Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings

Stefan Lessmann, *Student Member*, *IEEE*, Bart Baesens, Christophe Mues, and Swantje Pietsch

Abstract—Software defect prediction strives to improve software quality and testing efficiency by constructing predictive classification models from code attributes to enable a timely identification of fault-prone modules. Several classification models have been evaluated for this task. However, due to inconsistent findings regarding the superiority of one classifier over another and the usefulness of metric-based classification in general, more research is needed to improve convergence across studies and further advance confidence in experimental results. We consider three potential sources for bias: comparing classifiers over one or a small number of proprietary data sets, relying on accuracy indicators that are conceptually inappropriate for software defect prediction and cross-study comparisons, and, finally, limited use of statistical testing procedures to secure empirical findings. To remedy these problems, a framework for comparative software defect prediction experiments is proposed and applied in a large-scale empirical comparison of 22 classifiers over 10 public domain data sets from the NASA Metrics Data repository. Overall, an appealing degree of predictive accuracy is observed, which supports the view that metric-based classification is useful. However, our results indicate that the importance of the particular classification algorithm may be less than previously assumed since no significant performance differences could be detected among the top 17 classifiers.

Index Terms—Complexity measures, data mining, formal methods, statistical methods, software defect prediction.

1 Introduction

The development of large and complex software systems is a formidable challenge and activities to support software development and project management processes are an important area of research. This paper considers the task of identifying error prone software modules by means of metric-based classification, referred to as *software defect prediction*. It has been observed that the majority of a software system's faults are contained in a small number of modules [1], [20]. Consequently, a timely identification of these modules facilitates an efficient allocation of testing resources and may enable architectural improvements by suggesting a more rigorous design for high-risk segments of the system (e.g., [4], [8], [19], [33], [34], [44], [51], [52]).

Classification is a popular approach for software defect prediction and involves categorizing modules, represented by a set of software metrics or code attributes, into fault-prone (fp) and non-fault-prone (nfp) by means of a classification model derived from data of previous development projects [57]. Various types of classifiers have been applied to this task, including statistical procedures [4], [28], [47], tree-based methods [24], [30], [43], [53], [58], neural networks [29], [31], and analogy-based approaches [15], [23], [32]. However, as noted in [48], [49], [59], results regarding the superiority of one method over another or the usefulness of metric-based classification in general are not always consistent across different studies. Therefore, "*we need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models*" [49].

We argue that the size of the study, the way predictive performance is measured, as well as the type of statistical test applied to secure conclusions have a major impact on cross-study comparability and may have produced inconsistent findings. In particular, several (especially early) studies in software defect prediction had to rely upon a small number of, commonly proprietary, data sets, which naturally constrains the generalizability of observed results as well as replication by other researchers (see also [44]). Furthermore, different accuracy indicators are used across studies, possibly leading to contradictory results [49], especially if these are based on the number of misclassified fp and nfp modules. Finally, statistical hypothesis testing has only been applied to a very limited extent in the software defect prediction literature. As indicated in [44], [49], it is standard practice to derive conclusions without checking significance.

In order to remedy these problems, we propose a framework for organizing comparative classification experiments in software defect prediction and conduct a large-scale benchmark of 22 different classification models over 10 public-domain data sets from the NASA Metrics

- *S. Lessmann and S. Pietsch are with the Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, D-20146 Hamburg, Germany. E-mail: lessmann@econ.uni-hamburg.de, mailing@swantje-pietsch.de.*
- *B. Baesens is with the Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium. E-mail: Bart.Baesens@econ.kuleuven.ac.be.*
- *C. Mues is with the School of Management, University of Southampton, Southampton, SO17 1BJ, UK. E-mail: c.mues@soton.ac.uk.*

Manuscript received 11 May 2007; revised 24 Dec. 2007; accepted 23 Apr. 2008; published online 15 May 2008.
Recommended for acceptance by B. Littlewood.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2007-05-0159.
Digital Object Identifier no. 10.1109/TSE.2008.35.

Data (MDP) repository [10] and the PROMISE repository [56]. Comparisons are based on the *area under the receiver operating characteristics curve (AUC)*. As argued later in this paper, the AUC represents the most informative and objective indicator of predictive accuracy within a benchmarking context. Furthermore, we apply state-of-the-art hypothesis testing methods [12] to validate the statistical significance of performance differences among different classification models. Finally, the benchmarking study assesses the competitive performance of several established and novel classification models so as to appraise the overall degree of accuracy that can be achieved with (automated) software defect prediction today, investigate whether certain types of classifiers excel, and thereby support the (pre)selection of candidate models in practical applications. In this respect, our study can also be seen as a follow-up to Menzies et al.'s recent paper [44] on defect predictions, providing additional results as well as suggestions for a methodological framework.

