**Noname manuscript No.**
(will be inserted by the editor)

# Towards benchmarking feature subset selection methods for software fault prediction

**Wasif Afzal · Richard Torkar**

Received: date / Accepted: date

**Abstract** Despite the general acceptance that software engineering datasets often contain noisy, irrelevant or redundant variables, very few benchmark studies of feature subset selection (FSS) methods on real-life data from software projects have been conducted. This paper provides an empirical comparison of state-of-the-art FSS methods: information gain attribute ranking (IG); Relief (RLF); principal component analysis (PCA); correlation-based feature selection (CFS); consistency-based subset evaluation (CNS); wrapper subset evaluation (WRP); and an evolutionary computation method, genetic programming (GP), on five fault prediction datasets from the PROMISE data repository. For all the datasets, the area under the receiver operating characteristic curve—the AUC value averaged over 10-fold cross-validation runs—was calculated for each FSS method-dataset combination before and after FSS.

Two diverse learning algorithms, C4.5 and naïve Bayes (NB) are used to test the attribute sets given by each FSS method. The results show that although there are no statistically significant differences between the AUC values for the different FSS methods for both C4.5 and NB, a smaller set of FSS methods (IG, RLF, GP) consistently select fewer attributes without degrading classification accuracy. We conclude that in general, FSS is beneficial as it helps improve classification accuracy of NB and C4.5. There is no single best FSS method for all datasets but IG, RLF and GP consistently select fewer attributes without degrading classification accuracy within statistically significant boundaries.

## 1 Introduction

A bulk of literature on prediction and estimation in software engineering contributes to software fault/defect prediction (also termed as software quality classification/software quality modeling). Software fault prediction research uses software metrics to predict the response variable which can either be the class of a module (e.g., fault-prone and not fault-prone) or a quality factor (e.g., number of faults) for a module (Khoshgoftaar and Seliya, 2004). This paper is concerned with classifying software components/modules as fault-prone and not fault-prone (Lessmann et al, 2008), (Hall et al, 2011), (Catal and Diri, 2009b). Such a classification task is useful for the following reasons:

- Knowing which software components are likely to be fault-prone supports better targeting of software testing effort. This in turn has the potential to improve test efficiency and effectiveness.
- Fault-prone software components are candidates of refactoring whereby their internal structure can be improved.

W. Afzal
School of Innovation, Design & Engineering
Mälardalen University, Västerås, Sweden
E-mail: wasif.afzal@mdh.se

R. Torkar
Blekinge Institute of Technology, Karlskrona, Sweden. &
Chalmers University of Technology — University of Gothenburg, Gothenburg, Sweden
E-mail: richard.torkar@cse.gu.se

Despite the presence of a large number of models for software fault prediction, there is lack of a definitive advice on what prediction models are useful under different contexts. In order to increase confidence in the results of software fault prediction studies, more and more research is focussing on the need for a robust process and methodology to build prediction models (Song et al, 2011; Fenton and Neil, 1999; Hall et al, 2011). Central to such a methodology are issues such as data quality, measurement of predictive performance (Foss et al, 2003), resampling methods to use (Afzal et al, 2012) and reporting of fault prediction experiments (Hall et al, 2011). For data quality, important issues are data preprocessing (Gray et al, 2011), class imbalance (Khoshgoftaar et al, 2010; Shivaji et al, 2009) and impact of feature subset selection (FSS) methods (Rodriguez et al, 2007a,b). This paper contributes to the last aspect of data quality: use of FSS methods in software fault prediction.

The purpose of FSS is to find a subset of the original features of a dataset, such that an induction algorithm that is run on data containing only these features generates a classifier with the highest possible accuracy (Kohavi and John, 1997). There are several reasons to keep the number of features in a data set as small as possible:

1. Reducing the number of features allows classification algorithms to operate faster, more effectively (Hall and Holmes, 2003) and with greater simplicity (Jain et al, 2000).
2. Smaller number of features help reduce the curse of dimensionality[1].
3. Smaller number of features reduce measurement cost as less data needs to be collected (Chen et al, 2005a).
4. FSS helps to achieve a better understandable model and simplifies the usage of various visualization techniques (Janecek et al, 2008).

The simplest approach to FSS would require examining all possible subsets of the desired number of features in the selected subset and then selecting the subset with the smallest classification error. However, this leads to a combinatorial explosion, making exhaustive search all but impractical for most of the data sets (Jain et al, 2000). Naturally many FSS methods are search-based (Burke and Kendall, 2005), combined with an attribute utility estimator to evaluate the relative merit of alternate subsets of attributes (Hall and Holmes, 2003).

Several researchers in software engineering have emphasized the need to investigate only relevant variables. According to Dybå et al. (Dybå et al, 2006): "*Careful selection of which independent variables to include and which variables to exclude, is, thus, crucial to raising the power of a study and the legitimacy of its potential findings*". Further emphasizing the importance of FSS, Song et al. (Song et al, 2011) argue that "*[. . . ] before building prediction models, we should choose the combination of all three of learning algorithm, data pre-processing and attribute selection method, not merely one or two of them*". It is also generally accepted that software engineering data sets often contain noisy, irrelevant, or redundant variables (Chen et al, 2005a; Afzal et al, 2009), therefore, it is important to evaluate FSS methods for software engineering data sets. In literature, there are few studies that compare FSS methods for software fault prediction (Section 2) but no benchmark study on commonly used FSS methods on real-life public data from software projects has been conducted. Moreover, the use of evolutionary algorithms (e.g., genetic algorithm, genetic programming) have sporadically been investigated as FSS methods (Vivanco et al, 2010; Smith and Bull, 2003; Muni et al, 2006; Yang and Honavar, 1998) but not to an extent of comparing with state-of-the-art FSS methods, using publicly available real-life data from software projects.

This paper provides an empirical comparison of the state-of-the-art FSS methods and an evolutionary computation method (genetic programming (GP)) on five software fault prediction data sets from the PROMISE data repository (Boetticher et al, 2007). Two diverse learning algorithms, C4.5 and naïve Bayes (NB), are used to test the attribute sets given by each FSS method. We are interested in investigating if the classification accuracy of C4.5 and NB significantly differ before and after the application of FSS methods. In order to formalize the purpose of the empirical study, we set forth the following hypotheses to test:

$H_0$: The classification accuracy of C4.5 and NB is not significantly different before and after applying the FSS methods, i.e., $ACC_{C4.5} = ACC_{NB}$.

$H_1$: The classification accuracy of C4.5 and NB is significantly different before and after applying the FSS methods, i.e., $ACC_{C4.5} \neq ACC_{NB}$.

[1] The requirement that the number of training data points to be an exponential function of the feature dimension.

The results of this study indicate that FSS is generally useful for software fault prediction using NB and C4.5. However there are no clear winners for either of the two learning algorithms for the variety of FSS methods used. Based on individual accuracy values, RLF, IG and GP are the best FSS methods for software fault prediction accuracy while CNS and CFS are also good overall performers.

The paper is organized as follows. The next Section describes related work. Section 3 describes the FSS methods used in this study while Section 4 describes the datasets used, the evaluation measure and the experimental setup of the replication study. Section 5 presents the results of the empirical study and presents a discussion. Validity evaluation is given in Section 6 while the paper is concluded in Section 7.

## 2 Related work

Molina et al. (Molina et al, 2002), Guyon and Elisseeff (Guyon and Elisseeff, 2003), Blum and Langley (Blum and Langley, 1997), Dash and Liu (Dash and Liu, 1997) and Liu and Yu (Liu and Yu, 2005) provide good surveys reviewing work in machine learning on FSS. This section will, however, summarize the work done on FSS in predictive modeling within software engineering. FSS techniques in software engineering have been applied for software cost/effort estimation and software quality classification (also called as software defect/fault prediction). As the below paragraphs would illustrate, there is no definitive guidance available on FSS techniques to use in software engineering predictive modeling.

