SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2009; **39**:423–440
Published online 2 December 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.905

Mining extremely small data sets with application to software reuse

Yuan Jiang, Ming Li and Zhi-Hua Zhou*, †

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

SUMMARY

A serious problem encountered by machine learning and data mining techniques in software engineering is the lack of sufficient data. For example, there are only 24 examples in the current largest data set on software reuse. In this paper, a recently proposed machine learning algorithm is modified for mining extremely small data sets. This algorithm works in a twice-learning style. In detail, a random forest is trained from the original data set at first. Then, virtual examples are generated from the random forest and used to train a single decision tree. In contrast to the numerous discrepancies between the empirical data and expert opinions reported by previous research, our mining practice shows that the empirical data are actually consistent with expert opinions. Copyright © 2008 John Wiley & Sons, Ltd.

Received 22 October 2007; Revised 26 August 2008; Accepted 3 September 2008

KEY WORDS: data mining; machine learning; software reuse; extremely small data set; twice learning; ensemble learning; random forest

1. INTRODUCTION

Machine learning and data mining techniques have achieved great success in many fields [1]; however, their usefulness in software engineering has not been demonstrated well. A serious problem here is the lack of large data sets. This is because of the fact that every instance in such data sets should describe a real case of software engineering practice, yet collecting a lot of software engineering cases with uniform description is very expensive and difficult. For example, in

*Correspondence to: Zhi-Hua Zhou, National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China.
†E-mail: zhouzh@nju.edu.cn

Contract/grant sponsor: National Science Foundation of China; contract/grant numbers: 60505013, 60635030, 60721002
Contract/grant sponsor: Jiangsu Science Foundation; contract/grant number: BK2008018
Contract/grant sponsor: National 973 Fundamental Research Program of China; contract/grant number: 2002CB312002

software reuse [2], the largest data set at present contains only 24 instances that are described by 27 variables [3].

Usually, the availability of sufficient training examples is one of the prerequisites of a successful machine learning or data mining deployment. When there are only a few training examples, serious *overfitting* often occurs. That is, the learner, such as a decision tree or a neural network, is easy to be trained with perfect performance on the training examples but the learner can hardly generalize well on new instances because it has overly fit the peculiarities of the training examples.

Note that the number of training examples that are sufficient is task-dependent. For example, four different training examples are sufficient for the task of learning the XOR function, yet thousands of training examples might be insufficient for a complicated task such as weather forecasting. In general, if a data set cannot fully cover the whole variable space, then the data set is referred as a *small data set*. In this sense, the above-mentioned software reuse data set is an *extremely small data set* because the number of its examples is even less than that of variables. Such a degree of data scarcity is more serious than that of the usual small data sets encountered in popular machine learning and data mining settings [4].

Obviously, there are two possible ways to overcome the data scarcity problem. One is to collect more data whereas the other is to design techniques that can deal with extremely small data sets. As mentioned before, in software engineering the first way is difficult to go along with, and thus the second way is more desirable. Based on this recognition, this paper develops the RF2Tree (Random Forest to Tree) algorithm. RF2Tree is a variant of a recently proposed machine learning technique [5] and can be used to mine extremely small data sets. The RF2Tree works in a *twice-learning* style. In the first phase of the learning process, a random forest [6] is built from the original data set. Then, in the second phase, the random forest is used to generate many virtual examples that are used to train a single decision tree.

The RF2Tree algorithm is applied to the above-mentioned software reuse data set. The mining results are discussed and compared with some previous studies [3,7,8]. In contrast to the numerous discrepancies between the empirical data and the expert opinions reported by previous research, our mining practice shows that the empirical data are actually consistent with expert opinions.

The remainder of this paper is organized as follows. Section 2 presents the RF2Tree algorithm. Section 3 reports on the mining practice on software reuse data set. Section 4 concludes the paper.

2. RF2Tree

2.1. The algorithm

Ensemble learning [9] is a machine learning paradigm where multiple learners are trained to solve the same problem. Since the predictive accuracy of an ensemble is usually significantly better than that of a single learner, ensemble learning has attracted much attention in the machine learning and data mining community. Recently, Zhou and Jiang [5] proposed to use a neural network ensemble to preprocess the training data for a rule learning approach. They used the original training data set to generate an ensemble at first. Then, they randomly generated new instances and passed them to the ensemble for classification. The outputs from the ensemble were regarded as labels of these instances. By combining the predicted labels and the training inputs, new examples were obtained and used to enlarge the training data set. The enlarged training data set was finally used

DOI: 10.1002/spe

by a rule learning approach. This approach has been applied to gene expression data and has obtained interesting results [10]. Zhou and Jiang [11] showed that using an ensemble to preprocess the training data set for a succedent learner (e.g. a decision tree) according to the above routine is beneficial so long as: (1) the original training data set is too small to fully capture the target distribution, or the original training set contains noise and (2) the ensemble is more accurate than the single learner if both of them are directly trained from the original training data set.

It seems that the above twice-learning paradigm can be adopted to deal with extremely small data sets, because the ensemble built in the first phase of the learning process can be used to enlarge the data sets and hence enables the success of the learning in the second phase. Considering that variables describing software engineering practices are usually categorical, the RF2Tree utilizes decision tree ensemble instead of neural network ensemble in the first-phase learning. A decision tree is a flow-chart-like tree structure, where each internal node denotes a *test* on a variable, each branch represents an outcome of the test, and leaf nodes represent classes. An ensemble generated by RandomForests [6] is used here[‡], which consists of many decision trees each of which is grown as follows: (1) if the number of training examples is n, randomly sample n examples with replacement from the original training data set and use the sample to train a tree; (2) if there are m variables, a number $l \gg m$ is specified such that at each tree node, l variables are randomly selected out of the m variables and the best one of these l variables is used to split the node; and (3) each tree is grown to the largest. Upon receiving a new instance to classify, every tree will give a classification, and the classification given by the most number of trees in the forest is regarded as the classification of the new instance.

After the random forest is built, the original training data set can be enlarged with virtual examples. In detail, the RF2Tree uses the forest to classify some randomly generated new instances and puts the resulting examples, i.e. classified instances, into the training data set. Thus, a single decision tree can be built from a training data set far larger than the original one. Here, the new instances are generated in the style such that all possible values of different variables that have not appeared in the original training data set are tried as far as the memory of the computer is capable to hold all of them; otherwise, they are subsampled to fit the memory capacity. For example, for each nominal variable, we uniformly sample a value from all possible values; for each numeric variable, we can sample a value from a Gaussian distribution estimated from the variable values of all original training data. The single decision tree is realized by the J4.8 algorithm in the WEKA package [12]. The pseudo-code describing the RF2Tree algorithm is shown in Table I.