This paper is organized as follows: Section 2 first reviews accuracy indicators for classification and discusses the distinctive merits of receiver operating characteristic (ROC) analysis, after which statistical testing procedures for model comparisons are presented. Section 3 is devoted to the benchmarking experiment and discusses the respective setup, findings, as well as limitations. Conclusions are given in Section 4.

2 Components of the Benchmarking Framework

In this section, we present the two major components of our framework. First, we discuss the difficulties associated with assessing a classification model in software defect prediction and advocate the use of the AUC to improve cross-study comparability. Subsequently, the statistical testing procedures applied within the benchmarking experiment are introduced.

2.1 Accuracy Indicators for Assessing Binary Classification Models

The task of (binary) classification can be defined as follows: Let $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ be a training data set of N examples, where $\boldsymbol{x}_i \in \Re^M$ represents a software module that is characterized by M software metrics and $y_i \in \{\mathrm{nfp}, \mathrm{fp}\}$ denotes its binary class label. A classification model is a mapping from instances $\boldsymbol{x}$ to predicted classes y: $f(\boldsymbol{x}) : \Re^M \mapsto \{\mathrm{nfp}, \mathrm{fp}\}$.

Binary classifiers are routinely assessed by counting the number of correctly predicted modules over hold-out data. This procedure has four possible outcomes: If a module is fp and is classified accordingly, it is counted as true positive (TP); if it is wrongly classified as nfp, it is counted as false negative (FN). Conversely, an nfp module is counted as true negative (TN) if it is classified correctly or as false positive (FP) otherwise. El-Eman et al. describe a large number of performance indicators which can be constructed from these four basic figures [15].

A defect prediction model should identify as many fp modules as possible while avoiding false alarms. Therefore, classifiers are predominantly evaluated by means of their TP rate (TPR), also known as sensitivity, rate of detection, or hit rate, and by their FP rate (FPR) or false alarm rate (e.g., [24], [32], [44], [67]):

$$\mathrm{TPR} = \mathrm{TP}/(\mathrm{FN} + \mathrm{TP}); \mathrm{FPR} = \mathrm{FP}/(\mathrm{TN} + \mathrm{FP}). \tag{1}$$

We argue that such error-based metrics, although having undoubted practical value, are conceptually inappropriate for empirical comparisons of the competitive performance of classification algorithms. This is because they are constructed from a discrete classification of modules into fp and nfp. Most classifiers do not produce such crisp classifications but instead produce probability estimates or confidence scores, which represent the likelihood that a module belongs to a particular class. Consequently, threshold values have to be defined for converting such continuous predictions into discrete classifications [17]. The Bayes rule of classification guides the choice of threshold value: Let $p(\mathrm{fp})$ and $p(\mathrm{nfp})$ denote the prior probabilities of fp and nfp modules, respectively. The objective of software defect classification is to estimate the a posteriori probability of a module with characteristics $\boldsymbol{x}$ to be fp, which we denote by $p(y = \mathrm{fp}|\boldsymbol{x})$, with analogous meaning for $p(y = \mathrm{nfp}|\boldsymbol{x})$. Let C_{FP} denote the cost of conducting an FP error, i.e., classifying an nfp module incorrectly as fp, and C_{FN} the cost of an FN error (misclassifying an fp module). Then, Bayes rule (e.g., [27]) states that modules should be classified as fp if

$$\frac{p(\boldsymbol{x}|y = \mathrm{fp})}{p(\boldsymbol{x}|y = \mathrm{nfp})} > \frac{p(\mathrm{nfp}) \cdot C_{FP}}{p(\mathrm{fp}) \cdot C_{FN}}, \tag{2}$$

whereby $p(\boldsymbol{x}|y = \mathrm{fp})$ and $p(\boldsymbol{x}|y = \mathrm{nfp})$ represent the so-called class conditional probabilities, which are related to the a posteriori probabilities via Bayes theorem.

The Bayes optimal threshold, i.e., the right-hand side of (2), depends on prior probabilities and misclassification costs or their respective ratios. However, within a benchmarking context, classifiers should be compared over several data sets from several different software releases and/or projects (see also [9], [44], [52]) and it is extremely unlikely that information on class and cost distributions is available for every data set. Consequently, the necessary information to determine meaningful and objective threshold values is usually missing. This problem can be alleviated by relying on default values or estimating settings from the data [33]. However, two studies that use the same classifiers and data sets could easily come to different conclusions just because different procedures for determining classification thresholds are employed. Furthermore, it should be noted that detailing the concrete strategy for determining thresholds is not a standard practice in the defect prediction literature. Consequently, comparing algorithms by means of discrete classifications leaves considerable room for bias and may cause inconsistencies across studies. Our key point is that this risk can be easily avoided if defect predictors are assessed independently from thresholds, i.e., over all possible combinations of misclassification costs and prior probabilities of fp and nfp modules. ROC analysis is a tool that realizes such an evaluation.