Dejaeger et al. (Dejaeger et al, 2012) used a generic backward input selection wrapper for FSS and reported significantly improved performance for software cost modeling in comparison to when no FSS was used. Similar results were reported by Chen et al. (Chen et al, 2005a,b). They showed that using wrapper improves software cost prediction accuracy and is further enhanced when used in combination with row pruning. However, for the COCOMO-styled datasets used in a study by Menzies et al. (Menzies et al, 2010) for estimating software effort/cost, wrapper FSS technique did not improve the estimation accuracy. Kirsopp et al. (Kirsopp et al, 2002) used random seeding, hill climbing and forward sequential selection to search for optimal feature subsets for predicting software project effort. They showed that hill climbing and forward sequential selection produce better results than random searching. Azzeh et al. (Azzeh et al, 2008), on the other hand, showed that their proposed fuzzy FSS algorithm consistently outperforms hill climbing, forward subset selection and backward subset selection for software effort estimation. Li et al. (Li et al, 2009) showed that a hybrid of wrapper and filter FSS techniques known as mutual information based feature selection (MICBR) can select more meaningful features while the performance was comparable to exhaustive search, hill climbing and forward sequential selection.

Menzies et al (Menzies et al, 2007) showed that there are no clear winners in FSS techniques for learning defect predictors for software fault/defect prediction. They compared information gain, correlation-based feature selection, relief and consistency based subset evaluation. Rodriguez et al. (Rodriguez et al, 2007a,b) also compared filter and wrapper FSS techniques for predicting faulty modules. They, however, concluded that wrapper FSS techniques have better accuracy than filter FSS techniques. Song et al. (Song et al, 2011) used wrapper FSS with forward selection and backward elimination search strategies for defect proneness prediction. They showed that different attribute selectors are suitable to different learning algorithms. Catal and Diri (Catal and Diri, 2009a) applied correlation-based FSS method on class-level and method-level metrics for software fault prediction. They showed that random forests gives the best results when using this FSS method. Khoshgoftaar et al. (Khoshgoftaar et al, 2006) found that the use of a stepwise regression model and a correlation-based FSS with greedy forward search did not yield improved predictions. Wang et al. (Wang et al, 2009) compared seven filter based FSS techniques and proposed their own combination of filter-based and consistency-based FSS algorithm. Their proposed algorithm and the Kolmogorov-Smirnov technique performed competitively with other FSS techniques. Koshgoftaar et al. (Khoshgoftaar et al, 2003) also showed better results with a FSS method based on the Kolmogorov-Smirnov two-sample statistical test. Altidor et al. (Altidor et al, 2010) compared their new wrapper FSS algorithm against 3-fold cross-validation, 3-fold cross-validation risk impact and a combination of the two. They showed that the performance of their new FSS technique is dependent on the base classifier (ranker aid), the performance metric and the methodology. Gao et al. (Gao et al, 2012) concluded that data sampling followed by wrapper FSS technique improves the accuracy of predicting high-risk program modules. Gao et al. (Gao et al, 2011) compared seven feature ranking techniques and four FSS techniques. Their proposed automatic hybrid search performed best among FSS techniques. Khoshgoftaar et al. (Khoshgoftaar et al, 2012) compared seven filter-based feature ranking techniques,

including a signal-to-noise (SNR) technique. SNR performed as well as the best performer of the six commonly used techniques. Wang et al. (Wang et al, 2012) compared several ensemble FSS techniques and concluded that although there are no clear winners but ensembles of few rankers are effective then ensembles of many rankers. Khoshgoftaar et al. (Khoshgoftaar et al, 2010) investigated the relation between six filter-based FSS methods with random under-sampling technique. They concluded that FSS based on sampled data resulted in significantly better performance than FSS based on original data.

## 3 Feature subset selection (FSS) methods

There are two commonly known categories of FSS methods: the filter approach and the wrapper approach. In the filter approach, the feature selection takes place independently of the learning algorithm and is based only on the data characteristics. The wrapper approach, on the other hand, conducts a search for a good subset using the learning algorithm itself as part of the evaluation function (Kohavi and John, 1997). Hall and Holmes (Hall and Holmes, 2003) provides another categorization for FSS methods, namely, those methods that evaluate individual attributes and those that evaluate subset of attributes.

We have chosen to empirically evaluate a total of seven FSS methods, two that evaluate individual attributes (information gain attribute ranking and Relief), three that evaluate subsets of attributes (correlation-based feature selection, consistency-based subset evaluation, wrapper subset evaluation), one classical statistical method for dimensionality reduction (principal components analysis) and one evolutionary computational method (genetic programming). Following is a brief description of the FSS methods used in this study.

### 3.1 Information gain (IG) attribute ranking

The foundation of IG attribute ranking is the concept of entropy which is considered as a measure of system's unpredictability. If $C$ is the class, the entropy of $C$ is given by:

$$H(C) = -\sum p(c) \log p(c)$$

where $p(c)$ is the marginal probability density function for class $C$. If the observed values of $C$ are partitioned based on an attribute $A$ and the entropy of $C$ after observing the attribute is less than the entropy of $C$ prior to it, there is a relationship between $C$ and $A$. The entropy of $C$ after observing $A$ is:

$$H(C|A) = -\sum_{a \epsilon A} p(a) \sum_{c \epsilon C} p(c|a) \log p(c|a)$$

where $p(c|a)$ is the conditional probability of $c$ given $a$.

Given that entropy is a measure of system's unpredictability, information gain is the amount by which the entropy of $C$ decreases (Quinlan, 1993). It is given by:

$$IG = H(C) - H(C|A) = H(A) - H(A|C)$$

IG is a symmetrical measure meaning that information gained about $C$ after observing $A$ is equal to the information gained about $A$ after observing $C$.

IG attribute ranking is one of the simplest and fastest attribute ranking methods (Hall and Holmes, 2003) but its weakness is that it is biased in favor of attributes with more instances even when they are not more informative (Novakovic, 2009).

In this study, IG attribute ranking is used with the ranker search method that ranks attributes by their individual evaluations.

### 3.2 Relief (RLF)

Relief is an instance-based attribute raking algorithm proposed by Kira and Rendell (Kira and Rendell, 1992). The Relief algorithm estimates the quality of attributes according to how they differentiate between

instances from different classes that are near to each other. So given a randomly selected instance $R$, Relief searches for a nearest hit $H$ (a nearest neighbor from the same class) and a nearest miss $M$ (a nearest neighbor from a different class). It then updates the relevance score for attributes depending on their values for $R$, $M$ and $H$. The process is repeated for a user-defined number of instances $m$. The basic Relief algorithm, taken from (Sikonja and Kononenko, 1997), is given in Figure 1.

```
Algorithm Relief
Input: for each training instance a vector of attribute values and the class value
Output: the vector W of estimations of the qualities of attributes

1.  set all weights W[A] := 0.0;
2.  for i := 1 to m do begin
3.      randomly select an instance R;
4.      find nearest hit H and nearest miss M;
5.      for A := 1 to #all_attributes do
6.          W[A] := W[A] - diff(A,R,H)/m + diff(A,R,M)/m;
7.  end;
```

**Fig. 1** The basic Relief algorithm.

The function $diff(Attribute, Instance1, Instance2)$ calculates the difference between the values of attribute for two instances. For discrete attributes, the difference is either 1 (the values are different) or 0 (the values are the same). For continuous attributes the difference is the actual difference normalized to the interval [0,1] (Hall and Holmes, 2003).

