case. The null hypothesis is that the two variables (for example, one independent attribute and one dependent attribute) are independent. The alternative hypothesis to be tested is that the two variables are dependent on each other. In WEKA[§] [11], by default, numeric attributes are discretized (using an MDL-based discretization scheme) before they are evaluated.

4.1.2. IG method. IG is able to assess the importance of an attribute by measuring the information gain with respect to the class. In general, the expected IG is the change in information entropy from a prior state to a state that takes some information. The gain is given by [11]

$$IG(Class, Attribute) = H(Class) - H(Class|Attribute)$$

where H specifies the entropy. More specifically, let A be the set of all attributes and $Class$ be the dependent attribute of all training examples, $value(a, y)$ with $y \in Class$ defines the value of a specific example for attribute $a \in A$, V represents the set of values of attribute a, i.e. $V = \{value(a, y) | a \in A \cap y \in Class\}$ and $|s|$ is the number of elements in the set s. The IG for an attribute $a \in A$ is defined as follows:

$$IG(Class, a) = H(Class) - \sum_{v \in V} \frac{|\{y \in Class | value(a, y) = v\}|}{|Class|} \times H(\{y \in Class | value(a, y) = v\})$$

[§]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.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

IG usually provides a good measure for deciding the relevance of an attribute, but it has some limitations. A notable problem is that it is biased in favor of features with more values. For example, suppose that we have some data describing a project's software modules. IG is often used to decide which of the attributes are the most relevant, so they can be used to estimate quality of software modules and make a test decision. One of the input attributes might be the software module's identification number (ID). This attribute has a high IG, because it uniquely identifies each module, but it does not make sense to treat a software module based on their ID. There are some strategies, such as IG ratio and SU, that can overcome this problem. They compensate for IG's lack of considering attributes with a large number of distinct values.

4.1.3. GR method. GR modifies the IG by taking into account the number of outcomes produced by the attribute test condition. It is defined as follows [11]:

$$GR(Class, a) = IG(Class, a)/H(a)$$

where

$$H(a) = -\sum_{v \in V} \frac{|\{y \in Class | value(a, y) = v\}|}{|Class|} \times \log_2 \left(\frac{|\{y \in Class | value(a, y) = v\}|}{|Class|} \right)$$

All the notations in the GR formula are the same as presented for IG.

4.1.4. SU method. SU also compensates for IG's bias toward attributes with more distinct values and normalizes its values to the range 0–1. The value 1 indicates that knowledge of attribute a completely predicts the value of *Class* and the value 0 indicates that *Class* and attribute a are independent. The SU measurement can be calculated by the following formula [11]:

$$SU(Class, a) = 2 \times \frac{IG(Class, a)}{H(Class) + H(a)}$$

Owing to the similarities of their equations, one might suspect that IG, GR, and SU would have very similar outcomes. However, our experimental results show differences among these techniques (see Section 5.2).

4.1.5. KS method. The KS method, as an attribute ranking technique, was recently proposed by our research team [9]. It utilizes the KS statistic to measure the maximum differences between the empirical distribution function of the attribute values 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 KS scores and be selected according to their KS scores and the number of attributes needed.

For the jth independent variable, the data are composed of two independent samples, *fp* and *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 $x_j^{(l)}$, $l = 1, \ldots, n_{nfp}$. $S_{X_j^{fp}}(x_j)$ is an empirical cumulative distribution function of the *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 $\{x_j^{(k)} | x_j^{(k)} \leq x_j,\ k = 1, \ldots, n_{fp}\}$.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

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 $\{x_j^{(l)} | x_j^{(l)} \leq x_j, l = 1, \ldots, n_{nfp}\}$.

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 KS score for attribute j.

4.1.6. Relief method. The RLF was first proposed by Kira and Rendell in 1992 [24]. The pseudocode of the algorithm is described in Figure 1 [25]. For a given instance R, Relief finds its nearest neighbor from the same and different class, called 'nearest hit H' and 'nearest miss M'. It updates the quality estimation W[A] for all the attributes A depending on their values for R, M, and H. The process is repeated m times, where m is specified by a user. Function *diff* (*Attribute*, *Instance1*, *Instance*2) are defined distinctly according to the different types of attributes. For discrete attributes, it is defined as

$$diff(A, I_1, I_2) = \begin{cases} 0 & value(A, I_1) = value(A, I_2) \\ 1 & otherwise \end{cases}$$

and for continuous attributes, it is defined as

$$diff(A, I_1, I_2) = \frac{|value(A, I_1) - value(A, I_2)|}{\max(A) - \min(A)}$$

The underlying hypothesis is that a relevant attribute is capable of distinguishing between instances from different classes and show no difference between instances from the same class. In the RLF implemented in WEKA, one parameter, *weightByDistance* (weight nearest neighbors by their distance), can be set to either *true* or *false*. In this study, we considered both cases. Therefore, we have two forms of Relief, one is Relief with *weightByDistance*=*false* (denoted as RF) and the other is Relief with *weightByDistance*=*true* (denoted as RT).

```
Relief Algorithm
Input:
   D: Training dataset (attribute values and the class value)
Output:
   The vector W (estimations of the qualities of attributes)
Method:
(1)  set all weights W[A] = 0;
(2)  for i = 1 to m do
(3)       randomly select an instance R;
(4)       find nearest hit H and nearest miss M;
(5)       for A = 1 to #of attributes do
(6)            W[A] = W[A] - diff(A,R,H)/m + diff(A,R,M)/m;
(7)       end for
(8)  end for
```

Figure 1. Relief algorithm.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

4.2. Feature subset selection techniques

The process of attribute subset selection consists of four basic steps [16]:

1. *Subset generation*. A candidate attribute subset is produced based on a certain search strategy. In this step, two problems need to be solved: where to start and how to produce subsequent candidate subsets. The start can be divided into *forward* (search starts with an empty set and then inserts attributes) and *backward* (search starts with a full set and then deletes attributes). For a given start approach, the search strategies are *ES* and *HS*. An ES examines all the attribute subsets of a given data set. It can guarantee finding the optimal result at the expense of $O(2^n)$ computational complexity, where n is the number of independent attributes in the data set. For example, the software measurement data sets in our study have 42 software metrics; hence, the search space is 2^{42}—which is prohibitively too large. Consequently, ES is typically used when the problem size is limited. In contrast, a HS can reduce the set of candidate solutions to a manageable size. We studied three search algorithms (in addition to not using any subset selection method after feature ranking), including ES, HS, and AHS, which was recently proposed by our research team [10].
2. *Subset evaluation*. During the search process, each generated subset needs to be assessed by an evaluation criterion. If the new subset turns out to be better, it substitutes the previous subset. Subset generation and evaluation is an iterative process until a stoping criterion is met. The filter model evaluates the subset of attributes by examining the intrinsic characteristic of the data without involving any learning algorithm. In contrast, the wrapper model evaluates the goodness of the subset of attributes by applying a predetermined learning algorithm on the selected subset of attributes. It tends to be computationally expensive. In this study, we used a modified version of consistency rate (CR) to measure the goodness of the selected attributes. This evaluation did not involve any learning algorithm, and therefore belongs to the filter category.
3. *Stopping criterion*. This determines when to stop the search algorithm. The stopping criteria vary with the different algorithms, and include criteria such as error threshold, size of attribute subset, maximum iterations, and run time.
4. *Result validation*. This is used to assess the effectiveness of an attribute selection method. Various performance metrics can be used to evaluate the prediction models before and after the attribute selection was made.

As presented above, subset evaluation is a critical step during the subset selection process. It can directly influence the selected subset of attributes, which, in turn, affects the quality of the classification models built on that selected subset. Numerous criteria can be used to determine goodness of the selected subset of attributes. One of the popular and widely used evaluation criteria is CR. In this study, for the three feature subset selection algorithms (ES, HS, and AHS), we used a modified version of CR as the evaluation criterion. The CR is computed using consistency count (the original approach uses inconsistency count). This modification was done for algorithm implementation purposes only [10]. CR has the *monotonic* property [26], which is described as follows:

- The complete attribute set (D) has the highest CR δ. In other words, the CR 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.

4.2.1. ES. ES generates every possible combination of feature subset and computes the respective CRs. A threshold is set up at the beginning according to the CR calculated for the first selected feature subset which is also set up as the best feature subset. As the algorithm proceeds, the current best subset may be replaced by one with the same or higher CR and fewer attributes. ES can

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

start with either a set with one feature and continue by adding features into the set or with a full feature set and then remove features from the set. It is obvious that ES is time-consuming and computationally expensive as it calculates every combination, many of which may be redundant or unnecessary. The efficiency deteriorates fast with the increase of the size of the search space. For example, if a given data set has n features, the number of combinations is 2^n which implies that there will be 2^n calculations for determining the CR. In conclusion, ES is inefficient and costly for a large number of features. An example of ES is Focus [26], which is implemented in this paper for comparison purposes.