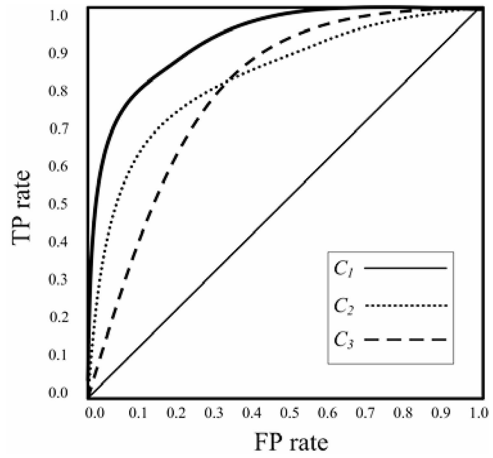


Fig. 1. Exemplary ROC curve of three classifiers with dominating classifier C_1.

The ROC graph is a 2D illustration of TPR on the Y-axis versus FPR on the X-axis (Fig. 1). An ROC curve is obtained by varying the classification threshold over all possible values [17]. Thereby, each ROC curve passes through the points (0, 0), representing a classifier that always predicts nfp, and (1, 1), the opposite case [44]. The ideal point is the upper left corner (0, 1) since such a classifier accurately identifies all fp modules $(\mathrm{TPR} = 1)$ while making no error $(\mathrm{FPR} = 0)$. Hence, points toward the northwest are preferable, i.e., achieve a high hit rate with low FPR. The advantages of the ROC analysis are its robustness toward imbalanced class distributions and to varying and asymmetric misclassification costs [54]. Therefore, it is particularly well suited for software defect prediction tasks which naturally exhibit these characteristics [33], [44].

To compare different classifiers, their respective ROC curves are drawn in ROC space. Fig. 1 provides an example of three classifiers, C_1, C_2, and C_3. C_1 is a dominating classifier because its ROC curve is always above that of its competitors, i.e., it achieves a higher TP rate for all FP rates.

As ROC curves of different classifiers may intersect (e.g., curves C_2 and C_3), one often calculates the AUC as a single scalar measure of expected performance [6]. Higher AUC values indicate that the classifier is on average more to the upper left region of the graph.

The AUC has the potential to significantly improve convergence across empirical experiments in software defect prediction because it separates predictive performance from operating conditions, i.e., class and cost distributions, and thus represents a general measure of predictiveness. The importance of such a general indicator in comparative experiments is reinforced when considering the discussion following Menzies et al.'s paper [44] about whether the accuracy of their models is or is not sufficient for practical applications and whether method A is or is not better than method B [42], [66]. Furthermore, the AUC has a clear statistical interpretation: It measures the probability that a classifier ranks a randomly chosen fp module higher than a randomly chosen nfp module, which is equivalent to the Wilcoxon test of ranks [17]. Consequently, any classifier achieving AUC well above 0.5 is demonstrably effective for identifying fp modules and gives valuable advice as to which modules should receive particular attention in software testing.

2.2 Statistical Comparison of Classification Models

Few reported studies in software defect prediction make use of statistical inference. For example, analysis of variance (ANOVA) is applied in [33], [34], [58] to determine if observed performance differences between candidate methods are statistically significant. However, as indicated in [44], [49], the prevailing approach is to derive conclusions solely from empirical results without applying formal hypothesis tests. As will be shown later, this practice may be misleading and consequently represents another possible source for inconsistency across experiments.

In a recent article, Demšar reviewed the problem of benchmarking classifiers and offered valuable guidance on how to organize such comparisons in a statistically sound manner [12]. Subsequently, we summarize his recommendations for the comparison of multiple algorithms over multiple data sets, which we deem most relevant for software defect prediction.[1]

