International Journal of Software Engineering
and Knowledge Engineering
Vol. 22, No. 2 (2012) 203−223

DOI: 10.1142/S0218194012400037

RESAMPLING METHODS IN SOFTWARE QUALITY CLASSIFICATION

WASIF AFZAL

Department of Graduate Studies and Applied Sciences
Bahria University, Islamabad, Pakistan
wasif.afzal@gmail.com

RICHARD TORKAR* and ROBERT FELDT†

Blekinge Institute of Technology
371 79, Karlskrona, Sweden
**richard.torkar@bth.se*
†*robert.feldt@bth.se*

In the presence of a number of algorithms for classification and prediction in software engineering, there is a need to have a systematic way of assessing their performances. The performance assessment is typically done by some form of partitioning or resampling of the original data to alleviate biased estimation. For predictive and classification studies in software engineering, there is a lack of a definitive advice on the most appropriate resampling method to use. This is seen as one of the contributing factors for not being able to draw general conclusions on what modeling technique or set of predictor variables are the most appropriate. Furthermore, the use of a variety of resampling methods make it impossible to perform any formal meta-analysis of the primary study results. Therefore, it is desirable to examine the influence of various resampling methods and to quantify possible differences. ***Objective and method*:** This study empirically compares five common resampling methods (hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping) using 8 publicly available data sets with genetic programming (GP) and multiple linear regression (MLR) as software quality classification approaches. Location of (PF, PD) pairs in the ROC (receiver operating characteristics) space and area under an ROC curve (AUC) are used as accuracy indicators. ***Results*:** The results show that in terms of the location of (PF, PD) pairs in the ROC space, bootstrapping results are in the preferred region for 3 of the 8 data sets for GP and for 4 of the 8 data sets for MLR. Based on the AUC measure, there are no significant differences between the different resampling methods using GP and MLR. ***Conclusion*:** There can be certain data set properties responsible for insignificant differences between the resampling methods based on AUC. These include imbalanced data sets, insignificant predictor variables and high-dimensional data sets. With the current selection of data sets and classification techniques, bootstrapping is a preferred method based on the location of (PF, PD) pair data in the ROC space. Hold-out validation is not a good choice for comparatively smaller data sets, where leave-one-out cross-validation (LOOCV) performs better. For comparatively larger data sets, 10-fold cross-validation performs better than LOOCV.

Keywords: Resampling methods; genetic programming; multiple regression; prediction; classification.

1. Introduction

Different dependent variables of interest have been the target of predictive studies in software engineering. Software quality classification is one such domain which concerns classifying software modules as either fault-prone (fp) or non-fault prone (nfp). A fault prone module is one in which the number of faults are higher than a selected threshold. Such a classification of software modules potentially has an effect on overall software quality since fault-prone modules are candidates for further reliability enhancement. Supervised learning algorithms from machine learning literature represent one of the relatively newer approaches for software quality classification whereby an induction algorithm builds a classifier from a given data set. Examples of such studies include applications of artificial neural networks, e.g. [15], classification and regression trees CART, e.g. [16], support vector machines, e.g. [37] and evolutionary computation, e.g. [24, 1].

With the availability of numerous techniques for constructing classification models, an important task in quality classification is appropriate model selection and evaluation. There are several key questions to answer in achieving this task, e.g.,

(1) What resampling method to use?
(2) What prediction accuracy measure to use?
(3) What statistical tests to use to compare the results?

While each of these questions are important, the focus of this study is to answer the choice of a resampling method to use. Specifically, we attempt to investigate which resampling method performs better while using genetic programming and multiple linear regression as software classification techniques.

It is common in machine learning that a portion of a data set is used to test the performance of the trained classifier. With limited data, different resampling methods are used to assess a model's generalizability. The choice of a resampling method is an important element in an overall model selection procedure that has attracted little investigation. With lack of convergence across various software classification models, the researchers have highlighted the need for greater use of public data sets, appropriate accuracy indicators and statistical testing procedures; but the choice of a resampling method is surprisingly less emphasized in the past. However increasing concern to investigate the choice of resampling methods is now being raised. Myrtveit *et al.* [28] and Lessmann *et al.* [23] highlighted the need to examine the influence of resampling methods and to quantify possible differences. Kitchenham and Mendes [18] pointed at the importance of explicitly stating the resampling method chosen:

> "Another important issue is whether to compare with predictions based on the entire data set or predictions based on dividing the data into training and testing data sets. Most researchers agree that the latter technique is better, but if we use anything other than a simple leave-one-out procedures results are not auditable unless the specific data set partitions are defined."

Further highlighting the need of multiple training sets, Kirsopp and Shepperd [17] came to the conclusion:

> "The major conclusion of this paper is, however, that it is dangerous to make inferences concerning the accuracy of prediction systems based on a small number of sampled training sets."

Kitchenham *et al.* [19] also highlighted that a variety of resampling techniques used by different studies is one of the reasons that impedes a formal meta-analysis of the primary study results:

> "Some studies used independent holdout samples; others used different types of cross validation (e.g. 3-fold, 20-fold, leave-one-out cross validation) ... These differences made it impossible to perform any formal meta-analysis of the primary study results."

A recent systematic review comparing genetic programming (GP) with other methods of predictive studies [3] indicate that for studies applying GP for software fault prediction and software reliability growth modeling, it is not always clear which resampling method is used. This shows that while use of resampling methods is an unsettled matter in predictive studies in software engineering in general, it requires even more investigation when using GP as a prediction technique. In addition to GP, we add a baseline approach (multiple linear regression) to investigate the impact of resampling methods on classification accuracy. We therefore seek an answer to the following research question in this paper:

RQ: How do different resampling methods compare with respect to predicting fault-prone software components using genetic programming (GP) and multiple linear regression (MLR)?

The effect of resampling methods on GP evolution has been sporadically discussed in research but not in a manner as in this study. Ross [33] studied the effects of randomly sampled training data on program evolution in GP. The study concluded that GP performance is better when the samples are more representative of the target behavior. The study did not empirically evaluate different resampling methods but highlighted a need of doing so:

> "The effects of re-sampling are not as clear, and further work is required to study the relationship between re-sampling rates and GP performance."