In this study the Relief method is used with the ranker search method that ranks attributes by their individual evaluations and the value of $m$ is set to 250 which is a recommended figure (Hall and Holmes, 2003).

### 3.3 Principal component analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique that transforms a set of possibly correlated variables into a set of linearly uncorrelated variables. These linearly uncorrelated variables are called principal components. The transformation is done by first computing the covariance matrix of the original variables and then finding its Eigen vectors (principal components). The principal components have the property that most of their information content is stored in the first few features so that remainder can be discarded. In this study, PCA is used with the ranker search method that ranks attributes by their individual evaluations.

### 3.4 Correlation-based feature selection (CFS)

Correlation-based feature selection (CFS) evaluates subsets of attributes rather than individual attributes (Hall, 2000). The technique uses a heuristic to evaluate subset of attributes. The heuristic balances how predictive a group of features are and how much redundancy is among them.

$$Merit_s = \frac{kr_{cf}}{\sqrt{k+k(k-1)r_{ff}}}$$

where $Merit_s$ is the heuristic merit of a feature subset $s$ containing $k$ features, $r_{cf}$ is the average feature-class correlation and $r_{ff}$ is the average feature-feature intercorrelation (Hall and Holmes, 2003). In order to apply $Merit_s$, a correlation matrix has to be calculated and a heuristic search to find a good subset of features. In this study, CFS is used with the Greedy stepwise forward search through the space of attribute subsets.

### 3.5 Consistency-based subset evaluation (CNS)

Consistency-based subset evaluation (CNS) is also an attribute subset selection technique that uses class consistency as an evaluation metric (Liu and Setiono, 1996). CNS looks for combinations of attributes

whose values divide the data into subsets containing a strong single class majority (Hall and Holmes, 2003). Liu and Setiono (Liu and Setiono, 1996) proposed the following consistency metric:

$$Consistency_s = 1 - \frac{\sum_{i=0}^{j} |D_i| - |Mi|}{N}$$

where $s$ is an attribute subset, $j$ is the number of distinct combinations of attribute values for $s$, $|D_i|$ is the number of occurrences of the $ith$ attribute value combination, $|M_i|$ is the cardinality of the majority class for the $ith$ attribute value combination and $N$ is the total number of instances in the data set.

In this study, greedy stepwise forward search is used to produce a set of attributes, ranked according to their overall contribution to the consistency of the attribute set (Hall and Holmes, 2003).

### 3.6 Wrapper subset evaluation (WRP)

The wrapper feature subset evaluation conducts a search for a good subset using the learning algorithm itself as part of the evaluation function. In this study, repeated five-fold cross-validation is used as an estimate for the accuracy of the classifier while a greedy stepwise forward search is used to produce a list of attributes, ranked according to their overall contribution to the accuracy of the attribute set with respect to the target learning algorithm (Hall and Holmes, 2003).

### 3.7 Genetic programming (GP)

Genetic programming (GP) is an evolutionary computation technique and is an extension of genetic algorithms. It is a "*systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done*" (Poli et al, 2008). GP applies iterative, random variation to an existing pool of computer programs to form a new generation of programs by applying analogs of naturally occurring genetic operators (Burke and Kendall, 2005). The basic steps in a GP system are given below (Poli et al, 2008):

1. Randomly create an initial population of programs.
2. Repeat (until stopping criterion is reached):
   (a) Execute each program and evaluate its fitness.
   (b) Select one or two programs to undergo genetic operations.
   (c) Create new programs by applying the genetic operations.
3. Return the best individual.

As compared with genetic algorithms, the population structures (individuals) in GP are not fixed length character strings, but programs that, when executed, are the candidate solutions to the problem.

The evolution of software fault prediction models using GP is an example of a symbolic regression problem. Symbolic regression is an error-driven evolution as it aims to find a function, in symbolic form, that fits (or approximately fits) data from an unknown curve (Koza, 1992). In simpler terms, symbolic regression finds a function whose output matches some target values. GP is well suited for symbolic regression problems, as it does not make any assumptions about the structure of the function.

Programs are expressed in GP as syntax trees, with the nodes indicating the instructions to execute and are called functions (e.g., $min$, $*$, $+$, $/$), while the tree leaves are called terminals which may consist of independent variables of the problem and random constants (e.g., $x$, $y$, 3). The fitness evaluation of a particular individual is determined by the correctness of the logical output produced for all of the fitness cases (Bäck et al, 2000). The fitness function guides the search in promising areas of the search space and is a way of communicating a problem's requirements to the GP algorithm. The control parameters limit and control how the search is performed like setting the population size and probabilities of performing the genetic operations. The termination criterion specifies the ending condition for the GP run and typically includes a maximum number of generations (Burke and Kendall, 2005). GP iteratively transforms a population of computer programs into a new generation of programs using various genetic

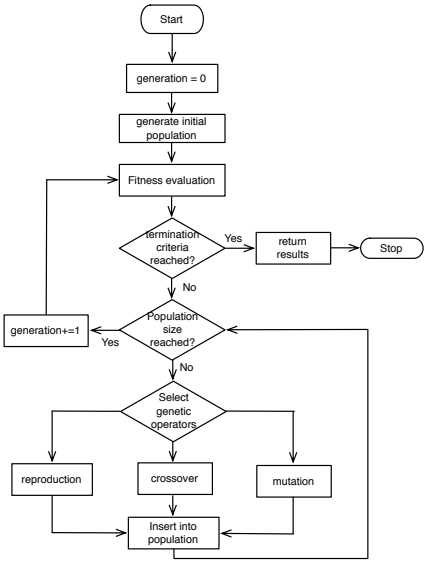


**Fig. 2** The GP process.

operators. Typical operators include crossover, mutation and reproduction. It is expected that over successive iterations, more and more useful structures or programs be evolved, eventually resulting in a structure having most useful sub-components. That structure would then represent the optimal or near-optimal solution to the problem. The crossover operator creates new structure(s) by combining randomly chosen parts from two selected programs or structures. The mutation operator creates a new structure by randomly altering a chosen part of a program. The reproduction operator simply copies a selected structure to the new population. Figure 2 shows the flowchart of the GP process.

For this study, the best GP program (having the minimum $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where $e_i$ is the actual outcome, $e_i^{'}$ is the classification result and $n$ is the size of the data set used to train the GP models) over the 10 runs of each fold of the 10-fold cross-validation is selected. The features making up this best GP program is then designated as the features selected by the GP algorithm. The control parameters that were chosen for the GP system are shown in Table 1. We did not fine tune these parameters for each new data set so as not to bias the results. The population size is related to the size of search space because if the search space is too large, GP will take longer times to find better solutions. The population size was fixed to 50 and this decision was based on our prior experience in experimentation with GP. The termination condition was set to 500 generations and was selected to give enough chance to GP for promoting variety in each generation. The tree initialization method selected was ramped half-and-half which results in very diverse population of trees, with balanced and unbalanced trees of several different depths (Silva, 2007). The probabilities of crossover, mutation and reproduction were set to 0.8, 0.1 and 0.1 respectively which was done again to promote maximum variation. The selection method used was roulette-wheel which is one of the few sampling methods used in GP to select parent individuals to produce their children.