4.2.2. HS. There are two fundamental goals for computing algorithms: finding a way to use less running time and producing an optimal solution. A heuristic algorithm (HS) is used when there is no known way to find an optimal solution or the optimal solution is too time-consuming in which case the goal of the heuristic is to develop a simple process with provable better running time and good solution. Since ES algorithms take significant amount of unnecessary time and are computationally costly, heuristic algorithms are good alternatives to complete quickly and return a decent result. There are many HS techniques in practice, such as best-first search [27], A* search [27], iterative deepening A* search [27], and SetCover [28]. The original idea for SetCover is that two instances with different class labels are said to be 'covered' when there exists at least one feature that has different values for the two instances [28]. In other words, two instances with two different class labels are considered to be consistent if two instances have at least one distinctive feature value between them. SetCover was implemented in this paper.

4.2.3. AHS. The AHS, an attribute subset selection method we recently proposed [10], also uses CR evaluation. The CR of the complete attribute set is computed first, and then starting with size 1 of any attribute, attribute subsets that have the locally highest CR are selected. These selected attribute subsets will be used to generate supersets. The process is repeated until finding the attribute subsets that have the same CR as the original data set with full features or the specified number of attributes is reached. The AHS algorithm is illustrated in Figure 2.

It should be noted that the ES, HS, and AHS subset selection search algorithms require that the input data be discretized, if the original data do not have discrete values for their attributes. In our study, we used the WEKA tool to discretize the data based on the equal frequency strategy [11]. The number of bins was set to 10 for the discretization process.

4.3. Classifiers

The five classifiers used in this case study are NB, MLP, SVM, LR, and KNN [29–33]. These are selected for their common use in the software engineering community, and for the fact that they do not have a built-in attribute selection capability. Generally, default settings of these learners are used as specified in WEKA [11]; however, changes to default parameters were made only in response to substantial improvement in classifier performance.

NB utilizes Bayes rule of conditional probability and is termed *naïve* because it assumes conditional independence of the features [32]. Prior research has shown that NB classifiers often perform well, even on real-world data where the variables are related [34].

MLPs [30] attempt to artificially mimic the functioning of a biological nervous system. Multiple nodes, or *neurons*, are connected in layers, with the output of each node being the thresholded weighted sum of its inputs from the previous layer. It has been shown that a multiple hidden layer neural network can approximate any function [35]. In our study, 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.

SVMs are a category of generalized linear classifiers [31] that map input vectors to a higher dimensional space where a maximal margin hyperplane is constructed. The data points or instances that are located on the margins of the maximum margin hyperplane are called support vectors.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

```
Algorithm: Automatic Hybrid Search (AHS)
Input:
   D, dataset
   S, full feature set of D
Output:
   L, consistent feature subsets
Method:
(1)  L = S
(2)  δ= CR(S, D)
(3)  T = all feature subset S' in S where |S'| = 1
(4)  max = -∞
(5)  while the size of any set in T < specified # of attributes
(6)     tempSet = δ
(7)     for each set T' in T
(8)         tempCR = CR(T', D)
(9)         if max < tempCR
(10)            max = tempCR
(11)            tempSet = T'
(12)        else if max = tempCR
(13)             tempSet = append(tempSet, T')
(14)        if max ≥ δ
(15)            L = tempSet
(16)            return L
(17)        else if |tempSet| = |T|
(18)            T = combinationSet(T, size + 1)
(19)        else
(20)            for any set tempSet' in tempSet
(21)                tempSet' = append (tempSet', f) where f is any feature
                    in S not in tempSet'
(22)            T = tempSet
(23) return L
```

Figure 2. AHS algorithm.

The sequential minimal optimization algorithm (SMO) developed by John Platt provides an effective method to train SVMs [36]. In our study, the complexity parameter, c, is set to '5.0', while the *buildLogisticModels* parameter is set to 'true' to produce proper probability estimates.

LR [33] is a statistical technique that can be used to solve binary classification problems. Based on the training data, an LR model is created which is used to decide the class membership of future instances. For our experiments, the default settings of WEKA are used.

KNN classifiers, also known as IBK (instance-based classifier) [29, 37], belong to the category of lazy learners. A case library stores all the examples of the training data, and when a classification is desired, the case library is used to obtain the KNN based on similarity with regard to the feature space. Subsequently, a decision rule, such as majority voting, can be used to classify new instances. In our study, *distanceWeighting* parameter was set to 'Weight by 1/distance', the *kNN* parameter was set to '30', and the *crossValidate* parameter was set to 'true'. In addition, we modified the learner 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 (TPR) and the true negative rate (TNR)).

4.4. Performance metric

In a binary (positive and negative[¶]) 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 V. The four values $|TP|$, $|TN|$, $|FP|$, and $|FN|$ 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, where $|\cdot|$ represents the number of instances in a given set.

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

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table V. Confusion matrix for a binary classification.

		Correct result	
		+	−
Obtained Result	+	TP	FP
	−	FN	TN

The four basic performance metrics, TPR, TNR, FPR (false positive rate), and FNR (false negative rate) are defined as follows:

$$TPR = \frac{|TP|}{|TP|+|FN|}, \quad TNR = \frac{|TN|}{|FP|+|TN|}$$

$$FPR = \frac{|FP|}{|FP|+|TN|}, \quad FNR = \frac{|FN|}{|TP|+|FN|}$$

These metrics are typically used in pairs when evaluating classifiers. For example, the FPR and the TPR are often considered simultaneously. The difficulty with this approach when comparing two classifiers, however, is that inconclusive results are often obtained. In other words, classifier A may have a higher TPR but a lower TNR than classifier B. The need to evaluate classifier performance with a single metric has led to the proposal of numerous alternatives. In this study, we used one of them, AUC.

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 graphs TPRs on the y-axis versus the FPRs on the x-axis as the decision threshold is varied. The resulting curve illustrates the trade-off between detection (TP) and false alarm (FP) rates [38]. Often, performance metrics for evaluation of classifiers consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. AUC is widely used for evaluating the predictive capability of classifiers in many application domains, including software engineering [2, 39–42]. 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.

5. EMPIRICAL CASE STUDIES

5.1. Experimental design

We first applied the seven feature ranking techniques to the fit data set, which contained all 42 software metrics, and obtained the respective rankings. We selected the top $\lceil \log_2 n \rceil$ attributes as the subsets of attributes, where n is the number of independent attributes in the original data set. The reasons that we selected the top $\lceil \log_2 n \rceil$ attributes (i.e. six metrics) include (1) the related literature lacks guidance on the number of features that should be selected when using a feature ranking technique; (2) our recent prior work showed that it was appropriate to use $\lceil \log_2 n \rceil$ features when using WEKA to build random forests learners for binary classification in general and imbalanced data sets in particular [43]. Although we used different learners in this study, a preliminary study showed that $\lceil \log_2 n \rceil$ is still a good choice for various learners; and (3) a software engineering expert with more than 20 years' experience recommended selecting $\lceil \log_2 n \rceil$ number of metrics for software systems such as our case study.

We also applied the three CR-based subset selection techniques, ES, HS, and AHS to the fit data set. As mentioned previously, ES enumerates all possible candidates for the solution and checks whether each candidate satisfies the problem's statement. The main disadvantage of ES is that,

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

for many real-world problems, the number of candidates may be prohibitively large. For example, the fit data set in this study contains 42 attributes, therefore, the number of all possible attribute subsets is 2^{42}. Evaluating all of these candidate subsets is impractical.

There are some strategies that can solve the above problem. One way is to reduce the search space, that is, the set of candidate solutions. In this experiment, we used the following method to reduce the search space. We first selected the 12 most relevant attributes from the complete set of attributes based on seven different feature ranking techniques. These selected subsets of attributes formed the new search spaces, each with 12 attributes. The new search space is significantly smaller than the original one. Subsequently, we applied ES, HS, and AHS to each smaller search space to obtain six-attribute subsets. As a result, we obtained 21 subsets of attributes. If we add the seven subsets of attributes directly obtained from the feature ranking techniques (the top six attributes based on each ranking), we have a total of 28 subsets of attributes produced by using seven ranking techniques combined with three search algorithms (ES, HS, and AHS) and NONE. The latter implies that no search algorithms were used, and the subsets of attributes were directly selected based on the rankings. To our knowledge, these are new hybrid techniques (combination of feature rankings and feature subset evaluations) that are presented and validated in this paper.

After the attribute selection, we used five learners, NB, MLP, SVM, LR, and KNN, to build classification models on the fit data set with various selected subsets of attributes. Then, we evaluated the trained models using the three independent test data sets. To examine whether the smaller subsets of attributes would downgrade the predictive performance of the classification models, we also applied the five learners to the original data sets which include 42 attributes and used the results as a baseline for the comparisons. We are interested in examining whether a reduced search space is harmful to the search result. Therefore, we produced the subset of attributes (with six features) using the AHS algorithm based on the original search space (42 attributes) and compared the outcome with the one using the smaller search spaces (12 attributes). The results based on the subset obtained when the HS algorithm is used on the original search space are the same as those of the AHS method, and therefore, are not presented. We note that since ES is practically infeasible, it is not used for the original vs reduced attributes' search comparison. All the different strategies used for feature selection in this study are presented in Figure 3 and the results are reported in terms of AUC, as shown in Table VI.