The null hypothesis, H_0, being tested in this setting is that all algorithms perform alike. That is, it is assumed that performance differences observed within an empirical experiment are just due to random chance. Performance may be measured by means of an arbitrary accuracy indicator, e.g., the AUC. Testing the significance of differences between multiple means, i.e., mean accuracies across different data sets, is a well-known statistical problem and ANOVA is specifically designed for this purpose. However, Demšar explicitly discourages the use of ANOVA for comparing classifiers because it is based on assumptions that are most likely violated within this setting [12]. In particular, ANOVA assumes that: 1) Performance differences are distributed normally, which can be taken for granted only if the sample size is large, i.e., the algorithms are compared over many data sets (~ 30), 2) all classifiers exhibit the same variance in predictive performance over all data sets (homogeneity of variance), and 3) the variance in performance differences across two classifiers is identical for all possible pairs of classifiers (sphericity assumption) [65]. On the one hand, the validity of these assumptions is difficult to check when the number of samples (i.e., data sets) is limited. On the other hand, violations, especially with respect to nonsphericity, have been shown to be highly detrimental to ANOVA and especially to the subsequently performed post hoc tests [55]. Consequently, Demšar recommends the Friedman test for classifier comparisons, which is a nonparametric alternative to ANOVA and relies on less restrictive assumptions [12].

Friedman's test is based on ranked performances rather than actual performance estimates and is therefore less susceptible to outliers. All classifiers are ranked according to their performance in ascending order for each data set and the mean rank of a classifier i, AR_i, is computed across all data sets. With K representing the overall number of data sets, L the number of classifiers, and r_j^i the rank of

[1] Note that dedicated tests are applicable for comparing only two classifiers over a single or multiple data sets [12].

classifier i on data set j, the test statistic of the Friedman test is calculated as

$$\chi_F^2 = \frac{12K}{L(L+1)}\left[\sum\nolimits_{i=1}^{L} AR_i^2 - \frac{L(L+1)^2}{4}\right], \\ AR_i = \frac{1}{K}\sum\nolimits_{j=1}^{K} r_j^i, \tag{3}$$

and is distributed according to the Chi-Square distribution with $L-1$ degrees of freedom [65].

If the value of the test statistic is large enough to reject the null hypothesis, it may be concluded that performance differences among classifiers are nonrandom. In this case, a so-called post hoc test can be applied to detect which specific classifiers differ significantly. Demšar recommends the test of Nemenyi for this task [12]. For all pairs of classifiers, it tests the null hypothesis that their respective mean ranks are equal, which may be rejected if the difference between their mean ranks exceeds the critical difference CD:

$$CD = q_{a,\infty,L}\sqrt{\frac{L(L+1)}{12K}}. \tag{4}$$

The value $q_{a,\infty,L}$ is based on the Studentized range statistic and is tabulated in standard statistical textbooks.[2]

3 Empirical Evaluation of Candidate Classifiers on NASA MDP Data

In this section, we describe the setup of the benchmarking study and elaborate on the experimental design. Subsequently, the empirical results are presented in detail, together with a discussion of possible limitations and threats to validity.

3.1 Data Set Characteristics

The data used in this study stems from the NASA MDP repository [10]. Ten software defect prediction data sets are analyzed, including the eight sets used in [44] as well as two additional data sets (JM1 and KC1, see also Table 1). Each data set is comprised of several software modules, together with their number of faults and characteristic code attributes. After preprocessing, modules that contain one or more errors were labeled as fp, whereas error-free modules were categorized as nfp. Besides LOC counts, the NASA MDP data sets include several Halstead attributes as well as McCabe complexity measures. The former estimates reading complexity by counting operators and operands in a module, whereas the latter is derived from a module's flow graph. The reader is referred to [26], [41], [44] for a more detailed description of code attributes or the origin of the MDP data sets. Individual attributes per data set, together with some general statistics, are given in Table 1.

3.2 Experimental Design

The benchmarking experiment aims at contrasting the competitive performance of several classification algorithms. To that end, an overall number of 22 classifiers is selected, which may be grouped into the categories of statistical approaches, nearest-neighbor methods, neural networks, support vector machines, tree-based methods, and ensembles. The selection aims at achieving a balance between established techniques, such as Naive Bayes, decision trees, or logistic regression, and novel approaches that have not yet found widespread usage in defect prediction (e.g., different variants of support vector machines, logistic model trees, or random forests). The classifiers are sketched in Table 2, together with a brief description of their underlying paradigms. A detailed description of most methods can be found in general textbooks like [14], [27]; specific references are given for less known/novel techniques.

The merit of a particular classifier (in terms of the AUC) is estimated on a randomly selected hold-out test set (so-called split-sample setup). More specifically, all data sets are randomly partitioned into training and test set using 2/3 of the data for model building and 1/3 for performance estimation. Besides providing an unbiased estimate of a classifier's generalization performance, the split-sample setup offers the advantage of enabling easy replication, which constitutes an important part of empirical research [2], [19], [49], [50]. Furthermore, its choice is motivated by the fact that the split-sample setup is the prevailing approach to assess predictive accuracy in software defect prediction [15], [16], [23], [28], [32], [33], [34], [37].