The scheme of using a trained system to generate examples has been employed in some paradigms for extracting rules from complicated learning systems [13,14]. The purposes of these studies are for knowledge acquisition or comprehensibility enhancement, while the purpose of the RF2Tree is to deal with extremely small data sets. Such a scheme has also been used in some information fusion paradigms [15], where the examples are generated by different sensors and then utilized to build a fuser. From the view of ensemble learning, the different sensors can be regarded as component learners while the fuser is the combiner. Therefore, in these paradigms the examples are generated and used inside the ensemble, that is, generated by the component learners and used by the combiner. In the RF2Tree, however, the examples are generated by the ensemble and used outside of the ensemble, that is, being used to train a learner that is not a part of the ensemble.

[‡]RandomForests is a strong decision tree ensemble method. A package including the code, manual, and other useful resources is available at http://stat-www.berkeley.edu/users/breiman/RandomForests/.

DOI: 10.1002/spe

Table I. The RF2Tree algorithm.

```
Input: training set S, decision tree algorithm L
Output: decision tree DT
Process:
    F* = RandomForests(S, L)
        /* train a random forest F* from S via RandomForests */
    S' = GenVirExp(F*)
        /* use the random forest to generate virtual examples as more as possible */
    DT = L(S ∪ S')
        /* grow a decision tree from the enlarged training data set */
```

2.2. Justification

We can provide theoretical justification for the working mechanism of the RF2Tree method, and derive two conditions to ensure the usefulness of the RF2Tree. In fact, as the RF2Tree is a variant of the algorithm proposed in [11], the justification is almost the same as that described in [11]. For the self-containess of this paper, we include the justification below.

Suppose the target to be learned is a function $F: X \to Y$, where X is the input space while Y is the label set. Note that such a function expresses a distribution in the variable space determined by X and Y. Let F_E denote the function implemented by a random forest ensemble trained from a given training set. Thus, the probability for F_E to approach F is as shown in Equation (1), where err_E denotes the error rate of the ensemble.

$$P_{F_E} = P_{F=F_E} = 1 - P_{F \neq F_E} = 1 - err_E \tag{1}$$

Let F_T denote the function implemented by a decision tree trained from a given training set. Thus, the probability for F_T to approach F is given by the following equation, where err_T denotes the error rate of the tree

$$P_{F_T} = P_{F=F_T} = 1 - P_{F \neq F_T} = 1 - err_T \tag{2}$$

Err_T can be broken into three parts. The first part is an error term caused by the limited learning ability of the decision tree. That is, the function $F_T^{(c)}$ implemented by a decision tree may make some error in prediction, even if we grow the tree from a training set, which contains no noise and captures the whole target distribution. Such a kind of error is denoted by $err_T^{(c)}$. Note that $err_T^{(c)}$ may be extremely small. The probability for $F_T^{(c)}$ to approach the target F is given by the following equation:

$$P_{F_T^{(c)}} = P_{F=F_T^{(c)}} = 1 - P_{F \neq F_T^{(c)}} = 1 - err_T^{(c)} \tag{3}$$

The second part is an error term ($err_T^{(n)}$) caused by the noise existed in the training set; the last part is an error term ($err_T^{(s)}$) caused by the fact that a finite sample, such as a training set, which does not contain all possible feature vectors or cannot fully capture the target distribution. Therefore, the error of a decision tree can be decomposed as

$$err_T = err_T^{(c)} + err_T^{(n)} + err_T^{(s)} \tag{4}$$

Obviously, an extremely small data set cannot fully capture the target distribution. If the data set contains no noise, then $err_T^{(n)}$ is zero and err_T is dominated by $err_T^{(s)}$. Let F_T^* denote the function

DOI: 10.1002/spe

implemented by a decision tree that grows from the training set processed by an ensemble in the way of the RF2Tree. To simplify the discussion, assume that the training set generated by the ensemble contains all possible feature vectors. Then, from the view of F_E, this training set captures the whole distribution. This distribution, however, is not the real target distribution F, and the probability for F_E to approach F is P_{F_E}. Thus, the probability for F_T^* to approach F can be expressed as follows:

$$P_{F_T^*} = P_{F=F_T^*} = P_{F_E} P_{F_T^{(c)}} \tag{5}$$

Substituting Equations (1) and (3) into Equation (5) results in Equation (6).

$$P_{F_T^*} = (1 - err_E)(1 - err_T^{(c)}) \tag{6}$$

Comparing Equation (6) with Equation (2), it can be derived that $P_{F_T^*}$ is larger than P_{F_T}, i.e. F_T^* approaches F better if Equation (7) holds

$$err_E < \frac{err_T - err_T^{(c)}}{1 - err_T^{(c)}} \tag{7}$$

As $err_T^{(s)}$ dominates err_T and $err_T^{(c)}$ may be extremely small, it is obvious that Equation (7) can be satisfied so far as err_E is much smaller than err_T. Thus, using a random forest ensemble to process the original extremely small training set in the way as the RF2Tree can benefit the construction of the decision tree, even when the original training set contains no noise, given that the ensemble is significantly more accurate than the single decision tree grown directly from the original training set. In fact, with a similar derivation, it can be found that when the original training set contains much noise, even if the original training set has fully captured the target distribution, the above twice-learning process is also beneficial, given that the ensemble is significantly more accurate than the decision tree grown directly from the original training set.

2.3. Verification

We conduct experiments to verify whether the RF2Tree algorithm can really work well on extremely small data sets. Note that according to the No Free Lunch theorem [16,17], no technique can be superior in all cases, and what we want is a technique that works well on as many cases as possible. Twenty two-class data sets from the UCI machine learning repository [18] are used. In addition, we conduct experiments on two software engineering data sets, namely *Eclipse* and *Firefox*, which have been studied in automatic bug reports assignment [19]. Each example in the data set corresponds to a bug report for Eclipse and Firefox from 1st May 2006 to 31st May 2006. *Eclipse* consists of 1035 examples belonging to 40 categories, while *Firefox* consists of 1339 examples belonging to 33 categories. In order to obtain extremely small data sets, examples in these original data sets are randomly discarded according to different e/a *ratios*, i.e. the ratio of the number of examples divided by the number of variables. Therefore, experiments are in fact conducted on 110 experimental data sets as five e/a *ratios*, including 0.5, 0.8, 1.0, 1.2, and 1.5 are considered. Note that *Eclipse* and *Firefox* are multi-class data sets and, thus, some of the classes might not appear in the training set generated according to some e/a ratios. Thus, we transform these multi-class data sets into two-class data sets by merging closely related classes into one metaclass.