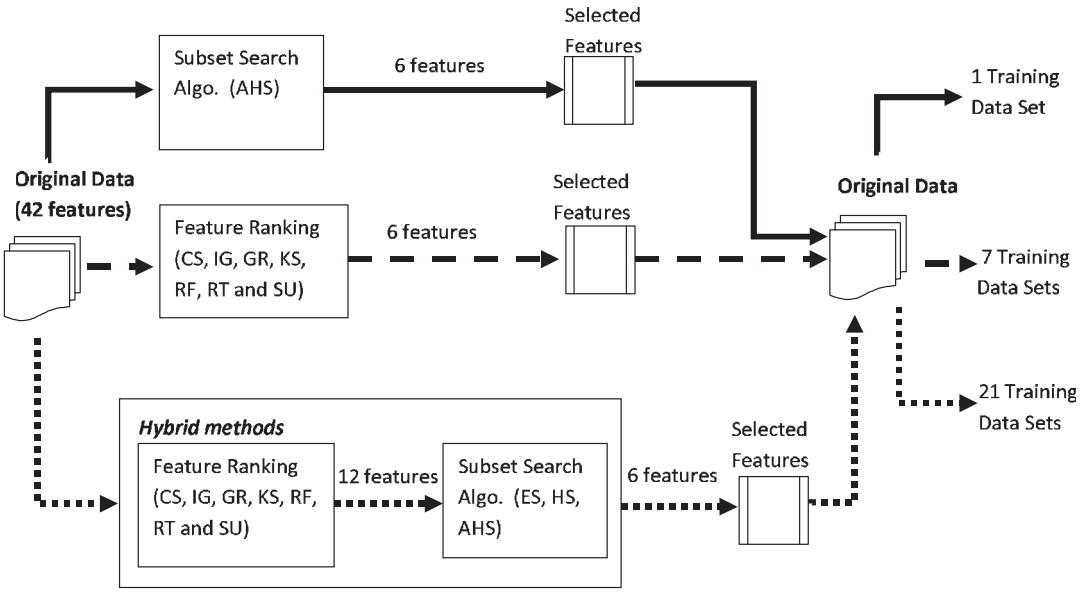

Figure 3. Feature selection strategies.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table VI. Performance of classification models in terms of AUC.

Ranking technique	Search algorithm	Learner	Fit	Test 1	Test 2	Test 3
CS	NONE	NB	0.7667	0.7850	0.7330	0.7697
		MLP	0.7441	0.7886	0.7584	0.7772
		SVM	0.6552	0.6030	0.5415	0.6012
		LR	0.7739	0.7936	0.7561	0.7786
		KNN	0.6955	0.6694	0.6741	0.7206
	ES	NB	0.7458	0.7489	0.7635	0.7302
		MLP	0.7356	0.7355	0.7642	0.7130
		SVM	0.6371	0.6469	0.5831	0.6011
		LR	0.7478	0.7447	0.7828	0.7186
		KNN	0.6984	0.7256	0.7423	0.7106
	HS*	NB	0.7761	0.7378	0.7522	0.7259
		MLP	0.7458	0.6966	0.7103	0.6727
		SVM	0.6297	0.7649	0.7211	0.6839
		LR	0.7696	0.7391	0.7661	0.7253
		KNN	0.7348	0.7289	0.7464	0.6939
	AHS*	NB	0.7761	0.7378	0.7522	0.7259
		MLP	0.7458	0.6966	0.7103	0.6727
		SVM	0.6297	0.7649	0.7211	0.6839
		LR	0.7696	0.7391	0.7661	0.7253
		KNN	0.7348	0.7289	0.7464	0.6939
GR	NONE	NB	0.8045	0.8275	0.8193	0.8113
		MLP	0.7959	0.8190	0.7982	0.8094
		SVM	0.6586	0.7214	0.7613	0.6980
		LR	0.7978	0.8176	0.8163	0.8081
		KNN	0.7651	0.7943	0.7976	0.7685
	ES	NB	0.7912	0.8114	0.8163	0.8214
		MLP	0.8000	0.8298	0.8380	0.8098
		SVM	0.6282	0.6951	0.6435	0.6016
		LR	0.8007	0.8279	0.8461	0.8150
		KNN	0.7474	0.8005	0.7868	0.7617
	HS	NB	0.8037	0.8181	0.8079	0.8257
		MLP	0.8042	0.8308	0.8364	0.8177
		SVM	0.6283	0.4890	0.4943	0.5795
		LR	0.8071	0.8294	0.8427	0.8265
		KNN	0.7610	0.7935	0.7560	0.7674
	AHS	NB	0.8037	0.8181	0.8079	0.8257
		MLP	0.8042	0.8308	0.8364	0.8177
		SVM	0.6283	0.4890	0.4943	0.5795
		LR	0.8071	0.8294	0.8427	0.8265
		KNN	0.7610	0.7935	0.7560	0.7674
IG	NONE	NB	0.7899	0.8119	0.8242	0.8208
		MLP	0.8029	0.8185	0.8494	0.8269
		SVM	0.6359	0.6182	0.6969	0.6431
		LR	0.8030	0.8178	0.8529	0.8361
		KNN	0.7371	0.7788	0.7551	0.7770
	ES	NB	0.7941	0.8164	0.8304	0.8218
		MLP	0.8034	0.8322	0.8432	0.8332
		SVM	0.6612	0.6966	0.7354	0.7344
		LR	0.8064	0.8303	0.8526	0.8360
		KNN	0.7450	0.7730	0.7666	0.8012
	HS	NB	0.7942	0.8158	0.8284	0.8225
		MLP	0.8062	0.8310	0.8504	0.8323
		SVM	0.6669	0.7934	0.7330	0.7319
		LR	0.8071	0.8310	0.8528	0.8374
		KNN	0.7521	0.7773	0.7832	0.7860
	AHS	NB	0.7942	0.8158	0.8284	0.8225
		MLP	0.8062	0.8310	0.8504	0.8323
		SVM	0.6669	0.7934	0.7330	0.7319

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table VI. *Continued*.

Ranking technique	Search algorithm	Learner	Fit	Test 1	Test 2	Test 3
		LR	0.8071	0.8310	0.8528	0.8374
		KNN	0.7521	0.7773	0.7832	0.7860
KS	NONE	NB	0.7896	0.8125	0.8225	0.8131
		MLP	0.8018	0.8159	0.8531	0.8258
		SVM	0.6939	0.8000	0.8373	0.8094
		LR	0.8033	0.8197	0.8477	0.8313
		KNN	0.7239	0.7717	0.7692	0.7845
	ES	NB	0.7938	0.8230	0.8151	0.8124
		MLP	0.8041	0.8241	0.8396	0.8208
		SVM	0.6606	0.6166	0.6496	0.5472
		LR	0.8054	0.8336	0.8489	0.8303
		KNN	0.7337	0.7808	0.7444	0.7872
	HS	NB	0.7938	0.8230	0.8151	0.8124
		MLP	0.8041	0.8241	0.8396	0.8208
		SVM	0.6606	0.6166	0.6496	0.5472
		LR	0.8054	0.8336	0.8489	0.8303
		KNN	0.7337	0.7808	0.7444	0.7872
	AHS	NB	0.7938	0.8230	0.8151	0.8124
		MLP	0.8041	0.8241	0.8396	0.8208
		SVM	0.6606	0.6166	0.6496	0.5472
		LR	0.8054	0.8336	0.8489	0.8303
		KNN	0.7337	0.7808	0.7444	0.7872
RF	NONE	NB	0.7907	0.7955	0.8440	0.8058
		MLP	0.8044	0.8116	0.8506	0.8158
		SVM	0.6340	0.6689	0.7039	0.6414
		LR	0.8120	0.8180	0.8522	0.8182
		KNN	0.7262	0.7180	0.7557	0.7306
	ES	NB	0.7932	0.8094	0.8489	0.7998
		MLP	0.8101	0.8116	0.8438	0.8065
		SVM	0.6499	0.6063	0.5910	0.5505
		LR	0.8183	0.8271	0.8537	0.8181
		KNN	0.7095	0.7515	0.7938	0.7322
	HS	NB	0.8089	0.8276	0.8347	0.8352
		MLP	0.8186	0.8217	0.8348	0.8214
		SVM	0.6434	0.6345	0.5345	0.6266
		LR	0.8229	0.8333	0.8392	0.8352
		KNN	0.7338	0.7473	0.7803	0.7477
	AHS	NB	0.8092	0.8174	0.8258	0.8303
		MLP	0.8198	0.8217	0.8414	0.8197
		SVM	0.6493	0.7776	0.8465	0.7532
		LR	0.8221	0.8338	0.8425	0.8354
		KNN	0.7269	0.7475	0.7727	0.7748
RT	NONE	NB	0.7930	0.7929	0.8317	0.7988
		MLP	0.8042	0.8072	0.8439	0.8103
		SVM	0.6599	0.5194	0.4278	0.4210
		LR	0.8113	0.8183	0.8481	0.8182
		KNN	0.7182	0.7038	0.7301	0.7323
	ES	NB	0.7932	0.8094	0.8489	0.7998
		MLP	0.8101	0.8116	0.8438	0.8065
		SVM	0.6499	0.6063	0.5910	0.5505
		LR	0.8183	0.8271	0.8537	0.8181
		KNN	0.7095	0.7515	0.7938	0.7322
	HS	NB	0.7723	0.7811	0.7898	0.7787
		MLP	0.7920	0.7854	0.7679	0.7855
		SVM	0.6385	0.6542	0.7061	0.7987
		LR	0.7958	0.7989	0.7811	0.7985
		KNN	0.7107	0.7548	0.7289	0.7467