**Table 1** GP control parameters.

| *Control parameter* | *Value* |
|---|---|
| Population size | 50 |
| Termination condition | 500 generations |
| Function set | {+,−,∗,/,sin,cos,log,sqrt} |
| Tree initialization | Ramped half-and-half method |
| Probabilities of crossover, mutation, reproduction | 0.8, 0.1, 0.1 |
| Selection method | roulette-wheel |

## 4 Experimental setup

In order to compare the performance of different FSS methods, the attribute sets selected by each method are tested with two learning algorithms, namely C4.5 and NB. These algorithms represent two different approaches (C4.5 being a decision-tree learner and NB being a probabilistic learner) and are considered state-of-the-art techniques. Also one of the previous benchmark studies (Hall and Holmes, 2003) have used the same algorithms for comparing the effectiveness of attribute selection.

The NB classifier is based on the Bayesian theorem. It analyses each data attribute independently and being equally important. The NB classifier learns the conditional probability of each attribute $A_i$ given the class label $C$, from the training data. Classification is done by applying the Bayes rule to compute the probability of $C$ given the particular instance of $A_1 \ldots An$, and then predicting the class with the highest posterior probability (Friedman et al, 1997). The NB classifier assumes that features are independent given class, that is, $P(X|C) = \prod_{i=1}^{n} P(X_i|C)$ where $X = (X_1 \ldots X_n)$ is a feature vector and $C$ is a class (Rish, 2001). By independence, it is meant as probabilistic independence, that is, $A$ is independent of $B$ given $C$ whenever $Pr(A|B, C) = Pr(A|C)$ for all possible values of A, B and C, whenever Pr(C) $> 0$ (Friedman et al, 1997).

C4.5 is the most well-known algorithm in the literature for building decision trees (Quinlan, 1993; Kotsiantis et al, 2007). C4.5 first creates a decision-tree based on the attribute values of the available training data such that the internal nodes denote the different attributes, the branches correspond to value of a certain attribute and the leaf nodes correspond to the classification of the dependent variable. The decision tree is made recursively by identifying the attribute(s) that discriminates the various instances most clearly, i.e., having the highest information gain. Once a decision tree is made, the prediction for a new instance is done by checking the respective attributes and their values.

We have applied the selected FSS methods to five real-world datasets from the PROMISE repository (Boetticher et al, 2007). These data sets are jEdit, AR5, MC1, CM1 and KC1_Mod. The datasets are available in ARFF (Attribute-Relation File Format), useable in the open source machine learning tool called WEKA (Waikato Environment for Knowledge Analysis) (Hall et al, 2009). The datasets are selected based on their variance in terms of number of instances and the number of attributes. The number of instances vary from being less than 50 up to several thousands, with the number of attributes varying from being in a single digit to nearly a hundred. The characteristics of datasets are given in Table 2.

**Table 2** Characteristics of datasets used in the study.

| No. | Dataset | Features | | | No. of classes | Train size | Test size |
|---|---|---|---|---|---|---|---|
| | | all | nominal | continuous | | | |
| 1 | jEdit | 9 | 1 | 8 | 2 | 369 | CV |
| 2 | AR5 | 30 | 1 | 29 | 2 | 36 | CV |
| 3 | MC1 | 39 | 1 | 38 | 2 | 9466 | CV |
| 4 | CM1 | 22 | 1 | 21 | 2 | 498 | CV |
| 5 | KC1_Mod | 95 | 1 | 94 | 2 | 282 | CV |

The source of jEdit data set is jEdit editor source code in Java and its Apache Subversion (SVN) log data. The data set contains metrics data computed by Understand C++ metric tool while bug data is extracted from SVN log files. The metrics are computed for jEdit release 4.0 while bugs are calculated between the releases 4.0 and 4.2. The source of AR5 data set is an embedded software used in manufacturing and implemented in C. Function/method level static code attributes are collected using Prest Metrics Extraction and Analysis Tool. The rest of the data sets (MC1, CM1, KC1_Mod) are NASA Metrics Data Program defect data sets. The metrics data consist of static code measures such as The McCabe and Halstead measures.

We restrict ourselves to evaluate the performance of *binary* classifiers which categorizes instances or software modules as being either fault-prone ($fp$) or non-fault prone ($nfp$). We are interested in predicting whether or not a module contains any faults, rather than the total number of faults. A common assessment procedure for binary classifiers is to count the number of correctly predicted modules over hold-out (test set) data. A fault prediction sheet (Menzies et al, 2004), as in Figure 3, is commonly used.

Based on the different possibilities in the fault prediction sheet, various measures are typically derived. El-Emam et al. (El-Emam et al, 2001) have derived a number of measures based on this; the most

| Classifier predicts faults | Module actually has faults: NO | Module actually has faults: YES |
|---|---|---|
| NO | TN: True Negative | FN: False Negative |
| YES | FP: False Positive | TP: True Positive |

**Fig. 3** The fault prediction sheet (confusion matrix).

common ones being rate of faulty module detection (or probability of detection ($PD$) or specificity), overall prediction accuracy ($acc$), probability of false alarm ($PF$ or recall) and precision ($prec$). However the measure of overall accuracy $acc$ has been criticized as being misleading since it ignores the data distribution and cost information (Ma and Cukic, 2007). The other measures of $PD$, $PF$ and $prec$ also reveal only one aspect of the prediction models at a time; thus their use introduces bias in performance assessment. Use of these measures also complicate comparisons and model selection since there is always a tradeoff between three measures, e.g. one model might exhibit a high $PD$ but lower $prec$ (Ma and Cukic, 2007).

A receiver operating characteristic (ROC) curve (Fawcett, 2006) and the area under a ROC curve (AUC) (Hanley and McNeil, 1982) have been shown to be more statistically consistent and discriminating than predictive accuracy, $acc$ (Ling et al, 2003). The ROC curve is also a more general way, than numerical indices, to measure a classifier's performance (Yousef et al, 2004). A ROC curve provides an intuitive way to compare the classification performances of different techniques. ROC is a plot of the trade-off between the ability of the classifier to correctly detect fault-prone modules ($PD$) and the number of non-fault prone modules that are incorrectly classified ($PF$) across all possible experimental threshold settings (Ma and Cukic, 2007; Jiang et al, 2008). In short the ($PF$, $PD$) pairs generated by adjusting the algorithms threshold settings forms an ROC curve. A typical ROC curve is shown in Figure 4.

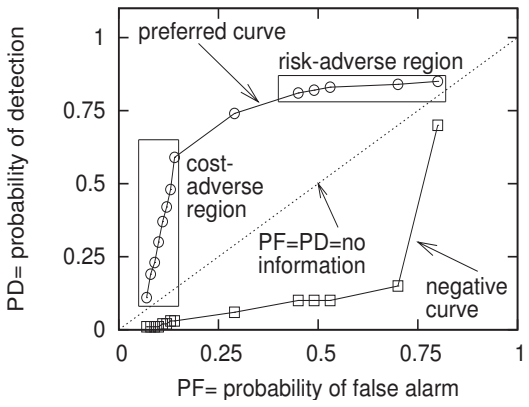


**Fig. 4** A typical ROC curve.

This concave curve has the probability of detection ($PD$) on $y$-axis while the $x$-axis shows the probability of false alarms ($PF$). The start and end points for the ROC curve are $(0, 0)$ to $(1, 1)$, respectively. The software engineers need to identify the points on the ROC curve that suits their risks and budgets for the project (Jiang et al, 2007). A straight line from $(0, 0)$ to $(1, 1)$ offers no information while the point $(PF = 0, PD = 1)$ is the ideal point on the ROC curve. A negative curve bends away from the ideal point while a preferred curve bends up towards the ideal point. As such, if we can divide the ROC space into four regions as shown in Figure 5, the only region with practical value for software engineers is region $A$ with acceptable $PD$ and $PF$ values. The regions $B$, $C$ and $D$ represent poor classification performance and hence are of little to no interest to software engineers (Ma and Cukic, 2007).