On each experimental data set, 100 runs of hold-out test is performed. In detail, an experimental data set is randomly partitioned into two sets with similar distributions in each run, where about

DOI: 10.1002/spe

Table II. Hold-out test errors with $e/a = 0.5$ (100 runs).

Data set	Single tree	RandomForests	RF2Tree
breast-cancer	0.330±0.364	0.240±0.337	0.240±0.329
breast-w	0.165±0.257	0.050±0.151	0.050±0.151
colic	0.323±0.261	0.210±0.210	0.253±0.228
colic.ORIG	0.458±0.171	0.356±0.149	0.400±0.133
credit-a	0.437±0.271	0.363±0.251	0.413±0.251
credit-g	0.460±0.241	0.320±0.206	0.337±0.214
diabetes	0.445±0.317	0.315±0.307	0.325±0.313
heart-statlog	0.375±0.344	0.245±0.329	0.275±0.336
hepatitis	0.342±0.153	0.248±0.025	0.248±0.025
ionosphere	0.278±0.185	0.177±0.157	0.228±0.155
kr-vs-kp	0.383±0.165	0.287±0.169	0.314±0.157
labor	0.300±0.244	0.193±0.196	0.227±0.200
mushroom	0.245±0.227	0.135±0.189	0.170±0.210
rangeseg	0.357±0.233	0.217±0.167	0.240±0.165
sick	0.096±0.152	0.000±0.000	0.000±0.000
sick-euthyroid	0.098±0.201	0.000±0.000	0.010±0.049
sonar	0.416±0.120	0.272±0.129	0.379±0.127
vote	0.180±0.234	0.080±0.158	0.090±0.163
wdbc	0.222±0.194	0.097±0.116	0.120±0.123
wpbc	0.378±0.213	0.236±0.134	0.228±0.145
Eclipse	0.346±0.113	0.169±0.085	0.163±0.064
Firefox	0.277±0.091	0.154±0.068	0.193±0.058

$\frac{2}{3}$ examples are used for training while the remaining $\frac{1}{3}$ examples are used for testing. A single J4.8 decision tree, a RandomForests ensemble and an RF2Tree are trained on the training data set and tested on the test data set. Such a process is repeated a 100 times with different training-test data partitions, and the average error rates are summarized in Tables II–VI where the numbers following '±' are standard deviations[§].

Pairwise t-tests with 95% significance level indicate that:

(1) When e/a=0.5, RF2Tree is significantly better than single tree on all the experimental data sets except on *credit-a* where there is no significant difference; the RF2Tree is slightly better than or comparable to the RandomForests on 12 data sets including *breast-cancer*, *breast-w*, *credit-g*, *diabetes*, *heart-statlog*, *hepatitis*, *kr-vs-kp*, *sick*, *sick-euthyroid*, *vote*, *wpbc*, and *Eclipse*.
(2) When $e/a = 0.8$, the RF2Tree is significantly better than the single tree on all the experimental data sets, and slightly better than or comparable to the RandomForests on seven data sets including *breast-cancer*, *breast-w*, *credit-g*, *hepatitis*, *rangeseg*, *vote*, and *wpbc*.
(3) When $e/a = 1.0$, the RF2Tree is significantly better than the single tree on all the experimental data sets, and slightly better than or comparable to RandomForests on eight data sets including *breast-cancer*, *breast-w*, *colic.ORIG*, *credit-g*, *diabetes*, *hepatitis*, *labor*, and *sick*.

[§] Some small values appear as 0.000 in Tables II–VI after rounding off.

DOI: 10.1002/spe

Table III. Hold-out test errors with $e/a = 0.8$ (100 runs).

Data set	Single tree	RandomForests	RF2Tree
breast-cancer	0.270±0.344	0.205±0.311	0.215±0.312
breast-w	0.160±0.225	0.070±0.152	0.070±0.152
colic	0.285±0.194	0.180±0.153	0.215±0.166
colic.ORIG	0.411±0.110	0.336±0.085	0.361±0.083
credit-a	0.300±0.233	0.222±0.194	0.248±0.199
credit-g	0.467±0.152	0.342±0.124	0.355±0.139
diabetes	0.463±0.263	0.343±0.278	0.367±0.294
heart-statlog	0.395±0.220	0.245±0.185	0.275±0.190
hepatitis	0.252±0.165	0.145±0.084	0.137±0.080
ionosphere	0.236±0.146	0.146±0.115	0.208±0.133
kr-vs-kp	0.385±0.172	0.279±0.131	0.312±0.135
labor	0.338±0.200	0.216±0.157	0.246±0.160
mushroom	0.160±0.155	0.067±0.104	0.087±0.125
rangeseg	0.328±0.181	0.212±0.139	0.226±0.132
sick	0.052±0.108	0.000±0.000	0.006±0.027
sick-euthyroid	0.209±0.116	0.143±0.000	0.157±0.056
sonar	0.382±0.109	0.226±0.089	0.314±0.108
vote	0.110±0.178	0.063±0.131	0.067±0.134
wdbc	0.183±0.136	0.078±0.089	0.121±0.107
wpbc	0.367±0.147	0.248±0.085	0.253±0.079
Eclipse	0.327±0.119	0.193±0.097	0.227±0.090
Firefox	0.224±0.077	0.136±0.060	0.192±0.060

(4) When $e/a = 1.2$, the RF2Tree is significantly better than the single tree on all the experimental data sets, and slightly better than or comparable to the RandomForests on 10 data sets including *breast-cancer*, *breast-w*, *credit-g*, *diabetes*, *hepatitis*, *rangeseg*, *sick*, *sick-euthyroid*, *vote*, and *wpbc*.
(5) When $e/a = 1.5$, the RF2Tree is significantly better than the single tree on all the experimental data sets except on *mushroom* there is no significant difference; the RF2Tree is slightly better than or comparable to the RandomForests on eight data sets including *breast-cancer*, *breast-w*, *credit-a*, *heart-statlog*, *hepatitis*, *labor*, *sick-euthyroid*, and *wpbc*.

The above observations confirm that the RF2Tree can be used to deal with extremely small data sets.

Note that the RandomForests is an ensemble whose comprehensibility is poor. If we only want to get a high predictive accuracy, the RandomForests may be sufficient; but if we want to study which factor is important (this is our purpose), RandomForests could not help as it is a black-box model, while the RF2Tree is more helpful for this purpose.