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table VI. *Continued.*

Ranking technique	Search algorithm	Learner	Fit	Test 1	Test 2	Test 3
	AHS	NB	0.8060	0.8166	0.8203	0.8147
		MLP	0.8122	0.8258	0.8452	0.8218
		SVM	0.6328	0.7031	0.7443	0.6649
		LR	0.8195	0.8294	0.8443	0.8295
		KNN	0.7337	0.7328	0.7287	0.7699
SU	NONE	NB	0.8115	0.8261	0.8089	0.8266
		MLP	0.8101	0.8277	0.8397	0.8304
		SVM	0.6417	0.2820	0.2763	0.2948
		LR	0.8123	0.8275	0.8421	0.8335
		KNN	0.7665	0.7995	0.7464	0.7722
	ES	NB	0.7910	0.8234	0.8172	0.8169
		MLP	0.8048	0.8262	0.8358	0.8320
		SVM	0.6724	0.6783	0.6655	0.6770
		LR	0.8049	0.8330	0.8477	0.8291
		KNN	0.7404	0.7827	0.7596	0.7943
	HS	NB	0.8015	0.8309	0.8106	0.8226
		MLP	0.8084	0.8378	0.8415	0.8342
		SVM	0.6598	0.7697	0.8409	0.7608
		LR	0.8102	0.8367	0.8444	0.8357
		KNN	0.7459	0.7843	0.7572	0.7819
	AHS	NB	0.8015	0.8309	0.8106	0.8226
		MLP	0.8084	0.8378	0.8415	0.8342
		SVM	0.6598	0.7697	0.8409	0.7608
		LR	0.8102	0.8367	0.8444	0.8357
		KNN	0.7459	0.7843	0.7572	0.7819
NONE	AHS	NB	0.7840	0.7737	0.7774	0.7882
		MLP	0.7826	0.7981	0.7543	0.7707
		SVM	0.6327	0.5992	0.5489	0.4935
		LR	0.7915	0.7917	0.7733	0.7819
		KNN	0.7427	0.7361	0.6989	0.7418
NONE	NONE	NB	0.7925	0.8149	0.7963	0.8059
		MLP	0.8128	0.8314	0.8322	0.8309
		SVM	0.7451	0.6662	0.6519	0.6779
		LR	0.8182	0.8287	0.7989	0.8246
		KNN	0.7658	0.7849	0.8054	0.7901

*The average performance over the two distinct feature subsets produced by each of these techniques.

In the experiments, 10 runs of 10-fold cross validation were performed when we trained the classification models on the fit data set (release 1 data set). For each of the 10 folds, one fold is used as the test data while the other nine folds are used as training data. The training data are used to build the classification model and the resulting model is used on the test fold. This cross validation is repeated 10 times (the folds), with each fold used exactly once as the test data. The 10 results from the 10 folds can then be averaged to produce a single estimation. A total of 15 000 classification models were built in the experiment. We used the WEKA tool for conducting experiments with the six feature rankings (excluding KS) and for the software quality model building process. The KS feature ranking technique and the ES, HS, and AHS search algorithms are developed by our research team.

5.2. Results and analysis

Table VI lists the mean AUC value for every classifier (learner) constructed over 10 runs of 10-fold cross validation on the fit data set (the first data column) and predictive results on the test data sets (the next three data columns). Each value in the table is described by four dimensions, ranking

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

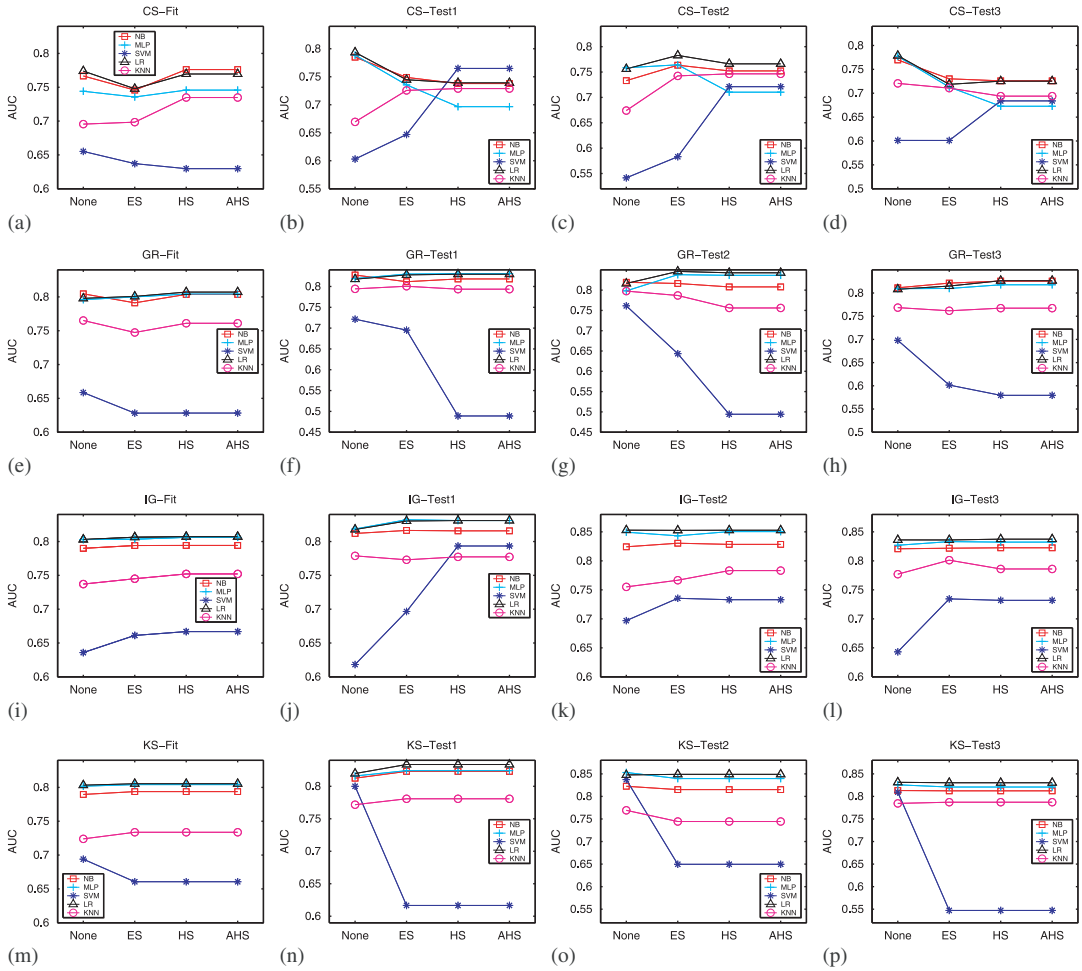

Figure 4. Performance of classification models in terms of AUC: (a) CS-Fit; (b) CS-Test 1; (c) CS-Test 2; (d) CS-TEST 3; (e) GR-Fit; (f) GR-Test 1; (g) GR-Test 2; (h) GR-TEST 3; (i) IG-Fit; (j) IG-Test 1; (k) IG-Test 2; (l) IG-TEST 3; (m) KS-Fit; (n) KS-Test 1; (o) KS-Test 2; and (p) KS-TEST 3.

technique, search algorithm, learner, and data set. For example, the first value in Table VI, 0.7667, represents the average of AUC over 10 runs of 10-fold cross validation on the fit data set when using the NB learner on the subset of attributes selected directly (i.e. the top six software metrics) by the CS feature ranking technique. Another example, in Table VI, 0.8114, which is located at the GR ranking technique category, ES subcategory, NB row, and Test1 column, represents the predictive result when the fitted model was applied to the Test 1 data set. The fitted model was trained on the fit data set with six attributes that were selected from the smaller search space (12 attributes) using the ES search algorithm. The smaller search space was constructed through the GR ranking technique. At the bottom of the table, we have some data listed in the 'NONE' ranking technique category and 'AHS' search algorithm category. This group of data represent the results on the subset of attributes when AHS was used on the original search space (42 attributes). Also, the data listed in the 'NONE' ranking technique category and 'NONE' search algorithm category represent the results on the original data sets. Note that some techniques produced more than one subset; in particular, HS and AHS used in conjunction with the CS ranking technique produced two results (subsets) each. For such cases, the average performance is presented. All these results except the last two (results using AHS on the original search space and results on the complete data sets) are mapped into a group of figures as shown in Figures 4 and 5. The figures