In this study, we use GP and MLR as a software quality classification approach and evaluate the influence of different resampling methods on the outcome of software quality classification. We present an extensive comparison between five common resampling methods: hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping using eight different publicly available data sets.

The rest of this paper is organized as follows: Section 2 presents relevant related studies. Section 3 presents the study design including an introduction to the different resampling methods used, an introduction to classification techniques used, the data sets used, performance estimation of classification accuracy and the experimental setup. The results are presented in Sec. 4, and discussed in Sec. 5. Validity issues make up Sec. 6 while conclusions appear in Sec. 7.

2. Related Work

Few comparisons of standard resampling methods have been performed in software engineering. Mittas and Angelis [26] used permutation tests and bootstrap to construct confidence intervals for the difference in accuracy measures for software cost prediction using regression and estimation by analogy. The emphasis of their study was not to find the best model but rather to recommend a systematic comparison of models using statistical hypothesis testing. Kirsopp and Shepperd [17] analyzed the influence of the number of training sets for software effort prediction using case-based prediction on two data sets. They evaluated the hold-out procedure and demonstrated that results may be misleading unless at least 5 different training sets, and preferable more than 20, are used. Green and Ohlsson [12] used artificial neural network ensembles to compare 5×5 fold cross-validation, 25-fold bootstrap and 25-fold hold-out using three cut-offs $(0.25, 0.50$ and $0.75)$. They showed that 5×5 fold cross-validation and hold-out with cut-offs 0.25 and 0.50 are the best resampling strategies for estimating the true performance of ANN ensembles. Kohavi [20] experimented with $C4.5$ and Naive-Bayesian classifier using 6 real-world data sets to compare 0.632 bootstrap and k-fold cross-validation with different values of k. They concluded that 10-fold stratified cross-validation is the best method for the data sets used. A study by Schaffer [34] concluded that on average 10-fold cross-validation strategy outperforms $C4.5$ decision trees and back-propagation neural networks. Molinaro *et al.* [27] used four classification algorithms (linear discriminant analysis, diagonal discriminant analysis, nearest neighbor and classification and regression trees (CART)) to compare different resampling methods. Among several conclusions, one of them was that leave-one-out cross-validation and 10-fold cross-validation had the smallest bias for diagonal discriminant analysis, nearest neighbor, CART and linear discriminant analysis.

While other references to resampling methods may be found in the literature, we have focussed above on more recent ones and their use in comparative studies.

3. Study Design

In this section we present an introduction to resampling methods, an introduction to GP and MLR, the data sets used, performance estimation of classification accuracy and the experimental setup.

3.1. *Resampling methods*

Resampling is an important concept in inferential statistics. It is used to draw a large number of samples from the original one and thus to reach an approximation of the underlying theoretical distribution. It is based on repeated sampling within the same data set [41].

Resampling is especially important for the validity of software engineering predictive studies since software engineering data sets are scarce and data limited. This has to do with difficulties in getting large data sets due to the data being confidential or where the data simply is too rudimentary in nature [2].

We examine the two most common resampling methods: cross-validation (repeated random sub-sampling, leave-one-out cross-validation and 10-fold cross validation) and bootstrapping. We also compare the split-sample or the hold-out method which acts as a baseline and an obvious split choice [39]. Due to space constraints, we refer the reader to a more detailed discussion on these methods in [7, 8, 36].

3.2. *Techniques used*

We used two techniques in this study: genetic programming (GP) and multiple linear regression (MLR). The parameter settings for the GP algorithm are given in Table 1. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} |e_i - e_i^*|$, where e_i is the actual classification, e_i^* is the estimated classification and n is the size of the data set used to train the GP models.

3.3. *Public domain data sets*

The data sets used in this study are taken from the PROMISE data repository [4] which is a collection of data sets freely available for performing predictive studies in software engineering. Specifically we make use of 8 data sets from the PROMISE repository namely AR6, AR1, PC1_req, JM1_req, CM1_req, AR3, AR4 and AR5.

Table 1. GP control parameters.

Control parameter	Value
Population size	30
Number of generations	100
Termination condition	100 generations
Function set	$\{+, -, *, \sin, \cos, \log\}$
Terminal set	$\{x\}$
Tree initialization	ramped half-and-half
Initial maximum number of nodes	28
Maximum number of nodes after genetic operations	512
Genetic operators	crossover, mutation, reproduction
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	lexictour
Elitism	replace

The reader is referred to PROMISE website [4] for a detailed information about these data sets.

3.4. *Performance estimation of classification accuracy*

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 fault, rather than the total number of faults.

We use the area under the receiver operating characteristic (ROC) curve (AUC) and the location of (PF, PD) pairs in the ROC space as a measure of classification performance for the different resampling methods. As such, if we can divide the ROC space into four regions as shown in Fig. 1, 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 [25].

3.5. *Experimental setup*

For each sample of an individual data set for each resampling method (except LOOCV), the GP and MLR are run 10 times. The best GP individual and the best MLR equation from the 10 runs are chosen for each sample of an individual data set. The sample statistics (AUC, PF, PD) were calculated for each of these best results and then averaged.

For LOOCV, running GP and MLR 10 times for each sample of an individual data set was not feasible due to high computational times, therefore GP and MLR were run once for each leave-one out sample of a particular data set. The sample statistics (AUC, PF, PD) from each leave-one-out samples were then averaged.

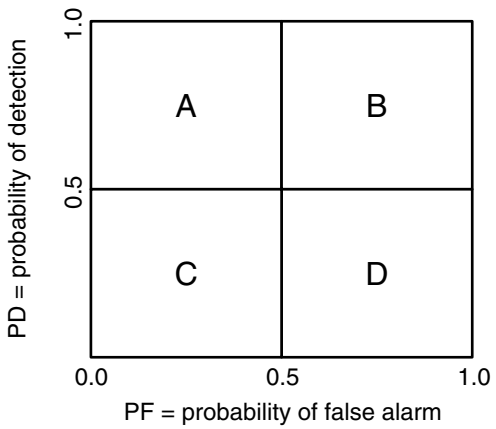


Fig. 1. Four regions in the ROC space.