Several classification models exhibit adjustable parameters, also termed hyperparameters, which enable an adaptation of the algorithm to a specific problem. It is known that a careful tuning of such hyperparameters is essential to obtain a representative assessment of the classifier's potential (see, e.g., [3], [63]). For example, neural network models require specification of network architecture (number of hidden layers, number of nodes per layer), whereas a pruning strategy has to be defined for tree-based classifiers. We adopt a grid-search approach to organize this model selection step. That is, a set of candidate values is defined for each hyperparameter and all possible combinations are evaluated empirically by means of 10-fold cross validation on the training data. The parameter combination with maximal cross-validation performance is retained and a respective classification model is constructed on the whole training data set. Since we advocate using the AUC for classifier comparison, the same metric is used during model selection to guide the search toward predictive parameter settings. The respective candidate values are described in the Appendix to enable a replication of our experiments.

3.3 Experimental Results

Next, we present the results of the empirical comparison in terms of the AUC. The last column of Table 3 reports the mean rank AR_i (3) of each classifier over all MDP data sets, which constitutes the basis of the Friedman test. The classifier yielding the best AUC for a particular data set is

[2] Note that more powerful post-hoc tests are available if one is interested in the performance of one particular classifier, e.g., to test if a novel technique performs significantly better than an established benchmark (see [11] for details).

TABLE 1
Code Attributes within the MDP Data Sets

		NASA MDP dataset									
		CM1	KC1	KC3	KC4	MW1	JM1	PC1	PC2	PC3	PC4
LOC counts	LOC_total	X	X	X	X	X	X	X	X	X	X
	LOC_blank	X	X	X		X	X	X	X	X	X
	LOC_code_and_comment	X	X	X		X	X	X	X	X	X
	LOC_comments	X	X	X		X	X	X	X	X	X
	LOC_executable	X	X	X		X	X	X	X	X	X
	Number_of_lines	X		X		X		X	X	X	X
Halstead attributes	content	X	X	X		X	X	X	X	X	X
	difficulty	X	X	X		X	X	X	X	X	X
	effort	X	X	X		X	X	X	X	X	X
	error_est	X	X	X		X	X	X	X	X	X
	length	X	X	X		X	X	X	X	X	X
	level	X	X	X		X	X	X	X	X	X
	prog_time	X	X	X		X	X	X	X	X	X
	volume	X	X	X		X	X	X	X	X	X
	num_operands	X	X	X		X	X	X	X	X	X
	num_operators	X	X	X		X	X	X	X	X	X
	num_unique_operands	X	X	X		X	X	X	X	X	X
	num_unique_operators	X	X	X		X	X	X	X	X	X
McCabe attributes	cyclomatic_complexity	X	X	X	X	X	X	X	X	X	X
	cyclomatic_density	X		X	X	X		X	X	X	X
	design_complexity	X	X	X	X	X	X	X	X	X	X
	essential_complexity	X	X	X	X	X	X	X	X	X	X
Miscellaneous	branch_count	X	X	X	X	X	X	X	X	X	X
	call_pairs	X		X	X	X		X	X	X	X
	condition_count	X		X		X		X	X	X	X
	decision_count	X		X		X		X	X	X	X
	decision_density	X		X		X		X	X	X	X
	design_density	X		X	X	X		X	X	X	X
	edge_count	X		X	X	X		X	X	X	X
	essential_density	X		X	X	X		X	X	X	X
	parameter_count	X		X		X		X	X	X	X
	maintenance_severity	X		X	X	X		X	X	X	X
	modified_condition_count	X		X		X		X	X	X	X
	multiple_condition_count	X		X		X		X	X	X	X
	global_data_complexity			X							
	global_data_density			X							
	normalized_cyclomatic_compl.	X		X	X	X		X	X	X	X
	percent_comments	X		X		X		X	X	X	X
	node_count	X		X	X	X		X	X	X	X
Number of code attributes		37	21	39	13	37	21	37	37	37	37
Number of modules		505	1571	458	125	403	9537	1059	4505	1511	1347
Number of fp modules		48	319	43	61	31	1777	76	23	160	178
Percentage of fp modules		9.50	20.31	9.39	48.80	7.69	18.63	7.18	0.51	10.59	13.21

highlighted in boldface. Note that all figures are based on hold-out test data; results on training data are omitted for brevity.