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

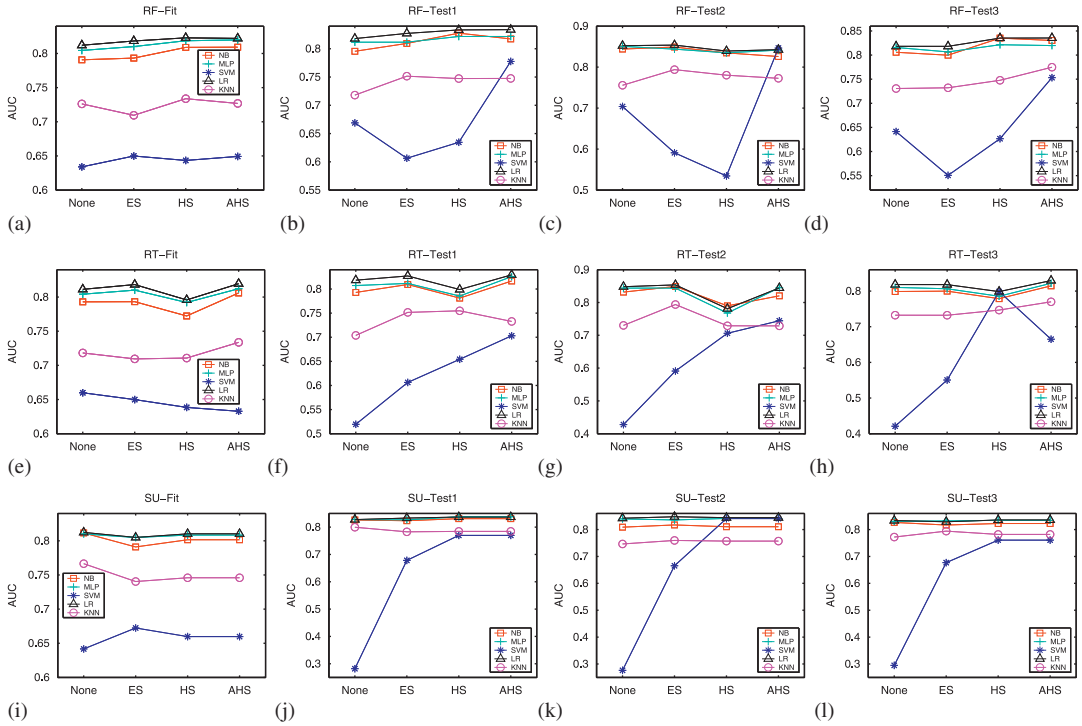

Figure 5. Performance of classification models in terms of AUC (cont.): (a) RF-Fit; (b) RF-Test 1; (c) RF-Test 2; (d) RF-TEST 3; (e) RT-Fit; (f) RT-Test 1; (g) RT-Test 2; (h) RT-TEST 3; (i) SU-Fit; (j) SU-Test 1; (k) SU-Test 2; and (l) SU-TEST 3.

present the classification performances of various techniques across different data sets. From these figures and the original table, we can observe the following facts:

1. Among the five learners, SVM performs the worst in terms of quality-of-fit across various subsets of attributes. Its predictive performance is also worse than the other four learners in most cases. In addition, the predictions are very unstable along the different search algorithms. The second worst learner is KNN, while the other three learners, NB, MLP, and LR show better results. Moreover, the predictions of NB, MLP, LR, and KNN are relatively consistent with their quality-of-fit.
2. Among the four different search algorithms, NONE, ES, HS, and AHS, there is no significant difference. But if we have to choose the best, AHS is the one. In addition, HS and AHS have the same results when combined with some ranking techniques such as CS, GR, IG, KS, and SU.
3. For the seven feature ranking techniques, the performance (in terms of AUC) of CS falls into the range of [0.65 0.8] across the four data sets (not including the results of SVM), while all the other ranking techniques' performance falls into the range of [0.7 0.85].
4. When we compared the quality-of-fit of various models with their predictive accuracy on the test data sets, we found that the predictive performance is similar to but in most cases better than the quality-of-fit. We also compared the results from the subsets of attributes with the results from the complete set of attributes; we found that the predictive performance is comparable and even better when a significant number of attributes were removed from the original data sets. This implies that the software quality classification and attribute selection were successfully applied in this study.

We also conducted a three-way ANalysis Of VAriance (ANOVA) F test [44] on the performance metric, AUC, to statistically examine the various effects on the performances of the classification models. An n-way ANOVA can be used to determine if the means in a set of data differ when

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

grouped by multiple factors. If they do differ, you can determine which factors or combinations of factors are associated with the difference. For example, the three-way ANOVA test in this study includes three factors: Factor A represents the results from the four subset selection strategies, NONE, ES, HS, and AHS. 'NONE' represents the results obtained directly from the rankings (no search algorithms are used). Factor B represents the results from seven feature ranking techniques, and Factor C represents the five learners. The interaction effects of two or three factors were also considered in the ANOVA test. Note that a total of 28 subsets of attributes were used in the ANOVA test (seven ranking techniques combined with four subset selection strategies). A three-factor, full factorial ANOVA model can be written as

$$y_{ijkl} = \mu + A_i + B_j + C_k + (AB)_{ij} + (AC)_{ik} + (BC)_{jk} + (ABC)_{ijk} + \varepsilon_{ijkl}$$

where

- y_{ijkl} represents the AUC of the lth observation for the ith level of A, jth level of B, and kth of C. Note that since 10 runs of 10-fold cross validation was used on the fit data set, $l = 1, 2, \ldots, 100$ for the fit results, while the generalization (prediction) results were obtained by applying the fitted models to the three test data sets, therefore $l = 1, 2, 3$ for the test results.
- μ is the overall mean performance (AUC).
- A, B, C are the three main factors or effects in the experiment, search algorithm, ranking technique, and learners, respectively.
- A_i, B_j, C_k are the treatment effects of the ith, jth, and kth levels of the experimental factors A, B, C, i is an index on the search algorithm, $i = 1, \ldots, 4$, j is an index on ranking technique, $j = 1, \ldots, 7$, and k is an index on the learner, $k = 1, \ldots, 5$.
- $(AB)_{ij}$, $(AC)_{ik}$, $(BC)_{jk}$ are two-way interaction terms between the main effects.
- $(ABC)_{ijk}$ is a three-way interaction term between the main effects.
- ε_{ijkl} is the random error.

The ANOVA model can be used to test the hypothesis that the AUC for the main factors A_i, B_j, C_k are equal against the alternative hypothesis that at least one mean is different. If the alternative hypothesis (i.e. that at least one mean is different) is accepted, numerous procedures can be used to determine which of the means are significantly different from the others. This involves the comparison of two means, with the null hypothesis that the means are equal. In this study, we performed the multiple comparison tests using Tukey's honestly significant difference criterion. All tests of statistical significance utilize a significance level α of 5%.

The ANOVA results are presented in Table VII. The two subtables are the ANOVA tests for the fit data set and test data sets (three test data sets are treated together), respectively.

- For the fit data set, the p-values (last column of Table VII) for all main factors and the interaction terms are less than a typical cutoff value of 0.05, indicating that the classification performances, in terms of AUC, are not the same for all groups in each factor or term. In other words, the classification performances are significantly different from each other for at least a pair of groups in the corresponding factors or terms.
- For the three test data sets, the p-values for all main factors as well as the interaction terms are much less than 0.05, indicating that the classification performances are not equal for all groups in each factor or term.

We further conducted multiple comparisons for three main factors to identify which pair(s) of means significantly differ from each other. The test results are shown in Figure 6, which includes six subfigures. Each subfigure displays graphs with each group mean represented by a symbol (∘) and an interval around the symbol (95% confidence interval). Two means are significantly different ($\alpha = 0.05$) if their intervals are disjoint, and are not significantly different if their intervals overlap. The results show the following points.

- For Factor A (subset selection strategy), a significant difference in the performance exists only between ES and AHS for the fit data set (AHS performed significantly better than ES).

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table VII. Analysis of variance.

Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
(a) ANOVA fit					
A	0.074	3	0.0246	5.30	0.0012
B	2.187	6	0.3646	78.63	0
C	47.295	4	11.8239	2550.19	0
$A \times B$	0.290	18	0.0161	3.48	0
$A \times C$	0.119	12	0.0099	2.14	0.0122
$B \times C$	1.325	24	0.0552	11.91	0
$A \times B \times C$	0.516	72	0.0072	1.55	0.0021
Error	64.261	13 860	0.0046		
Total	116.068	13 999			
(b) ANOVA test					
A	0.0247	3	0.0082	14.77	0
B	0.2810	6	0.0468	83.94	0
C	1.8223	4	0.4556	816.47	0
$A \times B$	0.1812	18	0.0101	18.04	0
$A \times C$	0.0904	12	0.0075	13.50	0
$B \times C$	0.1065	24	0.0044	7.95	0
$A \times B \times C$	0.7407	72	0.0103	18.44	0
Error	0.1562	280	0.0006		
Total	3.4031	419			