For hold-out validation, each data set is randomly split into a training set (2/3 of the data) and a testing set (1/3 of the data).

4. Results

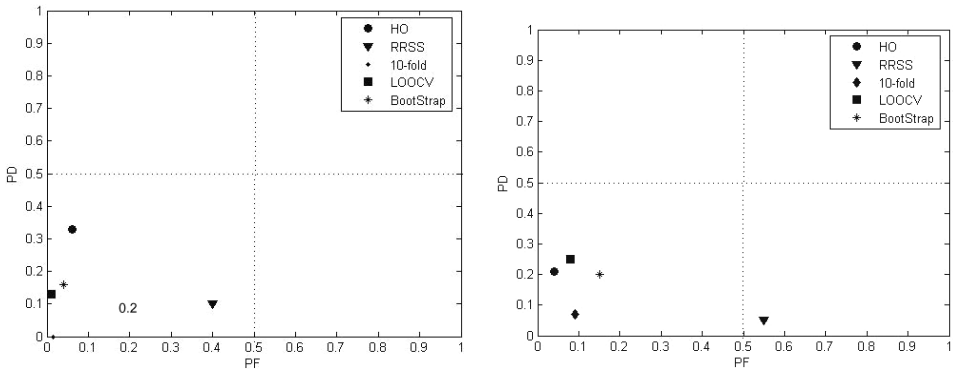
In this section we present the results of the empirical comparison in terms of (PF, PD) pair data in the ROC space and the AUC. All the results are based on the AUC, PF and PD values that represent the average over all the sub-samples (except for the hold-out validation where we have a single split of data).

4.1. *AR6 data set*

Table 2 shows the (PF, PD) pairs for the AR6 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 2(a) and 2(b). In the case of GP, for all the resampling methods, the (PF, PD) pairs are in the region C of the ROC space but hold-out and bootstrap resampling methods tend to have comparatively higher PD and lower PF values which is desirable. In the case of MLR, none of the resampling methods have the (PF, PD) pairs in the preferred region of the ROC space.

Table 2. (PF, PD) pair data for the AR6 data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0.33	0.06	0.21	0.04
Repeated random sub-sampling validation	0.10	0.40	0.05	0.55
10-fold cross-validation	0	0.01	0.07	0.09
Leave-one-out cross-validation	0.13	0.01	0.25	0.08
Bootstrapping	0.16	0.04	0.2	0.15



(a) GP (b) MLR

Fig. 2. (PF, PD) pair data for the AR6 data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

4.2. *AR1 data set*

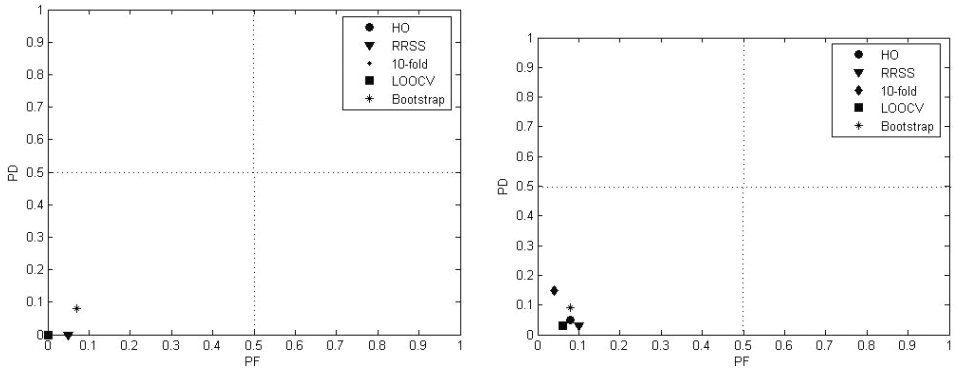
Table 3 shows the (PF, PD) pairs for the AR1 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 3(a) and 3(b). Again as with data set AR6, for GP, all the resampling methods have (PF, PD) pairs in the region C while bootstrap tends to have only a slightly better PF and PD values. For MLR, for all the resampling methods, the (PF, PD) pairs are in the region C of the ROC space which is undesirable.

4.3. *AR3 data set*

Table 4 shows the (PF, PD) pairs for the AR3 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 4(a) and 4(b). For GP, three resampling methods (hold-out, LOOCV and repeated random sub-sampling) have (PF, PD) pairs in the region C while bootstrap and 10-fold are in the preferred region A. For MLR, among all the resampling methods, bootstrapping and 10-fold CV have the (PF, PD) pairs in the desirable region A of the ROC space.

Table 3. (PF, PD) pair data for the AR1 data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0	0	0.05	0.08
Repeated random sub-sampling validation	0	0.05	0.03	0.1
10-fold cross-validation	0	0	0.15	0.04
Leave-one-out cross-validation	0	0	0.03	0.06
Bootstrapping	0.07	0.08	0.09	0.08

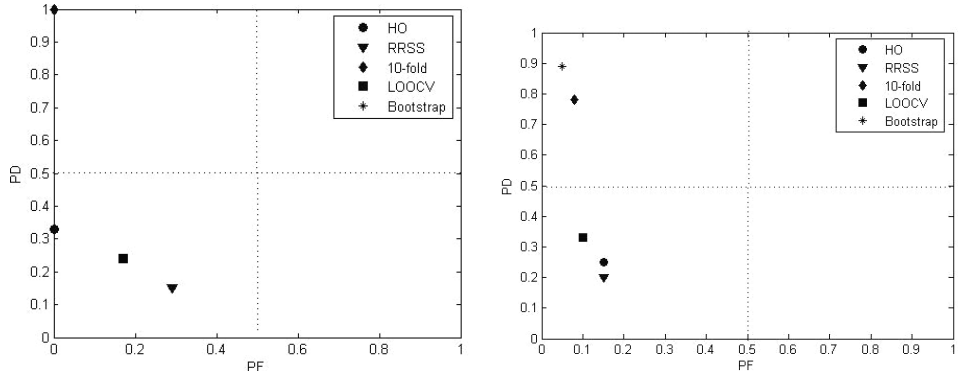


(a) GP (b) MLR

Fig. 3. (PF, PD) pair data for the AR1 data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