3. MINING PRACTICE

3.1. Data set

In order to find out the key factors affecting the success of software reuse, Morisio *et al.* [3] employed a well-documented interview process to investigate 24 European projects from 19

DOI: 10.1002/spe

Table IV. Hold-out test errors with $e/a = 1.0$ (100 runs).

Data set	Single tree	RandomForests	RF2Tree
breast-cancer	0.437±0.235	0.363±0.196	0.367±0.192
breast-w	0.130±0.165	0.060±0.107	0.062±0.125
colic	0.318±0.163	0.219±0.145	0.284±0.155
colic.ORIG	0.360±0.105	0.295±0.072	0.303±0.067
credit-a	0.294±0.215	0.190±0.171	0.238±0.188
credit-g	0.400±0.141	0.289±0.127	0.304±0.112
diabetes	0.420±0.275	0.293±0.265	0.317±0.261
heart-statlog	0.342±0.193	0.218±0.201	0.250±0.198
hepatitis	0.265±0.169	0.152±0.092	0.152±0.098
ionosphere	0.251±0.129	0.170±0.106	0.225±0.126
kr-vs-kp	0.319±0.159	0.211±0.129	0.256±0.131
labor	0.267±0.183	0.173±0.136	0.187±0.135
mushroom	0.168±0.138	0.089±0.109	0.121±0.138
rangeseg	0.297±0.160	0.183±0.133	0.205±0.129
sick	0.140±0.074	0.100±0.000	0.102±0.025
sick-euthyroid	0.076±0.114	0.001±0.012	0.014±0.043
sonar	0.370±0.102	0.222±0.099	0.322±0.096
vote	0.115±0.120	0.078±0.104	0.088±0.110
wdbc	0.150±0.121	0.080±0.093	0.113±0.115
wpbc	0.365±0.112	0.267±0.070	0.284±0.080
Eclipse	0.237±0.100	0.140±0.084	0.171±0.093
Firefox	0.237±0.070	0.152±0.063	0.205±0.059

companies in the period 1994–1997. Here 27 variables were used to describe each project, including 10 state variables representing attributes over which a company has no control, six high-level control variables representing key high-level management decisions about a reuse program, 10 low-level control variables representing specific approaches to the implementation of reuse, and a variable indicating whether the project was a success or not (Menzies and DiStefano [7] provided a nice summarization of these variables). Morisio *et al*. presented their entire data set in their paper. Although the data set contains only 24 examples, it is the largest empirical data set on software reuse at present.

Later, Menzies and Di Stefano employed several data mining tools to analyze the data set [7]. The tools they used include the APRIORI association rule learner, the J4.8 decision tree, and the TAR2 treatment leaner. Besides some coincided conclusions, Menzies and Di Stefano reported that there were numerous discrepancies between the mining results and the conclusions drawn by Morisio *et al.* In particular, they claimed that there were more factors, such as the *Size of Baseline*, affecting the success of software reuse. Morisio *et al*. [8], however, expressed negative opinion to Menzies and Di Stefano's new findings.

Morisio *et al*.'s results were mainly derived from their expert experiences, while Menzies and Di Stefano's results were obtained through data mining techniques. Thus, if Morisio *et al*. were wrong, it may suggest that expert experiences are not as reliable as it might have been expected; if Menzies and Di Stefano were wrong, it may suggest that data mining and machine learning techniques are helpful in this area unless a far larger data set is available. Either of the consequences is meaningful.

DOI: 10.1002/spe

Table V. Hold-out test errors with $e/a = 1.2$ (100 runs).

Data set	Single tree	RandomForests	RF2Tree
breast-cancer	0.500±0.185	0.442±0.177	0.448±0.179
breast-w	0.140±0.156	0.045±0.097	0.045±0.097
colic	0.276±0.173	0.184±0.132	0.218±0.145
colic.ORIG	0.334±0.095	0.278±0.064	0.296±0.071
credit-a	0.270±0.171	0.196±0.136	0.226±0.156
credit-g	0.438±0.145	0.334±0.114	0.348±0.121
diabetes	0.387±0.258	0.293±0.243	0.303±0.247
heart-statlog	0.328±0.221	0.198±0.176	0.222±0.186
hepatitis	0.310±0.127	0.225±0.102	0.238±0.101
ionosphere	0.204±0.101	0.120±0.082	0.159±0.097
kr-vs-kp	0.271±0.143	0.186±0.128	0.211±0.124
labor	0.237±0.141	0.132±0.130	0.160±0.142
mushroom	0.125±0.135	0.050±0.081	0.068±0.093
rangeseg	0.280±0.167	0.183±0.129	0.197±0.128
sick	0.115±0.059	0.083±0.000	0.086±0.019
sick-euthyroid	0.169±0.124	0.093±0.029	0.102±0.053
sonar	0.352±0.096	0.226±0.073	0.310±0.097
vote	0.091±0.124	0.060±0.095	0.060±0.095
wdbc	0.133±0.104	0.076±0.074	0.093±0.091
wpbc	0.369±0.117	0.288±0.077	0.294±0.088
Eclipse	0.223±0.084	0.129±0.071	0.148±0.069
Firefox	0.224±0.060	0.148±0.051	0.192±0.053

In this section, the RF2Tree algorithm is used to mine this extremely small data set. Note that in the data set, the 23rd example is with missing values. Fortunately, as the RF2Tree utilizes decision trees that enable the use of training examples with missing values, all the 24 examples can be exploited in the mining process.

3.2. Mining process

Recall that two conditions must be satisfied for the RF2Tree to work well, i.e. the original training data set is really small, and the random forest is more accurate than the single decision tree if both of them were directly trained from the original training data set. We first check whether the two conditions are satisfied before trying to mine the data.

The first condition is obviously satisfied. In order to judge whether the second condition is satisfied, 100 runs of 16-8 hold-out test is performed. In detail, the original data set is randomly partitioned into two sets, where 16 examples are used for training while the remaining eight examples are used for testing. A single J4.8 decision tree, a RandomForests ensemble, and an RF2Tree are trained on the training data and then tested on the test data. To get a reliable estimate, the above process is repeated a 100 times and the average error rates are recorded in Table VII, where the numbers following '±' are standard deviations.

Table VII shows that when only the state variables and high-level control variables are used, the RandomForests performs better than the single decision tree and, therefore, the second condition

DOI: 10.1002/spe

Table VI. Hold-out test errors with $e/a = 1.5$ (100 runs).