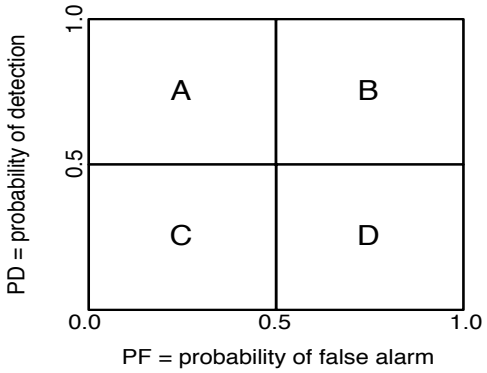


**Fig. 5** Four regions in the ROC space.

**Table 3** FSS results with NB.

| Dataset | NB | IG | RLF | PCA | CFS | CNS | WRP | GP |
|---|---|---|---|---|---|---|---|---|
| jEdit | 0.659 | **0.67** | **0.67** | 0.629 | **0.668** | **0.67** | 0.629 | **0.67** |
| AR5 | 0.907 | **0.933** | **0.942** | **0.938** | **0.942** | 0.866 | 0.875 | **0.915** |
| MC1 | 0.909 | **0.919** | **0.92** | 0.907 | 0.881 | 0.906 | 0.794 | **0.93** |
| CM1 | 0.658 | **0.718** | **0.728** | 0.653 | **0.691** | **0.685** | **0.738** | **0.68** |
| KC1_Mod | 0.78 | **0.851** | **0.938** | **0.854** | **0.84** | **0.86** | **0.802** | **0.87** |

Area under the curve (AUC) (Bradley, 1997) acts as a single scalar measure of expected performance and is an obvious choice for performance assessment when ROC curves for different classifiers intersect (Lessmann et al, 2008) or if the algorithm does not allow configuring different values of the threshold parameter. AUC, as with the ROC curve, is also a general measure of predictive performance since it separates predictive performance from class and cost distributions (Lessmann et al, 2008). The AUC measures the probability that a randomly chosen $fp$ module has a higher output value than a randomly chosen $nfp$ module (Fawcett, 2006). The value of AUC is always between 0 and 1; with a higher AUC indicating that the classifier is on average more to the upper left region $A$ in Figure 5.

We have used AUC as a measure of classification performance for the different FSS methods. For all the datasets, the AUC value averaged over 10 fold cross-validation runs, was calculated for each FSS method-dataset combination before and after FSS. For each cross-validation fold, the FSS method reduced the number of features in the dataset before being passed to C4.5 and NB classifiers.

## 5 Results and Analysis

Table 3 show results for all the datasets for FSS with NB. This table shows the AUC statistic for each FSS method and along with the AUC statistic when no feature selection is performed (the second column). The values in bold indicate if the use of the FSS method leads to an improvement of the AUC value, in comparison with when no FSS method is used. A number of FSS methods give an improved AUC value in comparison with the original AUC value without any feature selection. However, we need to test for any statistically significant differences between the different groups of AUC values. Since we have more than two samples with non-normal distributions, the Kruskal-Wallis test with significance level of 0.05 is used to test the null hypothesis that all samples are drawn from the same population. The result of the test ($p = 0.86$) suggested that it is not possible to reject the null hypothesis and, thus, there is no difference between any of the AUC values for the different FSS methods using NB **and** the AUC values of using NB as a classifier before and after applying the FSS methods.

Table 4 shows the number of attributes selected by each FSS method for NB. Wrapper, CFS, Relief and GP produce comparable AUC values with fewer number of selected features. PCA and IG, on the other hand, tend to select a much wider range of features to provide comparable classification results using NB.

**Table 4** Number of features selected by each FSS method for NB. The figures in % indicate the percentage of original features retained.

| Dataset | Org. | IG | RLF | PCA | CFS | CNS | WRP | GP |
|---|---|---|---|---|---|---|---|---|
| jEdit | 9 | 6 (66.67%) | 6 (66.67%) | 5 (55.55%) | 5 (55.55%) | 6 (66.67%) | 3 (33.33%) | 2 (22.22%) |
| AR5 | 30 | 4 (13.33%) | 2 (6.67%) | 7 (23.33%) | 2 (6.67%) | 2 (6.67%) | 1 (3.33%) | 6 (20%) |
| MC1 | 39 | 20 (51.28%) | 12 (30.77%) | 14 (35.90%) | 4 (10.26%) | 15 (38.46%) | 1 (2.56%) | 4 (10.26%) |
| CM1 | 22 | 5 (22.73%) | 3 (13.64%) | 4 (18.18%) | 7 (31.82%) | 11 (50%) | 2 (9.09%) | 14 (63.64%) |
| KC1_Mod | 95 | 3 (3.16%) | 8 (8.42%) | 17 (17.89%) | 8 (8.42%) | 2 (2.10%) | 4 (4.21%) | 7 (7.37%) |
| Average | 39 | 7.6 (19.49%) | 6.2 (15.90%) | 9.4 (24.10%) | 5.2 (13.33%) | 7.2 (18.46%) | 2.2 (5.64%) | 6.6 (16.92%) |

**Table 5** FSS results with C4.5.

| Dataset | NB | IG | RLF | PCA | CFS | CNS | WRP | GP |
|---|---|---|---|---|---|---|---|---|
| jEdit | 0.594 | **0.644** | **0.623** | **0.636** | **0.612** | 0.592 | 0.636 | **0.62** |
| AR5 | 0.717 | **0.817** | **0.866** | **0.763** | **0.866** | **0.757** | **0.817** | **0.797** |
| MC1 | 0.791 | **0.829** | **0.796** | 0.708 | **0.795** | 0.776 | 0.747 | **0.854** |
| CM1 | 0.558 | **0.615** | **0.587** | 0.506 | 0.542 | **0.596** | 0.49 | **0.644** |
| KC1_Mod | 0.599 | **0.806** | **0.684** | 0.555 | 0.553 | 0.589 | 0.579 | **0.69** |

**Table 6** Number of features selected by each FSS method for C4.5. The figures in % indicate the percentage of original features retained.

| Dataset | Org. | IG | RLF | PCA | CFS | CNS | WRP | GP |
|---|---|---|---|---|---|---|---|---|
| jEdit | 9 | 3 (33.33%) | 4 (44.44%) | 5 (55.55%) | 5 (55.55%) | 6 (66.67%) | 5 (55.55%) | 2 (22.22%) |
| AR5 | 30 | 1 (3.33%) | 2 (6.67%) | 7 (23.33%) | 2 (6.67%) | 2 (6.67%) | 1 (3.33%) | 6 (20%) |
| MC1 | 39 | 9 (23.08%) | 19 (48.72%) | 14 (35.90%) | 4 (10.26%) | 15 (38.46%) | 1 (2.56%) | 4 (10.26%) |
| CM1 | 22 | 2 (9.09%) | 9 (40.91%) | 4 (18.18%) | 7 (31.82%) | 11 (50%) | 2 (9.09%) | 14 (63.64%) |
| KC1_Mod | 95 | 3 (3.16%) | 7 (7.37%) | 17 (17.89%) | 8 (8.42%) | 2 (2.10%) | 4 (4.21%) | 7 (7.37%) |
| Average | 39 | 3.6 (9.23%) | 8.2 (21.02%) | 9.4 (24.10%) | 5.2 (13.33%) | 7.2 (18.46%) | 2.6 (6.67%) | 6.6 (16.92%) |

Table 5 shows the AUC statistic for each FSS method using C4.5 along with the AUC statistic when no feature selection is used (second column). Again, the values in bold indicate that the use of FSS method leads to an improvement of the AUC value, in comparison with when no FSS is used. The result show that multiple FSS methods do improve the classification performance across all data sets. However, the result of using the Kruskal-Wallis test with $\alpha = 0.05$ ($p = 0.628$) suggested that it is not possible to reject the null hypothesis of all samples being drawn from the same population. Thus there is no significant difference between: (a) any of the AUC values for the different FSS methods using C4.5 and (b) the AUC values of using C4.5 as a classifier before and after applying the FSS methods.