Table 4. (PF, PD) pair data for the AR3 data.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0.33	0	0.25	0.15
Repeated random sub-sampling validation	0.15	0.29	0.20	0.15
10-fold cross-validation	1	0	0.78	0.08
Leave-one-out cross-validation	0.24	0.17	0.33	0.10
Bootstrapping	1	0	0.89	0.05



(a) GP (b) MLR

Fig. 4. (PF, PD) pair data for the AR3 data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

4.4. *AR4 data set*

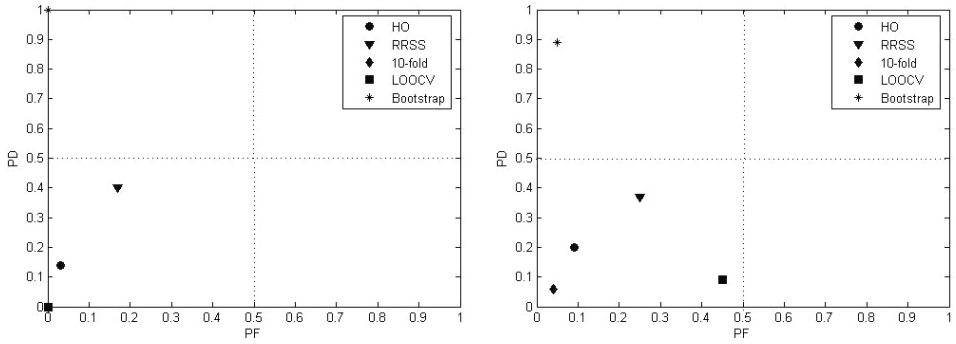
Table 5 shows the (PF, PD) pairs for the AR4 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 5(a) and 5(b). For GP, all of the resampling methods, except bootstrap, have (PF, PD) pairs in the region C. Bootstrap is in the preferred region A. For MLR, only bootstrap has the (PF, PD) pair in the preferred region A of the ROC space.

4.5. *AR5 data set*

Table 6 shows the (PF, PD) pairs for the AR5 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 6(a) and 6(b). For GP, all of the resampling methods, except repeated random sub-sampling, have (PF, PD) pairs in the region C. Repeated random sub-sampling is in the preferred region A. For MLR, only repeated random sub-sampling has the (PF, PD) pair in the preferred region A of the ROC space.

Table 5. (PF, PD) pair data for the AR4 data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0.14	0.03	0.20	0.09
Repeated random sub-sampling validation	0.4	0.17	0.37	0.25
10-fold cross-validation	0	0	0.06	0.04
Leave-one-out cross-validation	0	0	0.09	0.45
Bootstrapping	1	0	0.89	0.05



(a) GP (b) MLR

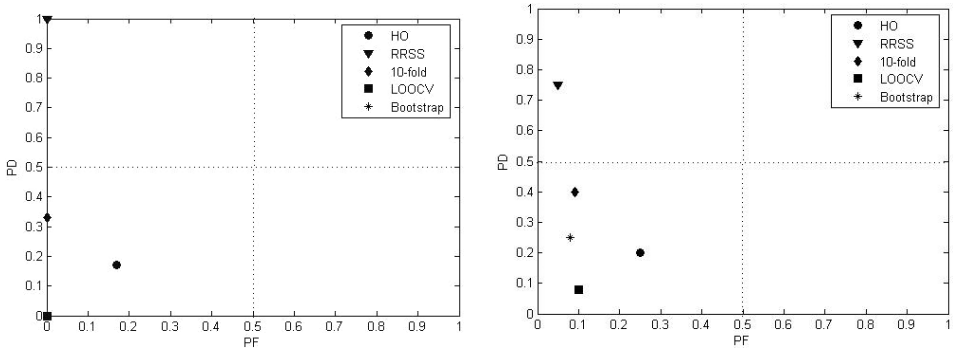
Fig. 5. (PF, PD) pair data for the AR4 data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

Table 6. (PF, PD) pair data for the AR5 data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0.17	0.17	0.20	0.25
Repeated random sub-sampling validation	1	0	0.75	0.05
10-fold cross-validation	0.33	0	0.40	0.09
Leave-one-out cross-validation	0	0	0.08	0.10
Bootstrapping	0.33	0	0.25	0.08

4.6. *PC1_req data set*

Table 7 shows the (PF, PD) pairs for the PC1_req data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 7(a) and 7(b). For GP, the only method not in region C is bootstrap with (PF, PD) pair of $(0, 0.62)$ that places it in region A. For MLR, bootstrap has the (PF, PD) pair in the preferred region A of the ROC space.

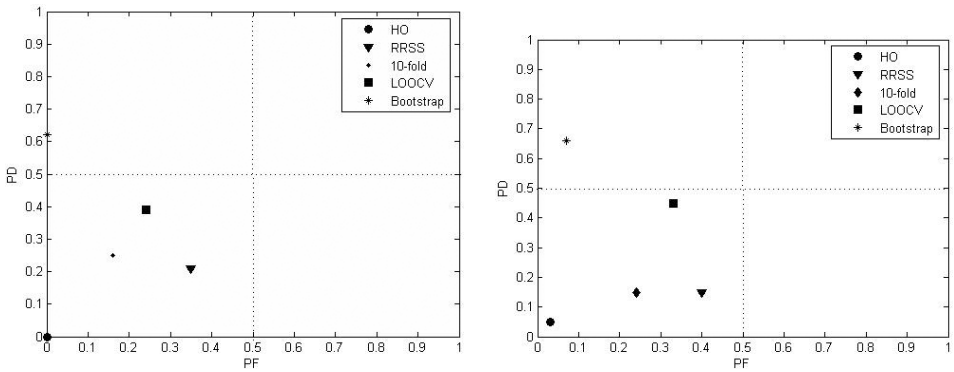


(a) GP (b) MLR

Fig. 6. (PF, PD) pair data for the AR5 data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