Data set	Single tree	RandomForests	RF2Tree
breast-cancer	0.362±0.202	0.260±0.176	0.268±0.171
breast-w	0.114±0.131	0.036±0.087	0.038±0.089
colic	0.273±0.141	0.204±0.119	0.225±0.126
colic.ORIG	0.372±0.062	0.336±0.045	0.353±0.042
credit-a	0.248±0.184	0.189±0.155	0.205±0.159
credit-g	0.403±0.140	0.275±0.102	0.302±0.111
diabetes	0.454±0.207	0.342±0.176	0.366±0.199
heart-statlog	0.346±0.175	0.236±0.135	0.247±0.148
hepatitis	0.271±0.129	0.170±0.087	0.181±0.087
ionosphere	0.177±0.090	0.097±0.061	0.145±0.084
kr-vs-kp	0.209±0.110	0.145±0.083	0.182±0.087
labor	0.259±0.143	0.166±0.120	0.176±0.118
mushroom	0.103±0.098	0.052±0.061	0.087±0.082
rangeseg	0.221±0.110	0.141±0.101	0.174±0.103
sick	0.105±0.083	0.063±0.017	0.072±0.035
sick-euthyroid	0.198±0.069	0.145±0.024	0.148±0.043
sonar	0.351±0.077	0.213±0.074	0.310±0.085
vote	0.084±0.107	0.056±0.086	0.062±0.087
wdbc	0.127±0.087	0.071±0.067	0.101±0.086
wpbc	0.351±0.095	0.255±0.082	0.265±0.074
Eclipse	0.207±0.075	0.139±0.073	0.159±0.057
Firefox	0.196±0.062	0.128±0.050	0.165±0.059

Table VII. 16-8 Hold-out test errors (100 runs).

Variables	Single decision tree	RandomForests	RF2Tree
State and high-level control	0.084±0.102	0.069±0.089	0.046±0.090
State and low-level control	0.325±0.179	0.185±0.140	0.211±0.143
State and all control	0.088±0.106	0.082±0.117	0.076±0.133

mentioned above is satisfied. In fact, on this data set the error rates of the RF2Tree is even lower than that of the RandomForests. Thus, using the RF2Tree to analyze the data set using only state variables and high-level control variables is a good choice.

On the other hand, Table VII shows that when only the state variables and low-level control variables are used, although the RandomForests performs better than the single decision tree, their error rates are relatively poor. Moreover, by comparing the first and third rows of Table VII, it can be found that after including the low-level control variables, the performances of all the learners deteriorate. Therefore, we conclude that these low-level control variables in this data set do not convey helpful information about the success or failure of software reuse.

As the above test confirms the usefulness of the RF2Tree in analyzing the data set, we start to mine the data as follows: Now the whole original data set with only state and high-level control variables is used as the training data set for the RF2Tree. At the root node of the final J4.8 decision tree, the

DOI: 10.1002/spe

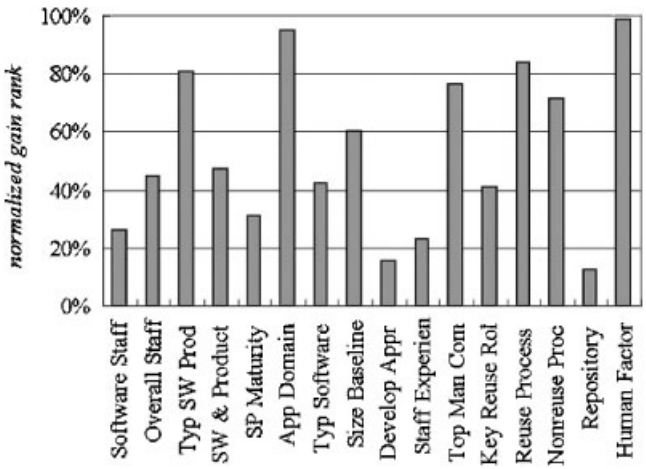

Figure 1. Normalized rank of information gain of state and high-level control variables.

importance of the variables is measured. Here, we normalize the information gain[¶] values of the variables by dividing them by the largest information gain value, and therefore obtain the *normalized information gain*. Moreover, we assign the largest rank to the variable with the largest information gain value, and therefore obtain the *normalized rank* of information gain after normalizing. Note that as both the RandomForests and the scheme of generating virtual training examples in the RF2Tree have some randomness, the above process is repeated a 100 times again. Tests on the original data set show that all the 100 RF2Trees achieved 100% accuracy on these 24 examples. The average normalized rank and normalized information gain are shown in Figures 1 and 2. The figures clearly indicate that *Human Factors*, *Reuse Process Introduced*, *Type of Software Production*, *Application domain*, *Top Management Commitment*, and *Nonreuse Process Modified* are important factors affecting the success or failure of software reuse.

Note that in order to increase the reliability of the results, our experiments have been repeated for 100 times. If a variable occasionally achieves an extremely high information gain value, its normalized information gain might be significantly increased while its normalized rank will be relatively robust to such occasional perturbation. On the other hand, variables with trivial differences in the normalized rank might have significant differences in the normalized information gain. Therefore, Figures 1 and 2 should be read together whereas the conclusion drawn from a single figure might be misleading.

Moreover, in addition to information gain, several other measures, including Gini index [20], DKM [21], and MDL [22] are exploited in the same manner as that described above. The average normalized ranks and normalized measures are shown in Figures 3–8, respectively. It is interesting that the discovered patterns are very stable because all these figures indicate that *Human Factors*, *Reuse Process Introduced*, *Type of Software Production*, *Application domain*, *Top Management*

[¶] In fact, J4.8 uses *information gain ratio*, which is a simple variant of information gain, to select variables for splitting the nodes.

DOI: 10.1002/spe

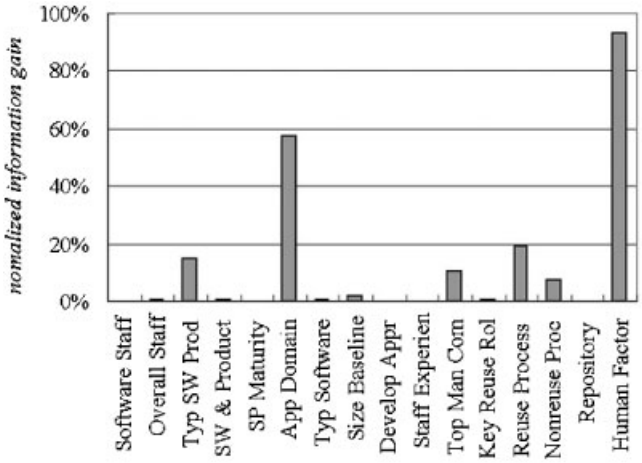

Figure 2. Normalized information gain of state and high-level control variables.