Most classifiers achieve promising AUC results of 0.7 and more, i.e., rank deficient modules higher than accurate ones with probability > 70 percent. Overall, this level of accuracy confirms Menzies et al.'s conclusion that "*defect predictors are demonstrably useful*" for identifying fp modules and guiding the assignment of testing resources [44]. Furthermore, one observes a concentration of novel and/or sophisticated classifiers like RndFor, LS-SVMs, MLPs, and Bayesian networks among the best performing algorithms. While, e.g., analogy-based classification is a popular tool for software defect prediction and has been credited for its accuracy in several studies (e.g., [15], [23], [32], [34], [38], [60]), Table 3 seems to suggest that analogy-based approaches (kNN and K*) are outperformed when compared against these state-of-the-art competitors.

However, to evaluate individual classification models and verify if some are generally superior to others, it is important to test whether the differences in AUC are significant. This is confirmed when conducting the Friedman test: Its p-value

TABLE 2
Classification Models Employed in the Comparative Experiment

Classification model		Philosophy
Statistical classifiers		Strive to construct a Bayes optimal classifier by estimating either posterior probabilities directly (LogReg), or class-conditional probabilities (LDA, QDA, NB, BayesNet) which are subsequently converted into posterior probabilities using Bayes' theorem. LDA/QDA assume a multivariate Gaussian density function, whereas NB is based on the assumption that attributes are conditionally independent, so that class-conditional probabilities can be estimated individually per attribute. BayesNet extends NB by explicitly modeling statements about independence and correlation among attributes. LARS adopts a different approach and consists of a multivariate linear regression model and heuristics to shrink the number of features. RVM has been proposed as an extension of the SVM (see below) which avoids the need to tune certain hyperparameters and may incorporate kernel functions SVMs are unable to process.
Linear Discriminant Analysis[2, 3]	(LDA)	
Quadratic Discriminant Analysis[2, 3]	(QDA)	
Logistic Regression[2, 3]	(LogReg)	
Naïve Bayes[1]	(NB)	
Bayesian Networks[1]	(BayesNet)	
Least-Angle Regression[2]	(LARS)	
Relevance Vector Machine[2] [62]	(RVM)	
Nearest neighbor methods		Belong to the group of analogy-based methods which classify a module by considering the *k* most similar examples. The definition of similarity differs among algorithms. An Euclidian distance is used in *k*-NN whereas K* employs an entropy-based distance function.
k-Nearest Neighbor[1]	(k-NN)	
K-Star[1] [11]	(K*)	
Neural Networks		Mathematical representations inspired by the functioning of the human brain. They depict a network structure which defines a concatenation of weighting, aggregation and thresholding functions that are applied to a software module's attributes to obtain an approximation of its posterior probability of being fp. The study includes two types of MLP classifiers which incorporate different approaches to avoid overfitting the training data, i.e. weight decay and Bayesian Learning.
Multi-Layer Perceptron[2, 4]	(MLP)	
Radial Basis Function Network[1]	(RBF net)	
Support vector machine-based classifiers		Utilize mathematical programming to optimize a linear decision function that discriminates between fp and nfp modules. A kernel function enables more complex decision boundaries by means of an implicit, nonlinear transformation of attribute values. This kernel function is polynomial for the VP classifier, whereas SVM and LS-SVM consider a radial basis function. L-SVM and LP are linear classifiers.
Support Vector Machine[2]	(SVM)	
Lagrangian SVM[2] [40]	(L-SVM)	
Least Squares SVM[2] [61]	(LS-SVM)	
Linear Programming[2]	(LP)	
Voted Perceptron[1] [22]	(VP)	
Decision tree approaches		Recursively partition the training data by means of attribute splits. The algorithms differ mainly in the splitting criterion which determines the attribute used in a given iteration to separate the data. C4.5 induces decision trees based on the information-theoretical concept of entropy, whereas CART uses the Gini criterion. ADT distinguishes between alternating splitter and prediction nodes. A prediction is computed as the sum over all prediction nodes an instance visits while traversing the tree.
C 4.5 Decision Tree[1]	(C 4.5)	
Classification and Regression Tree[2]	(CART)	
Alternating Decision Tree[1] [21]	(ADT)	
Ensemble methods		Meta-learning schemes that embody several base-classifiers. These are built independently and participate in a voting procedure to obtain a final class prediction. RndFor incorporates CART as base learner, whereas LMT utilizes LogReg. Each base learner is derived from a limited number of attributes. These are selected at random within the RndFor procedure, whereby the user has to predefine their number. LMT considers only univariate regression models, i.e. uses one attribute per iteration, which is selected automatically.
Random Forest[1] [7]	(RndFor)	
Logistic Model Tree[1] [36]	(LMT)	

[1] Classifier is implemented using the YALE workbench [45].