Table 6 shows the number of attributes selected by each FSS method for C4.5. WRP, IG, CFS and GP produce comparable average AUC values with fewer number of selected features. RLF, PCA and CNS tend to select a wider range of features to provide comparable classification results using C4.5.

As is clear from the above discussion, NB and C4.5 show insignificantly different classification accuracies for the variety of FSS methods used. This result is in agreement with the study by Hall and Holmes (Hall and Holmes, 2003) where the authors concluded that there is no single best approach for FSS for all situations. Song et al. (Song et al, 2011) and Menzies et al. (Menzies et al, 2007) also reach a similar conclusion:

"*[. . . ] we see that a data preprocessor/attribute selector can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important*" (Song et al, 2011).

"*[. . . ] the best attribute subsets for defects predictors can change dramatically from data set to data set. Hence, conclusions regarding the best attribute(s) are very brittle, i.e., may not still apply when we change data sets*" (Menzies et al, 2007).

Below we discuss the individual AUC values given for NB and C4.5 for different FSS methods.

From Table 3, it can be seen that for attribute selection with NB, the best AUC values are from three FSS methods (RLF, IG and GP) that improve NB on all five data sets and degrade it on none. CFS is the second best with improvement on four data sets and degradation on one. CNS, WRP and PCA give better performance on two data sets but also degrade performance on three data sets.

An overall pattern that is clear from Table 3 is that FSS is generally useful for NB's application to software fault prediction studies without significantly affecting classification accuracy. The results for NB in this study differ with the results given in the study by Hall and Holmes (Hall and Holmes, 2003). In that study, WRP was a clear winner in accuracy for NB. The potential reason for this performance could be attributed to the nature of the forward selection search in WRP which is used to generate the ranking such that strong attribute rankings are not identified. This search mechanism potentially works well in tandem with NB which has an attribute independence assumption (Hall and Holmes, 2003). However our results suggest that WRP is not at all a clear winner for NB where other FSS methods are also giving statistically insignificant results. This suggests that there are reasons other than the attribute independence assumption of NB that affects classification accuracy of NB with different FSS methods.

In terms of number of features selected for NB, the methods retaining the least number of features on average are WRP, CFS, RLF and GP. From Table 4 it can be seen that CFS chooses fewer features to all other FSS methods. From the techniques that were better on accuracy based on AUC values for NB, i.e., RLF and GP, are also among the methods that retains the least number of features. This is encouraging and shows that RLF and GP produce higher AUC values for NB while retaining minimum number of features on average, considering the data sets used in the experiment. PCA turns out to be worst in terms of retaining few features.

From Table 5, one can see the individual AUC values for attribute selection with C4.5. The results are in agreement with the results from NB. The best FSS methods for C4.5 are IG, RLF and GP which improve C4.5's performance on five data sets and degrade it on none. CFS improve C4.5's performance on three data sets and degrades it on two. CNS and PCA improve C4.5's performance on two data sets and degrades it on three. Result for WRP is that it degrades performance on four data sets and improves it on one.

As with NB, an overall pattern clear from Table 5 is that FSS is generally useful for C4.5's application to software fault prediction without significantly affecting classification accuracy. According to the study by Hall and Holmes (Hall and Holmes, 2003): "*The success of ReliefF and consistency with C4.5 could be attributable to their ability to identify attribute interactions (dependencies). Including strongly interacting attributes in a reduced subset increases the likelihood that C4.5 will discover and use interactions early on in tree construction before the data becomes too fragmented*". The fact that we did not get consistent results with both RLF and CNF allows us to suggest that the ability to identify attribute interactions (dependencies) might not be the only differentiating factor in classification accuracy with respect to C4.5. As was our argument in case of NB, we argue that there are factors other than the ability to identify attribute interactions that are affecting classification accuracy of C4.5.

In terms of number of features retained for C4.5 (Table 6), WRP retains the minimum percentage of features on average, followed by IG, CFS, GP, CNS and RLF respectively. PCA is the worst in terms of retaining features for C4.5 with 24.10%. Our results show WRP as a clear winner in our case while CFS is at third place in terms of retaining the minimum number of features on average. From the methods that were better on accuracy based on AUC values for C4.5 (IG, RLF and GP), IG and GP are at second and fourth place respectively in terms of retaining minimum percentage of features on average. This might suggest that IG and GP are suitable FSS methods for C4.5 considering the data sets we used in this study. RLF is down in ranking in Table 6, however its larger feature set sizes are justified by higher classification accuracy than the other methods.

Below we summarize the results of our study:

- FSS is useful and generally improves classification accuracy.
- There are no statistically significant differences for either NB or C4.5 for the variety of FSS methods used.
- Based on individual AUC values, IG, RLF and GP improve NB and C4.5 on five data sets and degrade them on none.
- CFS, RLF and GP retain the minimum percentage of features on average for NB.
- PCA is the worst in terms of retaining the minimum percentage of features on average for NB.
- There are factors other than the attribute independence assumption of NB that affect its classification accuracy with different FSS methods.
- IG, RLF and GP improve importance of C4.5 on five data sets and degrades it on none.
- WRP and CFS retains the minimum percentage of features on average for C4.5.

– There are factors other than the ability to identify attribute interactions that are affecting classification accuracy of C4.5.

After having discussed the results, we come to a crucial question: If various FSS methods perform differently for different machine learning algorithms, what factors are most important to consider while selecting FSS methods to use? Hall and Holmes (Hall and Holmes, 2003) argue in their paper that there are three factors to consider:

1. An understanding of how different FSS methods work.
2. Strengths and weaknesses of the target learning algorithm.
3. Background knowledge about data.

While agreeing to all of the above factors, we add that if the goal is to improve classification accuracy of a learner, a decision about selecting a FSS method has to be reached in combination with following additional criteria:

1. Choice of resampling method.
2. Choice of data filtering technique (to address class imbalance, outlier removal, handling missing values and discretizing numeric attributes).
3. Choice of accuracy measure to use.

Choice of a resampling method concerns how to divide historical data into training and test data. In order to assess the generalizability of a learner, it is necessary that the test data are not used in anyway to build the learners (Song et al, 2011). A recent study by Afzal et al. (Afzal et al, 2012) recommended the use of bootstrapping for software defect prediction studies. If not bootstrapping, the second recommended choice is to use leave-one-out cross validation for smaller data sets and 10-fold cross validation for large data sets. This subject however require more empirical studies to further strengthen these recommendations.

We also argue that the role of a data filtering technique in accurately classifying software components is important. A study by Gao et al. (Gao et al, 2012) demonstrated that data sampling (over-sampling or under-sampling) can counteract the adverse effect attributed to class imbalance in software fault prediction. They also concluded that feature selection became more efficient when used after data sampling. There are other examples of the use of data filtering techniques in software fault prediction, e.g., Menzies et al. (Menzies et al, 2007) and Song et al. (Song et al, 2011) used a log filtering preprocessor which replaces all numerics with their logarithms.