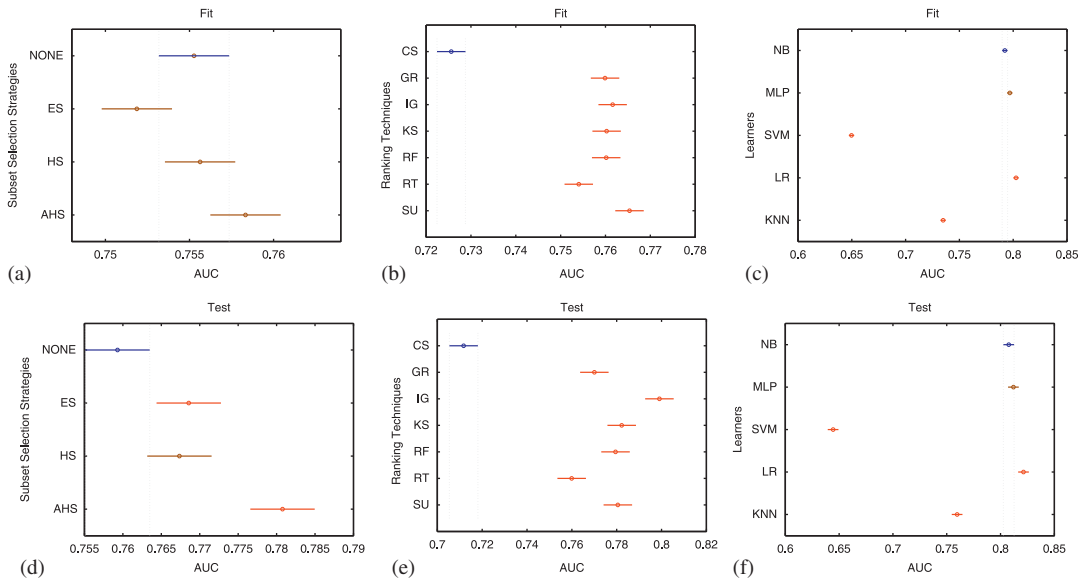

Figure 6. Multiple comparisons on three main factors: (a) Factor A—Fit; (b) Factor B—Fit; (c) Factor C—FIT; (d) Factor A—TEST; (e) Factor B—TEST; and (f) Factor C—TEST.

For the test results, AHS significantly outperformed ES, HS, and the direct rankings (NONE) alone. The predictive accuracy of the direct rankings (NONE) is lower than the three search algorithms.

It can be seen that ES can actually result in inferior classification depending on the learners, subset evaluation criterion, and performance metric adopted for classification evaluation. One may argue that since ES exhaustively considers all possible subsets, whatever HS or AHS selected must have been considered by ES too, so why does ES perform worse than HS or AHS? In this study, we used CR for the subset evaluation and AUC for the final classification evaluation. For this reason, the feature subsets selected by ES should have the same or better

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

CR compared to those selected by HS or AHS, but cannot guarantee to also produce better classification results in terms of AUC using a learner.

- For Factor B (feature ranking technique), CS performed significantly worse than other techniques for the fit data set and the test data sets. For the two forms of Relief, RF performed better than RT across all the data sets. SU demonstrated better quality-of-fit than the other techniques, while IG showed the best predictive capability among all the techniques.
- For Factor C (learner), SVM performed significantly worse than the other learners for the fit data set and the test data sets. The second worst leaner is KNN. NB and MLP showed similar performances across all the data sets. LR demonstrated the best quality-of-fit and predictive performance as well.

  A likely reason that SVM performed worse than the other learners could be that we used WEKA's implementation of SVM, which is just one version of the SVM learning algorithm. Other implementations of SVM should be considered in the future to see whether any improvements can be obtained.

As presented in Table VII (ANOVA), the classification performances, in terms of AUC, are also not the same for all groups in each interaction term. In this study, we have three two-way interaction terms, $A \times B$, $A \times C$, and $B \times C$ and a three-way interaction term, $A \times B \times C$. However, since the main concentration of this work is to investigate the various feature selection techniques, including direct feature rankings and combinations of feature rankings with the search algorithms, we would like to examine the classification performances from the 28 subsets of attributes based on the interaction term, $A \times B$, and ignore the other interaction terms. Also, we would like to compare the classification performances on the 28 subsets of attributes with the one on the complete set of attributes as well as the subset of attributes obtained from AHS on the original data set.

The results of multiple comparisons conducted on the 30 different subsets of attributes (including a complete set of attributes) are presented in Figure 7. Two subfigures are included, showing the results on the fit data set and the test data sets, respectively. The vertical axis lists the four search algorithms (denoted by 0 for NONE, E for ES, H for HS and A for AHS) combined with seven feature ranking techniques (denoted by CS, GR, IG, KS, RF, RT, and SU). For example, 'CS0' represents the result summarized over the five classifiers on the subset of attributes obtained when the CS ranking technique was used directly. Note that for the fit data set, the result is summarized over 500 classification models (5 classifiers×10 runs×10-fold cross validation = 500), while for

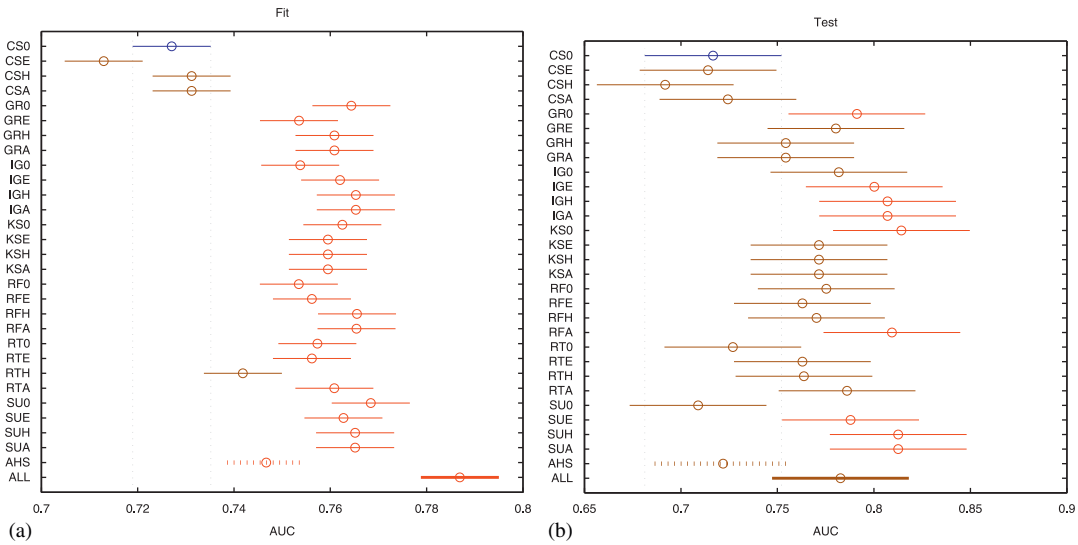

Figure 7. Multiple comparisons of classification models on 30 attribute sets: (a) Fit and (b) Test.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

the test data sets, the results are summarized over 15 classification models (5 classifiers×3 test data sets). Another example, 'KSA' means the results of five classifiers on the subset of attributes obtained when the AHS search algorithm applies to the search space created by the KS ranking technique. The last two labels, 'AHS' and 'ALL', represent the subset of attributes when AHS was used in the original search space (with 42 attributes) and a complete set of attributes, respectively. The 95% interval is highlighted by a dotted line for AHS and a thick line for ALL. From the figures, we can see the following facts:

- Any search algorithm when combined with CS ranking techniques showed neither good quality-of-fit nor good predictions. This is consistent with the conclusion we obtained when we previously studied main factor B.
- The RT0 and SU0 did not show good predictions although their quality-of-fit are reasonable. In contrast, RTH showed the moderate predictive performance although its quality-of-fit is worse than most other techniques.
- The classification models built on the complete set of attributes showed significantly better performance on the fit data set than those built on all the other subsets of attributes. However, its prediction is similar to or even worse than some of the subsets of attributes.
- The classification performance on the subset of attributes created by AHS on the original search space is only better than five other subsets of attributes (out 30 in total) and the prediction is even worse, only better than four other subsets of attributes. This is because AHS is a HS; it may not generate an optimal solution over the large feature space. However, when restricting the search space using a certain ranking technique, AHS can not only speed up the search process but can also maintain and even increase the quality of the search result.

In summary, seven feature ranking techniques, four subset selection strategies (three search algorithms plus a direct ranking), and their combinations were studied in this work. Among the three search algorithms, AHS showed better and more stable performances and therefore is recommended. For the seven feature ranking techniques, GR, IG, and KS when used alone outperformed the other ranking techniques and therefore are recommended. In addition, the search algorithms performed very well when the search space was restricted to a certain range, especially when the IG and SU ranking techniques were used to form the smaller search spaces. Finally, the experiments also demonstrated that the predictive capability of the classification models were comparable or even improved when a large number of attributes were eliminated from the original data set.

A summary of the best performing metric subsets using various techniques is provided in Tables VIII and IX. The first table reports all the selected metric subsets based on 29 techniques (four subset selection strategies used in conjunction with seven feature ranking techniques plus the AHS search algorithm used in the full search space with 42 attributes). Each column shows the result based on a given technique. For example, the column labeled 'CS' represents the CS ranking technique applied directly. Another example, the column labeled 'CS-ES' represents the selection technique that uses the ES search algorithm on the reduced search space created by the CS method. The selected attributes (metrics) are labeled 'X' in the table. Some techniques produced more than one result (subset), such as HS and AHS, when used in conjunction with the CS method, each including two results (subsets). We labeled them 1 and 2, respectively. The second table summarizes the software metrics' predictive powers in terms of their frequencies with respect to membership in all the selected feature subsets. With the numerous rankings obtained in this study, we observe a wide variability among the selected subsets produced by the different techniques. Some features or attributes are more frequent than others. For each attribute in the data sets, we record a count of its presence in the selected attribute subsets. It is assumed that the best attributes appear most in the selected subsets. Thus, the higher the frequency of an attribute in the selected subsets, the more useful it is in predicting the class label (fp or nfp). From the tables, we can see that the most frequently selected attributes are: number of distinct include files (FILINCUQ), number of different designers making changes (UNQ_DES), deployment percentage of the module (USAGE), base 2 logarithm of the number of independent paths (LGPATH), total

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table VIII. Selected metrics based on feature selection techniques.