Table 7. (PF, PD) pair data for the PC1_req data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	0	0	0.05	0.03
Repeated random sub-sampling validation	0.21	0.35	0.15	0.40
10-fold cross-validation	0.25	0.16	0.15	0.24
Leave-one-out cross-validation	0.39	0.24	0.45	0.33
Bootstrapping	0.62	0	0.66	0.07



(a) GP (b) MLR

Fig. 7. (PF, PD) pair data for the PC1_req data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

4.7. *JM1_req data set*

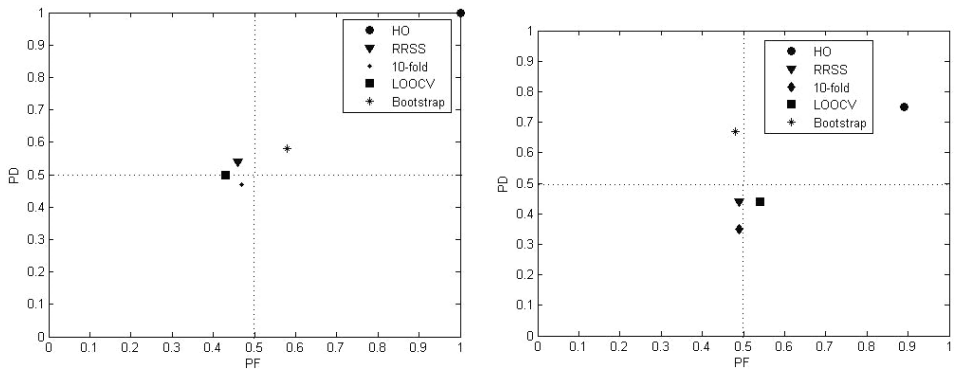
Table 8 shows the (PF, PD) pairs for the JM1_req data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 8(a) and 8(b). For GP, this time we see a wider spread of (PF, PD) pairs in the ROC space with repeated random sub-sampling validation having the (PF, PD) pair located in region A. Both hold-out validation and bootstrapping have high (PF, PD) values and are thus consequently placed in region B. For MLR, only bootstrap has the (PF, PD) pair in the preferred region A of the ROC space.

4.8. *CM1_req data set*

Table 9 shows the (PF, PD) pairs for the CM1_req data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figs. 9(a) and 9(b). For GP, we see here the concentration of (PF, PD) pairs within two regions of the ROC space. For 10-fold cross-validation and leave-one-out cross-validation, the (PF, PD) pairs lie in region A while for the rest of the resampling

Table 8. (PF, PD) pair data for the JM1_req data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	1	1	0.75	0.89
Repeated random sub-sampling validation	0.54	0.46	0.44	0.49
10-fold cross-validation	0.47	0.47	0.35	0.49
Leave-one-out cross-validation	0.50	0.43	0.44	0.54
Bootstrapping	0.58	0.58	0.67	0.48

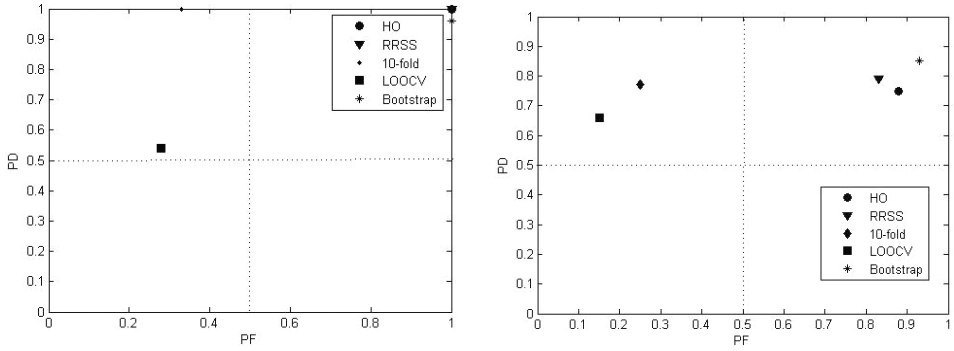


(a) GP (b) MLR

Fig. 8. (PF, PD) pair data for the JM1_req data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

Table 9. (PF, PD) pair data for the CM1_req data set.

	GP		MLR	
	PD	PF	PD	PF
Hold-out validation	1	1	0.75	0.88
Repeated random sub-sampling validation	1	1	0.79	0.83
10-fold cross-validation	1	0.33	0.77	0.25
Leave-one-out cross-validation	0.54	0.28	0.66	0.15
Bootstrapping	0.96	1	0.85	0.93



(a) GP (b) MLR

Fig. 9. (PF, PD) pair data for the CM1_req data set in the ROC space for GP and MLR. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

methods the (PF, PD) pairs are in region B. For MLR, LOOCV and 10-fold CV are in the preferred region A of the ROC space.

4.9. *AUC statistics*

Table 10 shows the empirical comparisons among the resampling methods in terms of mean AUC values for GP. The resampling methods providing the best AUC for a particular data set is highlighted in bold face. Bootstrapping results in higher AUC values for 3 data sets (AR1, AR5 and PC1_req) while 10-fold cross-validation and leave-one-out cross-validation results in the highest AUC value for two data sets each.

Bootstrapping AUC values suggest that it might be the most useful resampling method but we need to test for any significant differences in the AUC values. This is achieved by using the Kruskal−Wallis test which is a non-parametric alternative to analysis of variance. It is used to test the null hypothesis H_0 that k independent samples are from identical populations. The p-value came out to be 0.63 and therefore we cannot reject the null hypothesis (at $\alpha = 0.05$) that the samples are from identical populations.

Table 10. Hold-out test set results of 5 resampling methods in terms of the AUC for GP (RRSS is short for repeated random sub-sampling).