[2] Classifier is implemented using the MATLAB environment.

[3] These classifiers fail to produce a classification model if all attributes are used. Therefore, they are trained in conjunction with a backward-feature elimination heuristic [25] (see also Appendix I).

[4] Subsequently, we use the abbreviation MLP-1 to refer to a multi-layer perceptron neural network which has been trained with a weight decay penalty to prevent overfitting, whereas MLP-2 represents a network which uses a Bayesian learning paradigm (see also Appendix I).

of $2.1E - 009$ indicates that it is very unlikely that the observed performance differences among classifiers are just random. Consequently, one may proceed with a post hoc test to detect which particular classifiers differ significantly. This is accomplished by applying Nemenyi's post hoc test ($\alpha = 0.05$), i.e., conducting all pairwise comparisons between different classifiers and checking which models' performance differences exceed the critical difference (4).

TABLE 3
Hold-Out Test Set Results of 22 Classification Algorithms over 10 NASA MDP Data Sets in Terms of the AUC

	CM1	KC1	KC3	KC4	MW1	JM1	PC1	PC2	PC3	PC4	*AR*
Statistical classifiers											
LDA	0.77	**0.78**	0.62	0.73	0.82	0.73	0.82	0.87	0.82	0.88	9.7
QDA	0.70	**0.78**	0.74	0.80	**0.83**	0.70	0.70	0.80	0.78	0.86	13.1
LogReg	0.80	0.76	0.61	0.74	0.82	0.73	0.82	0.86	0.82	0.89	10.0
NB	0.72	0.76	0.83	0.68	0.80	0.69	0.79	0.85	0.81	0.85	12.9
Bayes Net	0.79	0.75	0.83	0.80	0.82	0.73	0.84	0.85	0.80	0.90	8.7
LARS	0.84	0.75	0.80	0.76	0.74	0.72	0.70	0.30	0.79	0.90	13.3
RVM	0.82	0.76	0.74	0.74	0.75	0.72	0.84	0.91	0.82	0.89	10.4
Nearest neighbor methods											
k-NN	0.70	0.70	0.82	0.79	0.75	0.71	0.82	0.77	0.77	0.87	14.5
K*	0.76	0.68	0.71	0.81	0.71	0.69	0.72	0.62	0.74	0.83	17.1
Neural networks											
MLP-1	0.76	0.77	0.79	0.80	0.77	0.73	0.89	**0.93**	0.78	0.95	6.9
MLP-2	0.82	0.77	0.83	0.76	0.76	0.73	**0.91**	0.84	0.81	0.94	6.9
RBF net	0.58	0.76	0.68	0.73	0.65	0.69	0.64	0.79	0.78	0.79	17.8
Support vector machine-based classifiers											
SVM	0.70	0.76	**0.86**	0.77	0.65	0.72	0.80	0.85	0.77	0.92	13.0
L-SVM	0.80	0.76	0.82	0.76	0.76	0.73	0.86	0.83	**0.84**	0.92	7.7
LS-SVM	0.75	0.77	0.83	0.81	0.60	0.74	0.90	0.85	0.83	0.94	6.8
LP	**0.90**	0.75	0.74	0.83	0.74	0.72	0.73	0.88	0.82	0.92	9.3
VP	0.72	0.76	0.74	0.73	0.73	0.54	0.75	0.50	0.74	0.83	18.2
Decision tree approaches											
C4.5	0.57	0.71	0.81	0.76	0.78	0.72	0.90	0.84	0.78	0.93	11.6
CART	0.74	0.67	0.62	0.79	0.67	0.61	0.70	0.68	0.63	0.79	19.3
ADT	0.78	0.69	0.74	0.81	0.76	0.73	0.85	0.70	0.76	0.94	11.8
Ensemble methods											
RndFor	0.81	**0.78**	0.86	**0.85**	0.81	**0.76**	0.90	0.82	0.82	**0.97**	**4.0**
LMT	0.81	0.76	0.78	0.80	0.71	0.72	0.86	0.83	0.80	0.92	10.4

The results of the pairwise comparisons are depicted in Fig. 2, utilizing a modified version of Demšar's significance diagrams [12]: The diagram plots classifiers against mean ranks, whereby all methods are sorted according to their ranks. The line segment to the right of each classifier represents its corresponding critical difference. That is, the right end of the line indicates from which mean rank onward another classifier is outperformed significantly. For illustrative purposes, this threshold is highlighted with a vertical dotted line in three cases. The leftmost vertical line is associated with RndFor. Therefore, all classifiers right to this line perform significantly worse than RndFor. The second line separates the MLP-1 classifier from RBF net, VP, and CART. Hence, these are significantly inferior to MLP-1 and any better-ranked method. Finally, the third line indicates that the Bayes net classifier is significantly better than CART.