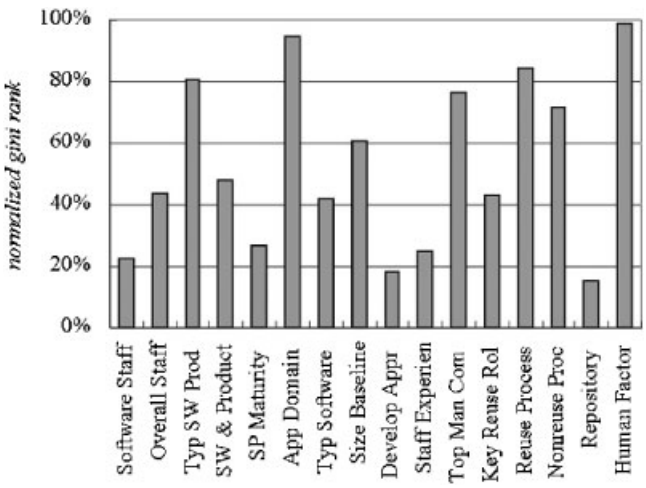

Figure 3. Normalized rank of Gini index of state and high-level control variables.

Commitment, and *Non-reuse Processes Modified* are important factors affecting the success or failure of software reuse.

3.3. Discussion

Our analysis results are summarized in Table VIII. For comparison, Morisio *et al.*'s results [3] and Menzies and Di Stefano's results [7] are also included in the table. Here, *partially* denotes the fact that the importance of these variables were concluded from Morisio *et al.*'s expert experiences.

DOI: 10.1002/spe

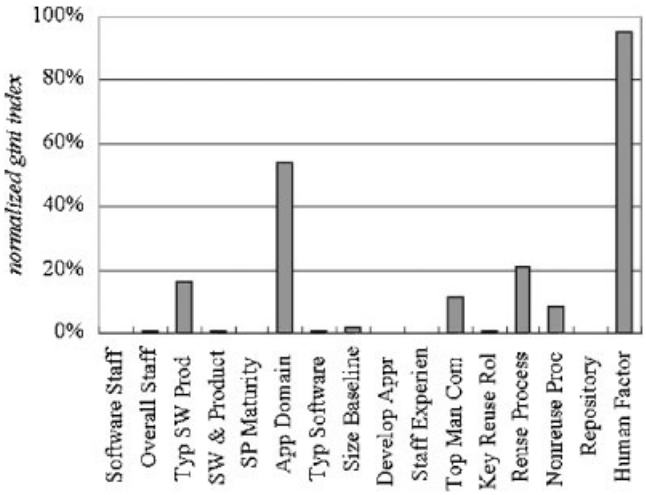

Figure 4. Normalized Gini index of state and high-level control variables.

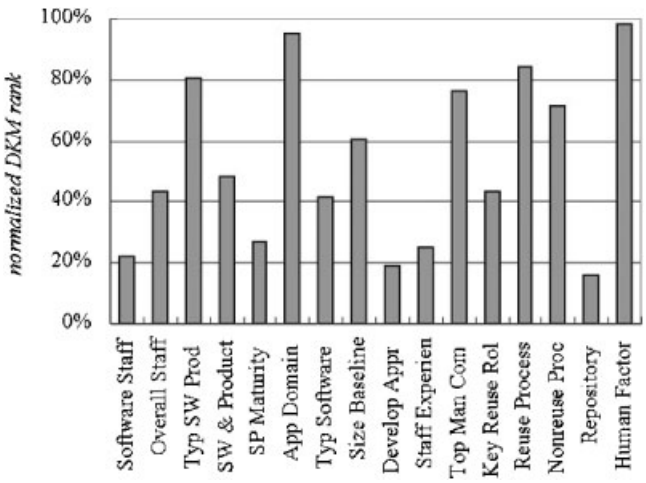

Figure 5. Normalized rank of DKM of state and high-level control variables.

Very weak and *barely* were used by Menzies and Di Stefano [7] to express their feeling that the data set offers little evidence that these variables are important. For example, they reported that *Top Management Commitment* was not found to be predictive or associated with anything else by their data mining tools [7]. *Intuitively* denotes that although we believe that these variables are important, our analysis does not show that the data set offers enough evidence.

As for state variables, Morisio *et al*. [3] found that only *Type of Software Production* has an effect on the success or failure of software reuse, while Menzies and Di Stefano [7] thought that the effect of this variable is 'dubious' because in the decision tree they preferred (the one shown

DOI: 10.1002/spe

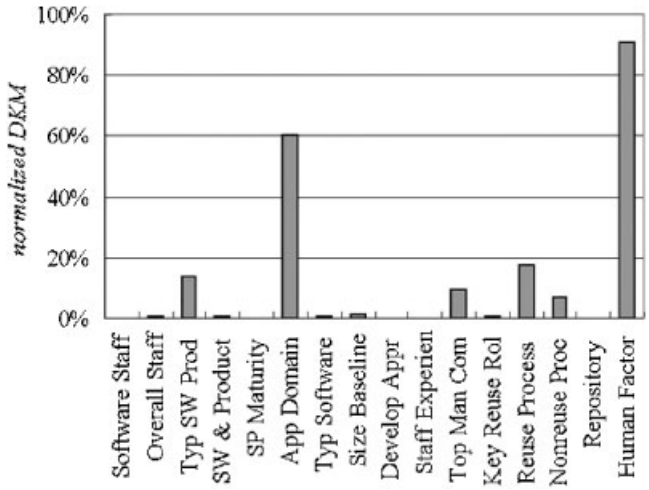

Figure 6. Normalized DKM of state and high-level control variables.