	Resampling methods				
Data set	Hold-out	RRSS	10-fold CV	LOOCV	Bootstrapping
AR1	0.5	0.48	0.5	0.49	**0.53**
AR3	0.60	0.5	0.54	**0.67**	0.60
AR4	0.55	0.54	**0.65**	0.60	0.55
AR5	0.5	0.66	0.60	0.54	**0.67**
AR6	**0.63**	0.46	0.49	0.59	0.56
PC1_req	0.5	0.5	0.5	0.53	**0.55**
JM1_req	0.5	0.57	**0.75**	0.53	0.49
CM1_req	0.5	0.52	0.42	**0.61**	0.5

Table 11. Hold-out test set results of 5 resampling methods in terms of the AUC for MLR (RRSS is short for repeated random sub-sampling).

	Resampling methods				
Data set	Hold-out	RRSS	10-fold CV	LOOCV	Bootstrapping
AR1	0.44	0.49	0.50	**0.55**	0.53
AR3	0.50	0.56	0.53	0.59	**0.61**
AR4	0.59	0.60	**0.62**	0.55	0.57
AR5	0.5	0.46	0.52	0.51	**0.55**
AR6	0.53	0.56	0.5	**0.59**	0.54
PC1_req	0.49	**0.52**	0.50	0.49	0.51
JM1_req	0.52	0.5	**0.63**	0.61	0.53
CM1_req	0.49	0.53	0.51	0.67	**0.71**

Table 11 shows the empirical comparisons among the resampling methods in terms of mean AUC values for MLR. The resampling methods providing the best AUC for a particular data set is highlighted in bold face. Bootstrapping results in higher AUC values for 3 data sets (AR3, AR5 and CM1_req) while 10-fold cross-validation and leave-one-out cross-validation results in the highest AUC value for two data sets each.

Again for MLR, bootstrapping AUC values suggest that it might be the most useful resampling method but we need to test for any significant differences in the AUC values. Using the Kruskal-Wallis test at $\alpha = 0.05$ the p-value turns out to be 0.08 which indicates that there is no significant differences between the AUC values from different resampling methods.

5. Analysis and Discussion

The results of the Wilcoxon rank sum tests represent an interesting outcome. Although there are a few differences between the AUC values for the different resampling methods they, however, do not differ significantly. There can be multiple

reasons for such an outcome and what we discuss here is intended to be suggestive rather than definitive:

Imbalanced data sets. Data sets having a high proportion of either fault-prone or non-fault prone modules would be likely to have non-significant performance outcomes no matter what resampling method is used. This is because the performance outcome of the dependent variable $y\varepsilon\{\text{fp} \mid \text{nfp}\}$ occurring most of the times would dominate the classification.

In our study AR1 data set has 6.61% of records representing fault-prone while CM1_req data set consists of 77.5% of records representing fault-prone. For JM1_req, with 45.94% of records representing fault-prone (thus representing a more balanced representation), 10-fold cross-validation is able to achieve an impressive AUC value of 0.75 for GP. This reasoning is also supported by a study by Kohavi [20] who recommends using stratification i.e. the folds are stratified so that they contain approximately the same proportions of the labels as the original data set.

Learning from imbalanced data sets is a well-known problem in machine learning. There are a number of proposed methods to resample the data in ways that diminish the effect of class imbalance. Over-sampling methods involve creating data for the minority class to reach a size close to that of the larger class while under-sampling methods eliminate larger class members to match the size of the smaller class. A study by Pelayo and Dick [30] show that over-sampling minority class examples improved the classification accuracy using *C*4.5 decision-tree classifier. While over-sampling methods might improve classification accuracy, one has to be mindful that creation of "synthetic" data might lack in appeal for a real-world use of a classification algorithm. Stratification *within* the data set might be a more useful alternative in this case.

Insignificant predictor variables. In case of a weak relationship between the predictor variables and the dependent variable, the performance outcomes are less dependent on the resampling methods used since the potential of these resampling methods would not be utilized optimally. The data sets PC1_req, JM1_req and CM1_req contain requirement metrics which showed a weak relationship in classifying fault-proneness in a study by Jiang *et al.*[14]. Therefore it is more likely that the resampling methods perform non-significantly on these data sets. For data sets AR6 and AR1 containing code attributes, the relationship with fault-proneness is present but is limited.

High dimensional data set. With data set having large features, feature selection is an important task. In this study the feature selection was not performed before running the GP algorithm, primarily to exploit GP's own feature selection capability [21]. However with high feature space the efficiency and effectiveness of GP (like any other machine learning technique) can dramatically drop [31]. This potentially minimizes the impact of a particular resampling method used.

We may conclude from above reasonings that the nature of the data sets play an important role in the classification which also has an influence on the performance of the resampling methods. Secondly the selection of a non-deterministic algorithm like GP is shown to have a minimal dependency on the resampling method chosen, though this behavior is also related to the actual nature of the data sets as already discussed above.

Moreover the sample size has an impact on the resampling methods. For comparatively smaller data sets (JM1_req, CM1_req) hold-out validation is clearly not a good choice due to bias resulting from a reduced training set size, this result being in agreement with the study by Green and Ohlsson [12]. LOOCV performs better in smaller data sets due to the optimum use of the training data (0.53 and 0.61 AUC values for JM1_req and CM1_req respectively for GP), while for larger data sets (AR6, AR1, PC1_req) it is interesting to find that there is not much difference between 10-fold cross-validation and LOOCV. Therefore 10-fold cross-validation might be more preferable to LOOCV for larger data sets. Such a choice would also guard against the potential large variance in the error estimate for LOOCV caused by a evaluating against only a single point in the test set [32].

In terms of location of (PF, PD) pairs in the ROC space, bootstrapping appears to be better placed. This indicates that bootstrapping should be considered as a resampling method for classification studies. In the context of software effort prediction studies, Kirsopp and Shepperd [17] argue that bootstrapping suffers from the disadvantage that resampling with replacement might not fit well with the real-world use where there will not be multiple copies of the same project. This argument however holds less for software fault prediction studies where certain independent variables relate to fault-proneness rather than project-level outcome, therefore it is more realistic to have multiple records of the same module within the scope of a single project. Bootstrapping, however, suffers from another potential drawback, i.e. the training and the test sets are not completely disjunct which is argued by some authors, e.g. [11], as an important consideration in defect prediction. It is still not known if this is the reason for relatively good performance of bootstrapping and should be investigated more.