Choice of an accuracy indicator to evaluate the performance of defect predictors is also an important decision criterion. The use of MMRE as an accuracy indicator has been criticized by several authors (Kitchenham et al, 2001; Myrtveit et al, 2005; Foss et al, 2003). Consequently, area under the receiver operating characteristic curve (AUC) is increasingly being used as a standard choice for performance evaluation in software fault predictions studies.[2]

It is important to highlight the performance of an evolutionary computational method (GP) as a FSS method. For both NB and C4.5, GP improved the AUC values for maximum number of data sets and degraded on the least number of data sets. For NB, GP is also among the methods that retained minimum percentage of features on average. It is worth noting that for GP feature selection is an implicit part of GP evolution. This enables automatic or semi-automatic selection of features during model generation. GP allows almost any combination of a number of features. Evolution can freely add/remove multiple features and can reconsider previous selections as new combinations are tried (Langdon and Buxton, 2004). A potential disadvantage of using an evolutionary algorithm like GP is that it can take more computational resources as compared with other methods. Therefore with GP it has to be a tradeoff between how much improvement in classification accuracy is required against available resources.

## 6 Validity evaluation

Wohlin et al. (Wohlin et al, 2000) discuss four types of threats to an experimental study: external (ability to generalize), conclusion (ability to apply statistical tests), internal (ability to correctly infer connections between dependent and independent variables) and construct (ability of dependent variable to capture the effect being measured).

[2] Section 4 provides more details about AUC.

*External validity* The datasets used in this study represent real-world use, collected during the course of real industry projects developed by professionals. The datasets differed in their number of attributes and sizes. However as noted by Gao et al. (Gao et al, 2012), analysis of another data set from different application domain may provide different results which is a likely threat in all empirical software engineering research.

*Conclusion validity* We were mindful that the type of statistical tests could potentially affect end results, therefore Kruskal Wallis test was used as we had more than two samples with non-normal distributions. This empirical study was performed using 10-fold cross-validation for statistically reliable results (recommended in (Kohavi, 1995; Afzal et al, 2012)). The performance of classifiers is compared using area under the receiver operating characteristic curve (AUC) which we motivate is a standard way of evaluating classification results.

*Internal validity* According to Gao et al. (Gao et al, 2012), different factors can affect the internal validity of fault proneness estimates: 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 dataset. We used the publicly available data sets so other researchers can replicate our work. Secondly we have given the parameter settings for different methods used to ease replication of our work.

*Construct validity* The datasets used in this study are the ones donated by the authors of fault prediction studies and mostly use structural measures. Structural measures are widely used in software fault prediction studies (Catal and Diri, 2009a), however finding the right predictors for software fault proneness is an active area of research.

## 7 Conclusions

Feature subset selection (FSS) methods are used to keep the number of features in a dataset as small as possible. Out of the various perceived advantages of using these FSS methods (Section 1), this study evaluate whether or not the use of FSS methods have any significant affect on the classification accuracy of software fault prediction when used with two diverse learning algorithms, C4.5 and naïve Bayes.

We compare a total of seven FSS methods, representing a mix of state-of-the-art methods and an evolutionary computation method, on five software fault prediction datasets from the PROMISE data repository. Our findings show that feature subset selection is generally useful for software fault prediction using naïve Bayes and C4.5. However there are no clear winners for either of the two learning algorithms for the variety of FSS methods used.

Based on individual AUC values, IG, RLF and GP improve naïve Bayes and C4.5 on five data sets and degrade it on none. RLF, GP and CFS also retain the minimum percentage of features on average for naïve Bayes. WRP and CFS retain the minimum percentage of features on average for C4.5.

In summary, our results suggest that RLF, IG and GP are the best FSS methods for software fault prediction accuracy using naïve Bayes and C4.5. CNS and CFS are also good overall performers. We recommend that any future software fault prediction study be preceded by an initial analysis of FSS methods, not missing on methods that have shown to be more consistent than their competitors. It is recommended in literature that for selecting a FSS method, a data miner needs to have an understanding of how different FSS methods work, strengths and weaknesses of the target learning algorithm and background knowledge about data. In the context of software fault prediction studies, we additionally recommend that the data miner needs to have an understanding of different resampling methods, data filtering techniques and accuracy measures for increasing the reliability and validity of prediction results. In this study, we do not offer an interpretation of features retained by different FSS methods. It is, nevertheless, an interesting future work to relate features retained by different FSS methods with functioning of the target learning algorithm and background knowledge about data.

## References


Afzal W, Torkar R, Feldt R, Gorschek T (2009) Genetic programming for cross-release fault count predictions in large and complex software projects. In: Chis M (ed) Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques, IGI Global, Hershey, USA, pp 94–126

Afzal W, Torkar R, Feldt R (2012) Resampling methods in software quality classification. International Journal of Software Engineering and Knowledge Engineering 22:203–223

Altidor W, Khoshgoftaar TM, Gao K (2010) Wrapper-based feature ranking techniques for determining relevance of software engineering metrics. International Journal of Reliability, Quality and Safety Engineering 17:425–464

Azzeh M, Neagu D, Cowling P (2008) Improving analogy software effort estimation using fuzzy feature subset selection algorithm. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE'08), ACM, New York, NY, USA

Bäck T, Fogel DB, Michalewicz Z (eds) (2000) Evolutionary computation 1 – Basic algorithms and operators. Taylor & Francis Group, LLC, 27 Madison Avenue, New York, USA

Blum AL, Langley P (1997) Selection of relevant features and examples in machine learning. Artificial Intelligence 97:245–271

Boetticher G, Menzies T, Ostrand T (2007) PROMISE repository of empirical software engineering data. Http://promisedata.org/ repository, West Virginia University, Department of Computer Science

Bradley AP (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30:1145–1159

Burke EK, Kendall G (eds) (2005) Search methodologies—Introductory tutorials in optimization and decision support techniques. Springer Science and Business Media, Inc., 233 Spring Street, New York, USA

Catal C, Diri B (2009a) Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences 179:1040–1058

Catal C, Diri B (2009b) A systematic review of software fault prediction studies. Expert Systems with Applications 36(4):7346 – 7354

Chen Z, Boehm B, Menzies T, Port D (2005a) Finding the right data for software cost modeling. IEEE Software 22:38–46

Chen Z, Menzies T, Port D, Boehm B (2005b) Feature subset selection can improve software cost estimation accuracy. SIGSOFT Software Engineering Notes 30(4):1–6

Dash M, Liu H (1997) Feature selection for classification. Intelligent Data Analysis 1(1-4):131–156

Dejaeger K, Verbeke W, Martens D, Baesens B (2012) Data mining techniques for software effort estimation: A comparative study. IEEE Transactions on Software Engineering 38:375–397

Dybå T, Kampenes VB, Sjøberg DI (2006) A systematic review of statistical power in software engineering experiments. Information and Software Technology 48(8):745 – 755

El-Emam K, Benlarbi S, Goel N, Rai SN (2001) Comparing case-based reasoning classifiers for predicting high risk software components. Journal of Systems and Software 55(3):301–320

Fawcett T (2006) An introduction to ROC analysis. Pattern Recognition Letters 27(8):861–874

Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Transactions on Software Engineering 25(5):675–689

Foss T, Stensrud E, Kitchenham BA, Myrtveit I (2003) A simulation study of the model evaluation criterion MMRE. IEEE Transactions on Software Engineering 29(11)

Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Machine Learning 29(2-3):131–163

Gao K, Khoshgoftaar TM, Wang H, Seliya N (2011) Choosing software metrics for defect prediction: an investigation on feature selection techniques. Software Practice and Experience 41(5):579–606

Gao K, Khoshgoftaar T, Seliya N (2012) Predicting high-risk program modules by selecting the right software measurements. Software Quality Journal 20:3–42