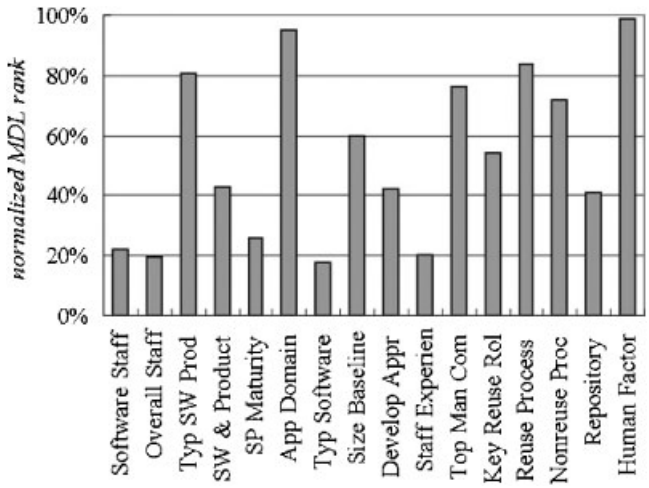

Figure 7. Normalized rank of MDL of state and high-level control variables.

in Figure 9(b)) that there is only a node denoting *Human Factors*. Our analysis supports Morisio *et al*.'s claim and we believe that Menzies and Di Stefano's claim suffers from the fact that their decision tree was grown directly from the small original data set. In fact, when we test single J4.8 decision trees in the 100 runs of 16-8 hold-out test process, eight kinds of decision trees have been observed. The tree shown in Figure 9(a) appears in $\frac{49}{100}$ runs while the one shown in Figure 9(b) appears in $\frac{38}{100}$ runs. Therefore, we think that Figure 9(a) is more reliable. Note that both the trees

DOI: 10.1002/spe

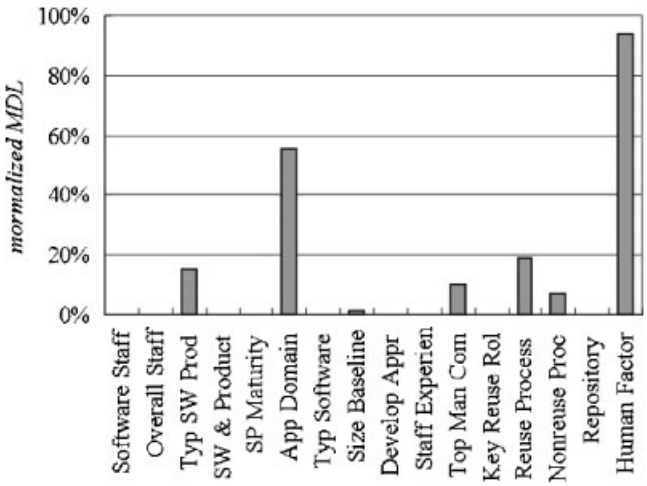

Figure 8. Normalized MDL of state and high-level control variables.

coincide with the fact that *Human Factors* is important to the success or failure of software reuse, and the only difference lies in that the tree shown in Figure 9(a) indicates that *Type of Software Production* should be considered when *Human Factors* is *yes*.

Menzies and Di Stefano [7] found that *Size of Baseline* is an important variable because 100% of the eight projects where *Size of Baseline* was 'large' were judged to be reuse successes. However, this interpretation was denied by Morisio *et al*. [8] and they indicated that this variable, which denotes the size of the delivered project, is not the reuse factor and has no relationship with the reuse factor.

Our analysis does not identify *Size of Baseline* as an important variable, but the analysis implies that *Application Domain* is important (as shown in Figures 1–8). In fact, in the data set all the projects in application domains of bank, engine controller, air traffic control, train simulation, manufacturing, management and control of measurement environment and finance were successful, while the projects in application domains of flight management systems, aerospace applications, and book-keeping were failed. We think that there might be two alternative interpretations. The first is that *Application Domain* is actually important because different application domains might have different requirements on reuse, which might affect the reuse process. The second is that our analysis method has been 'cheated' by *Application Domain*, which is the variable with the most number of values in the data set. This is because the scheme of generating virtual training examples employed by the RF2Tree tends to uniformly sample each variable and, therefore, the RF2Tree might be biased to variables with a lot of values. These two interpretations should be judged by experts in software reuse. But we believe that if more real data are available, we can also use machine learning and data mining techniques to determine which interpretation is correct.

As for high-level control variables, Table VIII shows that although Menzies and Di Stefano [7] concluded that the data set offers little evidence for *Top Management Commitment*, *Reuse Processes Introduced*, and *Non-reuse Processes Modified*, our analysis finds that the evidences are actually quite strong (as shown in Figures 1–8). This fact coincides with Morisio *et al*.'s analysis [3].

DOI: 10.1002/spe

Table VIII. Conclusions made on the relevance of the variables to the success or failure of software reuse.

Variables	Morisio *et al.*	Menzies and Di Stefano	This paper
State variables			
Software staff	×	×	×
Overall staff	×	×	×
Type of software production	√	×	√
Software and product	×	×	×
SP maturity	×	×	×
Application domain	Not analyzed	×	√
Type of software	×	×	×
Size of baseline	Not analyzed	√	×
Development approach	×	×	×
Staff experience	×	×	×
High-level control variables			
Top management commitment	√	√(Barely)	√
Key reuse roles introduced	√(Partially)	√(Very weak)	√(Intuitively)
Reuse processes introduced	√	√(Barely)	√
Non-reuse processes modified	√	√(Very weak)	√
Repository	√(Partially)	√	√(Intuitively)
Human factors	√	√	√
Low-level control variables			
Reuse approach	×	√	×
Work products	×	×	×
Domain analysis	×	√	×
Origin	×	×	×
Independent team	×	×	×
When assets developed	×	×	×
Qualification	×	×	×
Configuration management	×	×	×
Rewards policy	×	×	×
# Assets	×	×	×

√/×: with/without evidence. The labels *partially*, *very weak*, *barely*, and *intuitively* are explained in the text.

Morisio *et al*. [3] argued that all the high-level control variables are important for a successful reuse program, but they also admitted that *Introducing Key Reuse Roles* and setting up *Repository* are not sufficient for successful reuse. This is because the importance of these two variables was only derived from their expert experiences. Therefore, it is not strange that our analysis has not identified these two important factors. In fact, all the investigated projects had a repository, and hence the importance of the variable *Repository* could not be disclosed by any data analysis technique. We are not sure how Menzies and Di Stefano [7] concluded with their data mining tools that this variable is important.

As for low-level control variables, Menzies and Di Stefano [7] believed that *Reuse Approach* and *Domain Analysis* are important. As shown in Table VII, as the accuracies of all the learners deteriorate after appending low-level control variables to the set of state and high-level control variables, we do not think that these low-level control variables are important; our results coincide with Morisio *et al*.'s conclusion [3].

DOI: 10.1002/spe

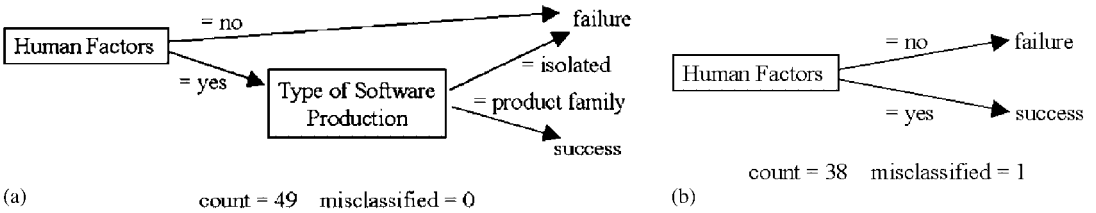

Figure 9. Two decision trees learned from the original data set: (a) the tree Morisio *et al*. preferred and (b) the tree Menzies and Di Stefano preferred.