	1 CS	2 CS-ES	3 CS-HS6-1	4 CS-HS6-2	5 CS-AHS6-1	6 CS-AHS6-2	7 GR	8 GR-ES	9 GR-HS	10 GR-AHS	11 IG	12 IG-ES	13 IG-HS	14 IG-AHS	15 KS	16 KS-ES	17 KS-HS	18 KS-AHS	19 RF	20 RF-ES	21 RF-HS	22 RF-AHS	23 RT	24 RT-ES	25 RT-HA	26 RT-AHS	27 SU	28 SU-ES	29 SU-HS	30 SU-AHS	31 AHS
1 CALUNQ																								X	X					
2 CAL2																														
3 CNDNOT																														
4 IFTH															X															
5 LOP																														
6 CNDSPNSM								X	X	X						X	X	X									X	X	X	X
7 CNDSPNMX																														
8 CTRNSTMX																					X	X		X	X					
9 KNT		X																												
10 NDSINT											X				X															
11 NDSENT																														
12 NDSEXT																														
13 NDSPND	X	X	X	X	X	X																								
14 LGPATH												X	X	X		X	X	X		X				X				X	X	X
15 FILINCUQ							X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
16 LOC																														
17 STMCTL												X																		
18 STMDEC								X	X	X																X	X	X	X	
19 STMEXE																														
20 VARGLBUS											X		X	X																
21 VARSPNSM							X	X			X				X											X	X			
22 VARSPNMX												X	X	X	X	X	X	X												
23 VARUSDUQ																X	X	X												
24 VARUSD2																														
25 DES_PR	X		X	X	X	X																								
26 BETA_PR	X		X	X	X	X	X																			X				
27 DES_FIX		X	X	X	X	X													X	X	X		X	X						
28 BETA_FIX	X																													
29 CUST_FIX	X						X		X	X											X	X				X		X	X	X
30 REQ_UPD																				X			X							
31 TOT_UPD											X	X	X	X	X															
32 REQ								X	X	X									X			X								
33 SRC_GRO	X	X	X		X																									
34 SRC_MOD				X		X																								X
35 UNQ_DES								X	X	X	X	X	X	X		X	X	X			X	X		X	X	X	X	X	X	X
36 VLO_UPD																			X			X								
37 LO_UPD		X																	X			X			X					
38 UPD_CAR																				X		X	X	X						X
39 USAGE		X	X	X	X	X													X	X	X	X	X	X	X					X
40 RESCPU							X																							
41 BUSCPU							X																							
42 TANCPU																														

span of branches of conditional arcs (CNDSPNSM), number of problems fixed that were found by designers in the prior release (DES_FIX), and number of problems fixed that were found by customers in the prior release (CUST_FIX).

Indeed, a large number of distinct include files in a software system can be problematic. One type of problem that may result from a large number of include files is the multiple inclusion problem. That is, one include file includes another, which includes the original include file. The hierarchies of these include files can become very complex, making it hard for software engineers to understand their inter-dependencies. Understandably, this characteristic of a software system is an excellent indicator of its quality as demonstrated by our method on the LLTS system. Likewise,

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

Table IX. Frequency of selected metrics.

		Frequency				Frequency	
ID	Metrics	#	%	ID	Metrics	#	%
15	FILINCUQ	23	74	34	SRC_MOD	3	10
35	UNQ_DES	19	61	1	CALUNQ	2	6
39	USAGE	14	45	10	NDSINT	2	6
14	LGPATH	11	35	30	REQ_UPD	2	6
6	CNDSPNSM	10	32	36	VLO_UPD	2	6
27	DES_FIX	10	32	4	IFTH	1	3
29	CUST_FIX	10	32	9	KNT	1	3
18	STMDEC	7	23	17	STMCTL	1	3
22	VARSPNMX	7	23	28	BETA_FIX	1	3
26	BETA_PR	7	23	40	RESCPU	1	3
13	NDSPND	6	19	41	BUSCPU	1	3
21	VARSPNSM	6	19	2	CAL2	0	0
38	UPD_CAR	6	19	3	CNDNOT	0	0
25	DES_PR	5	16	5	LOP	0	0
31	TOT_UPD	5	16	7	CNDSPNMX	0	0
32	REQ	5	16	11	NDSENT	0	0
8	CTRNSTMX	4	13	12	NDSEXT	0	0
33	SRC_GRO	4	13	16	LOC	0	0
37	LO_UPD	4	13	19	STMEXE	0	0
20	VARGLBUS	3	10	24	VARUSD2	0	0
23	VARUSDUQ	3	10	42	TANCPU	0	0

the number of different designers making changes is of high relevance. As practitioners, we know software is bound to change. In general, there are limitless possible causes of changes in software. Examples include but are not limited to feature enhancements and additions, bug fixes, etc. As software evolves, not only does the number of changes increase, but so does the number of different programmers changing the software. However, the more individuals there are making changes, the less likely it is that the design principles will be retained. This without a doubt will affect the software quality. Our method accurately identifies the deployment percentage of the module as a very relevant software metric for predicting quality. Basically, the higher the percentage of the module's deployment, the more likely the defects would be detected; in other words, if the module is used more, it is more likely that someone will run across bugs in that module. Our method also identifies two control flow graph metrics as relevant software metrics, the base 2 logarithm of the number of independent paths and the total span of branches of conditional arcs. Undoubtedly, the complexity of the structure of the program is a good indicator of its quality. Finally, practitioners can have an indication of the quality of a software system by examining the number of problems fixed that were found by designers and/or customers in the prior release. It is known that changes to the program (or bug fixes) often corrupt the original structure of the program and introduce some new defects (or bugs).

5.3. Threats to validity

A typical software development project is very human intensive, which can affect many areas of the development process including software quality and defect occurrence. Consequently, software engineering research that utilize 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 [45, 46]: (1) developed by a group, and not by an individual; (2) be as large as industry size projects, 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 system that is used in our case study was developed by professionals in a large software development organization using an established software development process and

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

management practices. The software was developed to address real-world problems in the telecommunications industry. We note that our case studies fulfill all of the above criteria specified by the software engineering community.

In an empirical software engineering effort, threats to external validity are conditions that limit the generalization of case study results. The analysis and conclusion presented in this paper are based upon the metrics and defect data obtained from multiple releases of a large telecommunications software. The same analysis for another software system, especially from a different application domain, may provide different results—a likely threat in all empirical software engineering research. However, our emphasis is more on the process of developing a hybrid feature selection process for selecting a good set of software metrics for defect prediction modeling. The process of feature ranking followed by feature subset selection search can be easily applied to software measurement data from a different project. Finally, all our final conclusions are based on statistical tests for significance.

The selection of certain classification algorithms in our study is unlikely to have much influence on the final conclusions, primarily because the attribute selection task is a preprocessing step that is done before training the prediction model. However, analysts should not rule out considering other classification algorithms when applying the proposed approach. The selection of specific parameter settings for a given classifier is likely to analyze the train data differently.

Threats to internal validity for a software engineering experiment are unaccounted influences that may affect case study results. In the context of this study, 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, which has been discussed earlier. In our comparative study, a common model-building and model-evaluation approach is used for all combinations of feature ranking techniques, feature subset selection search methods, and classifiers have been adopted. Moreover, the experiments and statistical analysis was performed by only one skilled person in order to keep modeling errors to a minimum.

A software engineering domain expert is consulted to set the number of software metrics to select from the original set after feature ranking. This number may be different for another project. However, based on our extensive prior work in software quality estimation and quantitative software engineering, we are confident that the selection of six metrics in the attribute subset is large enough to capture the quality-based characteristics of most software projects. We have found very often that only a handful of project-specific metrics are relevant for defect prediction, and that in many cases, very few metrics (among the available set) are selected by the learner in the final software quality prediction model.

6. CONCLUSION

Software metrics have a very important role in software development and software quality assurance activities. A software quality prediction model is often based on knowledge extracted from software measurement data. The quality of the underlying software measurement data is thus critical. An intelligent selection of software metrics before training a defect prediction model is likely to improve the end result—by removing redundant and less important features. We present an extensive investigation of feature selection for software quality modeling.

We studied seven filter-based feature ranking techniques and three filter-based subset selection search algorithms. Thus, each ranking technique is associated with each search algorithm. The search space for the subset selection algorithms is reduced for a more practical application of the proposed approach. A direct search approach is also evaluated, where the top six software metrics from a given ranking are selected as the feature subset. Software quality classification models are built subsequent to each feature selection process. We also look into the impact of a reduced search space on the feature subset selection outcome. A feature subset is obtained with the AHS search

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

algorithm on the original data (42 metrics), and then compared with the corresponding reduced search space outcomes.