Gray D, Bowes D, Davey N, Sun Y, Christianson B (2011) The misuse of the NASA metrics data program data sets for automated software defect prediction. IET Seminar Digests 2011(1):96–103

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. Journal of Machine Learning Research 3:1157–1182

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: An update. SIGKDD Explorations Newsletter 11:10–18

Hall MA (2000) Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the 2000 International Conference on Machine Learning (ICML'00), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Hall MA, Holmes G (2003) Benchmarking attribute selection techniques for discrete class data mining. IEEE Transactions on Knowledge and Data Engineering 15:1437–1447

Hall T, Beecham S, Bowes D, Gray D, Counsell S (2011) A systematic review of fault prediction performance in software engineering. IEEE Transactions on Software Engineering PP(99)

Hanley JA, McNeil BJ (1982) The meaning and use of the area under a receiver operating characteristic (ROC) curve. RADIOLOGY 143(1):29–36

Jain AK, Duin RPW, Mao J (2000) Statistical pattern recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence 22:4–37

Janecek A, Gansterer W, Demel M, Ecker G (2008) On the relationship between feature selection and classification accuracy. In: Proceedings of the 3rd Workshop on New Challenges for Feature Selection in Data Mining and Knowledge Discovery (FSDM'08), Microtome Publishing, Brookline, MA, USA

Jiang Y, Cukic B, Menzies T (2007) Fault prediction using early lifecycle data. In: Proceedings of the The 18th IEEE International Symposium on Software Reliability (ISSRE'07), IEEE Computer Society, Washington, DC, USA

Jiang Y, Cukic B, Menzies T, Bartlow N (2008) Comparing design and code metrics for software quality prediction. In: Proceedings of the 4th international workshop on predictor models in software engineering (PROMISE'08), ACM, New York, NY, USA

Khoshgoftaar TM, Seliya N (2004) Fault prediction modeling for software quality estimation: Comparing commonly used techniques. Empirical Software Engineering 8(3):255–283

Khoshgoftaar TM, Nguyen L, Gao K, Rajeevalochanam J (2003) Application of an attribute selection method to CBR-based software quality classification. In: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03), IEEE Computer Society, Washington, DC, USA

Khoshgoftaar TM, Seliya N, Sundaresh N (2006) An empirical study of predicting software faults with case-based reasoning. Software Quality Control 14:85–111

Khoshgoftaar TM, Gao K, Seliya N (2010) Attribute selection and imbalanced data: Problems in software defect prediction. IEEE Computer Society, Los Alamitos, CA, USA

Khoshgoftaar TM, Gao K, Napolitano A (2012) An empirical study of feature ranking techniques for software quality prediction. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) 22:161–183

Kira K, Rendell LA (1992) The feature selection problem: Traditional methods and a new algorithm. In: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)

Kirsopp C, Shepperd MJ, Hart J (2002) Search heuristics, case-based reasoning and software project effort prediction. In: Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO'02), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1367–1374

Kitchenham BA, Pickard LM, MacDonell S, Shepperd M (2001) What accuracy statistics really measure? IEE Proceedings Software 148(3)

Kohavi R (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint conference on Artificial Intelligence (IJCAI'95), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Kohavi R, John GH (1997) Wrappers for feature subset selection. Artificial Intelligence 97:273–324

Kotsiantis S, Zaharakis I, Pintelas P (2007) Machine learning: A review of classification and combining techniques. Artificial Intelligence Review 26(3):159–190

Koza JR (1992) Genetic programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA

Langdon WB, Buxton BF (2004) Genetic programming for mining DNA chip data from cancer patients. Genetic Programming and Evolvable Machines 5:251–257

Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 34(4):485–496

Li Y, Xie M, Goh T (2009) A study of mutual information based feature selection for case based reasoning in software cost estimation. Expert Systems with Applications 36(3, Part 2):5921 – 5931

Ling CX, Huang J, Zhang H (2003) AUC: A statistically consistent and more discriminating measure than accuracy. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)

Liu H, Setiono R (1996) A probabilistic approach to feature selection—A filter solution. In: Proceedings of the 1996 International Conference on Machine Learning (ICML'96), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 319–327

Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and Data Engineering 17(4):491–502

Ma Y, Cukic B (2007) Adequate and precise evaluation of quality models in software engineering studies. In: Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering (PROMISE'07), IEEE Computer Society, Washington, DC, USA, pp 1–

Menzies T, DiStefano J, Orrego A, Chapman RM (2004) Assessing predictors of software defects. In: Proceedings of the Workshop on Predictive Software Models, collocated with ICSM'04, URL `http://menzies.us/pdf/04psm.pdf`

Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33(1):2–13

Menzies T, Jalali O, Hihn J, Baker D, Lum K (2010) Stable rankings for different effort models. Automated Software Engineering 17:409–437

Molina LC, Belanche L, Àngela Nebot (2002) Feature selection algorithms: A survey and experimental evaluation. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), IEEE Computer Society, Washington, DC, USA, pp 306–313

Muni D, Pal N, Das J (2006) Genetic programming for simultaneous feature selection and classifier design. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 36(1):106–117

Myrtveit I, Stensrud E, Shepperd M (2005) Reliability and validity in comparative studies of software prediction models. IEEE Transactions on Software Engineering 31(5):380–391

Novakovic J (2009) Using information gain attribute evaluation to classify sonar targets. In: Proceedings of the 17th Telecommunications forum (TELFOR'09)

Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, URL `http://www.gp-field-guide.org.uk`, (With contributions by J. R. Koza)

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Rish I (2001) An empirical study of the naive Bayes classifier. In: Proceedings of the workshop on empirical methods in AI (IJCAI'01)

Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J (2007a) Detecting fault modules applying feature selection to classifiers. In: IEEE International Conference on Information Reuse and Integration (IRI'07)

Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J, Garre M (2007b) Attribute selection in software engineering datasets for detecting fault modules. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'07)

Shivaji S, Jr EJW, Akella R, Kim S (2009) Reducing features to improve bug prediction. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE'09), IEEE Computer Society, Washington, DC, USA

Sikonja M, Kononenko I (1997) An adaptation of relief for attribute estimation in regression. In: Proceedings of the 14th International Conference on Machine Learning (ICML'97)

Silva S (2007) GPLAB—A genetic programming toolbox for MATLAB. `http://gplab.sourceforge.net`, Last checked: 22 Dec 2014.

Smith MG, Bull L (2003) Feature construction and selection using genetic programming and a genetic algorithm. In: Proceedings of the 6th European Conference on Genetic Programming (EuroGP'03), Springer-Verlag, Berlin, Heidelberg

Song Q, Jia Z, Shepperd M, Ying S, Liu J (2011) A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering 37(3):356–370

Vivanco R, Kamei Y, Monden A, Matsumoto K, Jin D (2010) Using search-based metric selection and oversampling to predict fault prone modules. In: 2010 23rd Canadian Conference on Electrical and

Computer Engineering (CCECE'10)
Wang H, Khoshgoftaar T, Gao K, Seliya N (2009) High-dimensional software engineering data and feature selection. In: 21st International Conference on Tools with Artificial Intelligence (ICTAI'09), pp 83–90
Wang H, Khoshgoftaar TM, Napolitano A (2012) Software measurement data reduction using ensemble techniques. Neurocomputing 92(0):124 – 132
Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2000) Experimentation in software engineering: An introduction. Kluwer Academic Publishers, USA
Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm. IEEE Intelligent Systems and their Applications 13(2):44–49
Yousef WA, Wagner RF, Loew MH (2004) Comparison of non-parametric methods for assessing classifier performance in terms of ROC parameters. In: Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop (AIPR'04), IEEE Computer Society, Washington, DC, USA