4. CONCLUSION

There is no doubt that introducing empirical analysis into software engineering will be helpful. As collecting a large set of training examples is usually difficult in this area, machine learning, and data mining techniques, which are able to work well with a very small number of training examples are desirable. In this paper, we present the RF2Tree algorithm, which can be used to mine extremely small data sets. The mining practice on a software reuse data set illustrates the usefulness of the RF2Tree.

The small training set size has been identified as an important factor impacting the learning performance decades ago [23,24], but how to judge whether a data set is 'small' or not remains an open problem. Some previous work such as [24,25] provide some hints to this problem. It is evident that the number of examples that are sufficient is strongly related to the complexity of the task. During the past few years, many studies have been devoted to the measure of learning complexity, e.g. measuring the problem complexity by characterizing the geometrical complexity of the class boundary [26]. These studies may be helpful to judge the 'smallness' of data sets in the future.

Software reuse projects are usually developed in corporations, yet corporations are often very reluctant to give out their data. Thus, applying machine learning and data mining techniques to software reuse is very difficult beyond the data set contributed by Morisio *et al*. [3]. However, many other topics in software engineering can be investigated, such as software cost estimation [27,28]. These are interesting for future work.

It is worth noting that as we are mainly working on machine learning and data mining, and our knowledge on software reuse is very limited, our analysis presented in this paper is made from a data mining aspect. Thus, although our analysis suggests that on the concerned task the empirical data support the expert opinion well, the identified important factors in our analysis need to be judged by software reuse experts. Nevertheless, the work presented in this paper illustrates that recent advances in machine learning and data mining are ready to tackle real-world problems, and it is the time for software engineering researchers to embrace machine learning and data mining.

ACKNOWLEDGEMENTS

We want to thank the anonymous reviewers for their helpful comments and suggestions. We also want to thank John Anvik for providing the bug report data set. This research was supported by the National Science Foundation of China (60505013, 60635030, 60721002), the Jiangsu Science Foundation (BK2008018), and the National 973 Fundamental Research Program of China (2002CB312002).

DOI: 10.1002/spe

REFERENCES

1. Mjolsness E, DeCoste D. Machine learning for science: State of the art and future prospects. *Science* 2001; **293**(5537):2051–2055.
2. Poulin JS, Caruso JM, Hancock DR. The business case for software reuse. *IBM Systems Journal* 1993; **10**(4):567–594.
3. Morisio M, Ezran M, Tully C. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering* 2002; **28**(4):340–357.
4. Vapnik VN. *The Nature of Statistical Learning Theory*. Springer: Berlin, 1995.
5. Zhou Z-H, Jiang Y. Medical diagnosis with C4.5 rule preceded by artificial neural network ensemble. *IEEE Transactions on Information Technology in Biomedicine* 2003; **7**(1):37–42.
6. Breiman L. Random forests. *Machine Learning* 2001; **45**(1):5–32.
7. Menzies T, Di Stefano JS. More success and failure factors in software reuse. *IEEE Transactions on Software Engineering* 2003; **29**(5):474–477.
8. Morisio M, Ezran M, Tully C. Comments on more success and failure factors in software reuse. *IEEE Transactions on Software Engineering* 2003; **29**(5):478.
9. Zhou Z-H. Ensemble learning. *Encyclopedia of Biometrics*, Li SZ (ed.). Springer: Berlin, 2008.
10. Jiang Y, Li M, Zhou Z-H. *Generation of Comprehensible Hypotheses from Gene Expression Data* (*Lecture Notes in Bioinformatics*, vol. 3916). Springer: Berlin, 2006; 116–123.
11. Zhou Z-H, Jiang Y. NeC4.5: Neural ensemble based C4.5. *IEEE Transactions on Knowledge and Data Engineering* 2004; **16**(6):770–773.
12. Witten IH, Frank E. *Data Mining*: *Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann: San Francisco, CA, 2000.
13. Domingos P. Knowledge acquisition from examples via multiple models. *Proceedings of the 14th International Conference on Machine Learning*, Nashville, TN, 1997; 98–106.
14. Zhou Z-H, Jiang Y, Chen S-F. Extracting symbolic rules from trained neural network ensembles. *AI Communications* 2003; **16**(1):3–15.
15. Rao NSV. On fusers that perform better than best sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2001; **23**(8):904–909.
16. Wolpert DH, Macready WG. No free lunch theorems for search. *Technical Report*, Santa Fe Institute, Santa Fe, NM, February 1995.
17. Wolpert DH, Macready WG. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1997; **1**(1):67–82.
18. Blake C, Keogh E, Merz CJ. UCI repository of machine learning databases. Available from: http://www.ics.uci.edu/∼mlearn/MLRepository.html, Department of Information and Computer Science, University of California, Irvine, CA, 1998.
19. Anvik J, Hiew L, Murphy GC. Who should fix this bug? *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, 2006; 361–370.
20. Breiman L, Friedman JH, Olshen RA, Stone CJ. *Classification and Regression Trees*. Wadsworth: Belmont, CA, 1984.
21. Dietterich TG, Kearns M, Mansour Y. Applying the weak learning framework to understand and improve C4.5. *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, 1996; 96–104.
22. Kononenko I. On biases in estimating multi-valued attributes. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995; 1034–1040.
23. Jain AK, Chandrasekaran B. Dimensionality and sample size considerations in pattern recognition practice. *Handbook of Statistics*, vol. 2, Krishnaiah PR, Kanal LN (eds.). North-Holland: Amsterdam, 1987; 835–855.
24. Raudys SJ, Jain AK. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1991; **13**(3):252–264.
25. Raudys SJ. *Statistical and Neural Classifiers*: *An Integrated Approach to Design*. Springer: Berlin, 2001.
26. Ho TK, Basu M. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2002; **24**(3):289–300.
27. Chulani S, Boehm B, Steece B. Calibrating software cost models using Bayesian analysis. Available from: http://sunset.usc.edu/Research_Group/Sunita/down/tse.pdf, 2004.
28. Menzies T, Port D, Chen Z, Hihn J, Stukes S. Validation methods for calibrating software effort models. Available from: http://menzies.us/pdf/04coconut.pdf, 2004.

DOI: 10.1002/spe