In statistics, there are existing studies to show that bootstrapping offers significant improvements over cross-validation [10, 38]. It is fitting that the classification studies in software engineering learn from these positive results. It is also important that the use of bootstrapping (and its variants) are evaluated more for publicly available software engineering data sets. The work of Mittas and Angelis [26] is in the right direction and more such studies are required.

An important question to raise here is that whether or not are there differences in resampling methods when other techniques are used? For linear regression, it is shown in [5] that k-fold cross-validation performs better than LOOCV for model selection and evaluation; while Bootstrap gave an edge in terms of model evaluation when compared with cross-validation. Using decision trees and Naïve Bayes, Kohavi [20] showed the favorability of 10-fold cross-validation. Green and Ohlsson [12] found out

cross-validation and hold-out (cut-off 0.25, 0.50) as being able to better estimate the true performance of artificial neural network ensembles. Our study did not find 10-fold cross-validation and LOOCV to be giving significantly better results in comparison with other resampling methods, but showed some promise in the use of Bootstrap method. Perhaps more empirical studies involving different variants of Bootstrap and different number of folds in k-fold cross-validation are required to reach a confident statement about the choice of resampling method, given different classification techniques and data sets. Replication studies [29] hold promise in verifying the generalizability of existing research.

What we can summarize from this study is that there are some implications on predictive studies in software engineering:

(1) We need to evaluate the use of bootstrapping more for software engineering data sets. This is promising particularly for software fault prediction and software quality classification studies where multiple copies of the same records do not pose a threat for a real-world use.
(2) If Bootstrapping is not in contention, LOOCV is preferred for smaller data sets and 10-fold cross-validation for larger data sets.
(3) For imbalanced data sets stratification might improve the performance outcome i.e. the folds are stratified so that they contain approximately the same proportions of the labels as the original data set.
(4) Using automated tool support it might be possible to report results using more than one resampling methods.
(5) It is important to take into account the data set properties before making a decision about a resampling method to select. Feature selection is one of the important decision criteria.

6. Validity Evaluation

There can be different threats to the validity of study results [40].

Conclusion validity is concerned with a statistical relationship between the treatment and the outcome with a given significance. We used Kruskal-Wallis test at 0.05 significance level with a post-hoc test where we used Wilcoxon rank sum test with 0.05 significance level with Bonferroni correction. While it is assumed that the power of a non-parametric test is less than its parametric counterpart, we were not sure of the data satisfying the assumptions of the parametric tests; therefore we resorted to the non-parametric statistical tests. The data sets used are from different domains and of different sizes; we believe that they represent a suitable heterogeneous mix.

Internal validity is concerned with a causal relationship between the treatment and the outcome. GP is a non-deterministic algorithm and different runs of the algorithm may give different results. Therefore, we followed a standard practice, i.e. to run the GP algorithm multiple times for different folds of the different resampling methods; the exception being the LOOCV where running GP multiple times was deemed not feasible. The selection of resampling methods to compare was motivated

by two factors: firstly commonly used methods in software engineering predictive studies and secondly other methods which have given good results in other domains but not necessarily tried in software engineering to a large extent. The data sets used are publicly available so that the validity of the study claims can be verified through replication.

Construct validity is concerned with the relation between theory and observation. While there are different ways to compare different resampling methods (e.g. bias, variance, mean square error to name a few), we chose the ones described in Sec. 3.4 after taking into account the possible drawbacks with prior approaches. Moreover, although bootstrapping might have an edge with respect to the location of (PF, PD) pairs in the ROC space, one should be mindful that bootstrapping is known not to perform well for some methodologies like empirical decision trees [20] and artificial neural network ensembles [12] for being excessively optimistic.

External validity is concerned with generalization of the results. While other learning algorithms could have been selected our primary motive was not to compare different algorithms for classification accuracy but to compare the different resampling methods.

7. Conclusion

In this study we have reported an extensive empirical comparison of five resampling methods using GP and MLR as classifiers over eight public domain software data sets from the PROMISE data repository. We used AUC and the location of (PF, PD) pairs in the ROC space as accuracy indicators and used statistical testing procedures for contrasting different resampling methods.

Using (PF, PD) pair data across eight data sets, bootstrapping gave results in the preferred region of the ROC space for three data sets with GP and for four data sets with MLR, indicating that bootstrapping should be considered as a resampling method of choice in predictive studies in software engineering. However, where the statistical comparison of individual resampling methods is concerned, based on AUC, there were no significant differences. We then highlight the possible reasoning of such an outcome, attributed to imbalanced data sets, insignificant predictor variables and high-dimensional data sets. Hold-out validation performs less satisfactorily for comparatively smaller data sets where LOOCV performs better due to optimal use of the training data. For comparatively larger data sets 10-fold cross-validation is a better choice as compared to LOOCV.

Some interesting areas of future work can be undertaken as an extension of this study:

(1) To investigate the outcome of different variants of bootstrapping for different software engineering data sets.
(2) Resampling methods are known to have complications when applied to time-series data, something that remains relatively less explored [35, 22].

(3) Another interesting area is to study the impact of different settings of GP parameters [9, 6] versus the resampling methods so that one can assess how much variability in the outcome can be attributed to each factor.
(4) According to [13], it is important to consider the variance of different resampling methods.
(5) It is important to investigate the computational costs of different resampling methods.
(6) Stratified sampling needs to be compared with other traditional resampling methods in this paper.

Acknowledgments

This work is part of the BESQ+ research project funded by the Knowledge Foundation (grant: 20100311) in Sweden.