The empirical results, based on a large real-world software system, demonstrate that among the different search algorithms, our AHS approach generally provides better performances. Among the seven feature ranking techniques, CS has consistently poor performances. It was also observed that reducing the attribute search space (for practical reasons) did not have an adverse effect on either subset selection or on classification results. This is important since a smaller search space lends to a more quicker overall software quality modeling process. Among the five learners, NB, MLP, and LR performed better than SVM and KNN. With the exception of SVM, all the learners we considered are very commonly used in the software engineering community. A final inference is that even after removing 85% of the available number of software metrics the defect prediction models were not adversely affected; in fact, in some cases the results were better. This is another important point for software practitioners, since practitioners prefer fewer software metrics from the data collection, management, modeling, and analysis points of view.

The future work will involve conducting additional empirical studies with data from other software projects and application domains. Object-oriented systems and object-oriented metrics will be considered in our future research. An investigation of wrapper-based feature selection will provide a new perspective on filter-based approaches for the software engineering community. In addition, modeling the cost or effort of the different feature selection techniques is an important topic that will be studied in the future.

ACKNOWLEDGEMENTS

The authors are grateful to the various members of the Data Mining and Machine Learning Laboratory at Florida Atlantic University for their assistance in reviewing this manuscript. In particular, we acknowledge the special efforts of our colleague, Dr. Amri Napolitano. We also extend our appreciation to the guest editors, Drs Simon Poulding and Iain Bate, of this special issue, and to the anonymous reviewers for their insightful comments.

REFERENCES

1. Khoshgoftaar TM, Bullard LA, Gao K. Attribute selection using rough sets in software quality classification. *International Journal of Reliability, Quality and Safety Engineering* 2009; **16**(1):73–89.
2. 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* 2008; **34**(4):485–496.
3. Meulen MJ, Revilla MA. Correlations between internal software metrics and software dependability in a large population of small C/C++ programs. *Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering, ISSRE 2007*, Trollhattan, Sweden, 2007; 203–208.
4. Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J. Detecting fault modules applying feature selection to classifiers. *Proceedings of Eighth IEEE International Conference on Information Reuse and Integration*, Las Vegas, Nevada, 2007; 667–672.
5. Sunghun K, Zimmermann T, Whitehead EJ, Zeller A. Predicting faults from cached history. *Proceedings of the 29th International Conference on Software Engineering, ICSE 2007*, Minneapolis, MN, 2007; 489–498.
6. Pfleeger SL. Software metrics: Progress after 25 years? *IEEE Software* 2008; **25**(6):32–34.
7. Hall MA, Holmes G. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering* 2003; **15**(6):1437–1447.
8. Guyon I, Elisseeff A. An introduction to variable and feature selection. *Journal of Machine Learning Research* 2003; **3**:1157–1182.
9. Khoshgoftaar TM, Nguyen L, Gao K, Rajeevalochanam J. Application of an attribute selection method to cbr-based software quality classification. *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence*. IEEE: Sacramento, CA, 2003; 47–52.
10. Lin P, Wang H, Khoshgoftaar TM. A novel hybrid search algorithm for feature selection. *Proceedings of 21st International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, 2009; 81–86.
11. Witten IH, Frank E. *Data Mining*: *Practical Machine Learning Tools and Techniques* (2nd edn). Morgan Kaufmann: Los Altos, CA, 2005.
12. Harman M, Mansouri SA, Zhang Y. Search based software engineering: A comprehensive review. Available at: http://www.sebase.org/sbse/publications/ [12 March 2009].
13. Harman M. The current state and future of search based software engineering. *Proceedings of FOSE'07: 2007 Future of Software Engineering*. IEEE Computer Society: Washington, DC, 2007; 342–357.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

14. Rodríguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J, Garre M. Attribute selection in software engineering datasets for detecting fault modules. *Proceedings of 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, Lübeck, Germany, 2007; 418–423.
15. Chen Z, Menzies T, Port D, Boehm B. Finding the right data for software cost modeling. *IEEE Software* 2005; **22**(6):38–46.
16. Liu H, Yu L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering* 2005; **17**(4):491–502.
17. Furlanello C, Serafini M, Merler S, Jurman G. Entropy-based gene ranking without selection bias for the predictive classification of microarray data. *BMC Bioinformatics* 2003; **4**:54.
18. Forman G. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research* 2003; **3**:1289–1305.
19. Doraisamy S, Golzari S, Norowi NM, Sulaiman N, Udzir NI. A study on feature selection and classification techniques for automatic genre classification of traditional malay music. *Ninth International Conference on Music Information Retrieval*, Philadelphia, PA, U.S.A., 2008; 331–336.
20. Jong K, Marchiori E, Sebag M, van der Vaart A. Feature selection in proteomic pattern data with support vector machines. *Proceedings of the 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, San Diego, CA, 2004.
21. Ilczuk G, Mlynarski R, Kargul W, Wakulicz-Deja A. New feature selection methods for qualification of the patients for cardiac pacemaker implantation. *Computers in Cardiology, 2007*, Durham, NC, U.S.A., 2007; 423–426.
22. Hudepohl JP, Aud SJ, Khoshgoftaar TM, Allen EB, Mayrand J. EMERALD: Software metrics and models on the desktop. *IEEE Software* 1996; **13**(5):56–60.
23. Cameron AC, Trivedi PK. *Regression Analysis of Count Data*. Cambridge University Press: Cambridge, 1998.
24. Kira K, Rendell LA. A practical approach to feature selection. *Proceedings of the Ninth International Workshop on Machine Learning*, Aberdeen, Scotland, U.K., 1992; 249–256.
25. Marko Robnik-Šikonja IK. An adaptation of relief for attribute estimation in regression. *Proceedings of the 14th International Conference on Machine Learning*, Nashville, Tennessee, 1997; 296–304.
26. Dash M, Liu H. Consistency-based search in feature selection. *Artificial Intelligence* 2003; **151**(1–2):151–176.
27. Coppin B. *Artificial Intelligence Illuminated*. Jones and Bartlett: Sudbury, MA, 2004.
28. Oliveira A, Vincentelli A. Constructive induction using a non-greedy strategy for feature selection. *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, 1992; 355–360.
29. Aha DW, Kibler D, Albert MK. Instance-based learning algorithms. *Machine Learning* 1991; **6**(1):37–66.
30. Haykin S. *Neural Networks: A Comprehensive Foundation* (2nd edn). Prentice-Hall: Englewood Cliffs, NJ, 1998.
31. Shawe-Taylor J, Cristianini N. *Support Vector Machines* (2nd edn). Cambridge University Press: Cambridge, 2000.
32. John GH, Langley P. Estimating continuous distributions in Bayesian classifiers. *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, San Mateo, vol. 2, 1995; 338–345.
33. Le Cessie S, Van Houwelingen JC. Ridge estimators in logistic regression. *Applied Statistics* 1992; **41**(1):191–201.
34. Domingos P, Pazzani M. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 1997; **29**(2–3):103–130.
35. Sokolova M, Japkowicz N, Szpakowicz S. Beyond accuracy, f-score and ROC: A family of discriminant measures for performance evaluation. *Proceedings of the Australian Conference on Artificial Intelligence*, Hobart, Australia, 2006; 1015–1021.
36. Platt JC. Advances in kernel methods—Support vector learning. *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. MIT Press: Cambridge, MA, 1999; 185–208.
37. Aha DW. *Lazy learning*. Kluwer Academic Publishers: Norwell, MA, U.S.A., 1997.
38. Fawcett T. An introduction to ROC analysis. *Pattern Recognition Letters* 2006; **27**(8):861–874.
39. Fogarty J, Baker RS, Hudson SE. Case studies in the use of roc curve analysis for sensor-based estimates in human computer interaction. *Proceedings of Graphics Interface 2005*. The Canadian Human-Computer Communications Society: Victoria, BC, Canada, 2005; 129–136.
40. Tan PN, Steinbach M, Kumar V. *Introduction to Data Mining*. Addison-Wesley: Reading, MA, 2006.
41. Jiang Y, Lin J, Cukic B, Menzies T. Variance analysis in software fault prediction models. *Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering*, Bangalore, Mysore, India, 2009; 99–108.
42. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering* 2007; **33**(1):2–13.
43. Khoshgoftaar TM, Golawala M, Van Hulse J. An empirical study of learning from imbalanced data using random forest. *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2. IEEE Computer Society: Washington, DC, U.S.A., 2007; 310–317.
44. Berenson ML, Goldstein M, Levine D. *Intermediate Statistical Methods and Applications: A Computer Package Approach* (2nd edn). Prentice-Hall: Englewood Cliffs, NJ, 1983.
45. Wohlin C, Runeson P, Host M, Ohlsson MC, Regnell B, Wesslen A. *Experimentation in Software Engineering*: *An Introduction*, Kluwer International Series in Software Engineering. Kluwer Academic Publishers: Boston, MA, 2000.
46. Votta LG, Porter AA. Experimental software engineering: A report on the state of the art. *Proceedings of the 17th International Conference on Software Engineering*. IEEE Computer Society: Seattle, WA, U.S.A., 1995; 277–279.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/spe