References

1. W. Afzal and R. Torkar, A comparative evaluation of using genetic programming for predicting fault count data, in *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA'08)*, Piscataway, New Jersey, 2008.
2. W. Afzal and R. Torkar, Lessons from applying experimentation in software engineering prediction systems, in *Proceedings of the 2nd Int. WS on Software Productivity Analysis and Cost Estimation (SPACE'08)*, co-located with APSEC'08, 2008.
3. W. Afzal and R. Torkar, On the application of genetic programming for software engineering predictive modeling: A systematic review, *Expert Systems with Applications* **38**(9) (2011) 11984−11997.
4. G. Boetticher, T. Menzies and T. Ostrand, PROMISE repository of empirical software engineering data, 2007, http://promisedata.org/.
5. L. Brieman and P. Spector, Submodel selection and evaluation in regression: The x-random case, *International Statistical Review* **60** (1992) 291−319.
6. F. Castillo, A. Kordon, G. Smits, B. Christenson and D. Dickerson, Pareto front genetic programming parameter selection based on design of experiments and industrial data, in *Proceedings of the GECCO'06*, New York, USA, ACM, 2006.
7. B. Efron and G. Gong, A leisurely look at the bootstrap, the jackknife, and cross-validation, *The American Statistician* **37**(1) (1983) 36−48.
8. B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap* (CRC Press, Boca Raton, 1998).
9. R. Feldt and P. Nordin, Using factorial experiments to evaluate the effect of genetic programming parameters, in *Proceedings of the EuroGP'00* (Springer-Verlag, 2000).
10. G. Gong, Cross-validation, the jackknife, and the bootstrap: Excess error estimation in forward logistic regression, *Journal of the American Statistical Association* **81**(393) (1986) 108−113.
11. D. Gray, D. Bowes, N. Davey, Y. Sun and B. Christianson, Software defect prediction using static code metrics underestimates defect-proneness, in *2010 International Joint Conference on Neural Networks (IJCNN'10)*, 2010.
12. M. Green and M. Ohlsson, Comparison of standard resampling methods for performance estimation of ANN ensembles, in *Proc. 3rd Int. Conf. on Computational Intelligence in Medicine and Healthcare*, 2007.

13. T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer Series in Statistics, 2009).
14. Y. Jiang, B. Cukic and T. Menzies, Fault prediction using early lifecycle data, in *Proc. of ISSRE'07* (IEEE Computer Society, 2007).
15. N. Karunanithi, D. Whitley and Y. K. Malaiya, Using neural networks in reliability prediction, *IEEE Software* **9**(4) (1992) 53–59.
16. T. M. Khoshgoftaar, E. B. Allen, W. D. Jones and J. P. Hudepohl, Classification tree models of software quality over multiple releases, in *Proc of ISSRE'99* (IEEE Computer Society, 1999).
17. C. Kirsopp and M. Shepperd, Making inferences with small numbers of training sets, *IEEE Proceedings Software* **149**(5) (2002) 123–130.
18. B. Kitchenham and E. Mendes, Why comparative effort prediction studies may be invalid, in *Proc. of PROMISE'09*, ACM, 2009.
19. B. A. Kitchenham, E. Mendes and G. H. Travassos, Cross versus within-company cost estimation studies: A systematic review, *IEEE TSE* **33**(5) (2007) 316–329.
20. R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in *Proc. of the 14th Int. Joint Conf. on AI (IJCAI'95)* (Morgan Kaufmann, 1995).
21. W. B. Langdon and B. F. Buxton, Genetic programming for mining DNA chip data from cancer patients, *Genetic Programming and Evolvable Machines* **5**(3) (2004) 251–257.
22. B. LeBaron and A. S. Weigend, A bootstrap evaluation of the effect of data splitting on financial time series, *IEEE Trans. Neural Networks* **9**(1) (1998) 213–220.
23. S. Lessmann, B. Baesens, C. Mues and S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans Software Engineering* **34**(4) (2008) 485–496.
24. Y. Liu and T. M. Khoshgoftaar, Genetic programming model for software quality classification, in *Proc of HASE'01*, IEEE, 2001.
25. Y. Ma and B. Cukic, Adequate and precise evaluation of quality models in software engineering studies, in *Proc of PROMISE'07*, IEEE Computer Society, 2007.
26. N. Mittas and L. Angelis, Comparing cost prediction models by resampling techniques, *Journal of Systems and Software* **81**(5) (2008) 616–632.
27. A. M. Molinaro, R. Simon and R. M. Pfeiffer, Prediction error estimation: A comparison of resampling methods, *Bioinformatics* **21**(15) (2005) 3301–3307.
28. I. Myrtveit, E. Stensrud and M. Shepperd, Reliability and validity in comparative studies of software prediction models, *IEEE TSE* **31**(5) (2005) 380–391.
29. M. Ohlsson and P. Runeson, Experience from replicating empirical studies on prediction models, in *Proc. of 8th IEEE Symposium on Software Metrics*, 2002.
30. L. Pelayo and S. Dick, Applying novel resampling strategies to software defect prediction, in *Proc. of NAFIPS'07*, 2007.
31. R. Poli, W. B. Langdon and N. F. McPhee, A field guide to genetic programming, published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.
32. R. B. Rao and G. Fung, On the dangers of cross-validation. An experimental evaluation, in *Proc. of SIAM'08*, 2008.
33. B. J. Ross, The effects of randomly sampled training data on program evolution, in *Proc. of GECCO'00*, Morgan Kaufmann (2000).
34. C. Schaffer, Selecting a classification method by cross-validation, *Machine Learning* **13**(1) (1993) 135–143.
35. T. A. B. Snijders, On cross-validation for predictor evaluation in time series, *Dijkstra* (1988) pp. 56–69.

36. M. Stone, Cross-validatory choice and assessment of statistical predictions, *Journal of the Royal Statistical Society*, Series B-**36** (1974) 111−147.
37. L. Tian and A. Noore, Computational intelligence in reliability engineering, in *Computational Intelligence Methods in Software Reliability Prediction* (Springer, Berlin/Heidelberg, 2007), pp. 375−398.
38. M. C. Wang, Re-sampling procedures for reducing bias of error rate estimation in multinomial classification, *Computational Statistics and Data Analysis*, **4**(1) (1986) 15−39.
39. I. Witten and E. Frank, *Datamining — Practical Machine Learning Tools and Techniques* (Morgan Kaufmann, 2005).
40. C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering: An Introduction* (Kluwer, 2000).
41. C. H. Yu, Resampling methods: Concepts, applications, and justification, *Practical Assessment, Research & Evaluation* **8**(19) (2003).