The statistical comparison reveals an interesting finding: Despite noteworthy differences in terms of the AUC among competing classifiers, all methods—with few exceptions—do not differ significantly. This result may be explained as follows: The relationship between the code attributes and the dependent variable $y \in \{\text{fp}|\text{nfp}\}$ is clearly present but limited (e.g., AUC ~ 0.7). This relationship is disclosed by almost all classifiers and seems to be predominantly linear. This view is reinforced when considering that relatively simple classifiers like LP, LogReg, LDA, and especially L-SVM provide respectable results. These techniques separate fp and nfp modules by means of a linear decision function and are consequently restricted to merely accounting for linear dependencies among code attributes. In other words, their competitive performance indicates that the degree of nonlinearity within the MDP data sets is limited. Following this reasoning, one may conclude that the choice of classification modeling technique is less important than generally assumed and that practitioners are free to choose from a broad set of candidate models when building defect predictors.

However, it should be noted that Nemenyi's test checks the null hypothesis that two classifiers give equal performance. Failing to reject this H_0 does not guarantee that it is true. For example, Nemenyi's test is unable to reject the null hypothesis that RndFor and LARS have the same mean rank. This can mean that the performance differences

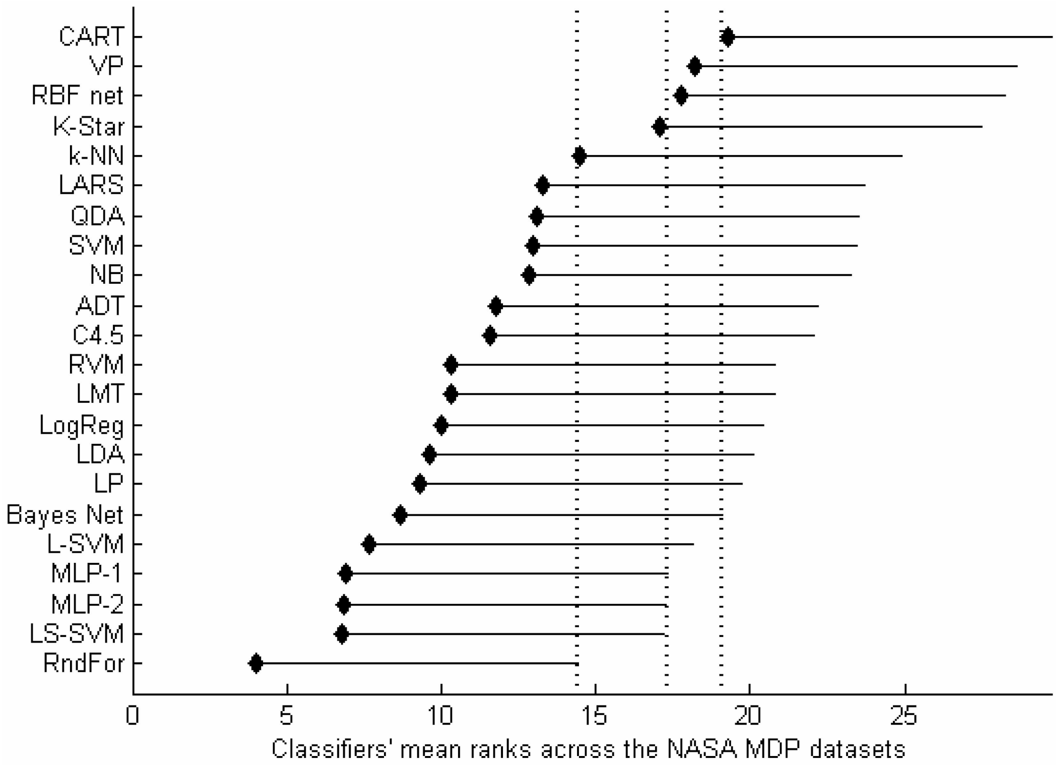


Fig. 2. Results of the pairwise comparisons of all classifiers using Nemenyi's post hoc test with $\alpha = 0.05$.

between these two are just due to chance. But, the result could also be caused by a Type II error: Possibly the Nemenyi test does not have enough power to detect a significant difference at $\alpha = 0.05$. In other words, only rejecting H_0 allows the conclusion that it is very likely (with probability $1 - \alpha$) that two classifiers differ significantly.

With the former in mind, a general conclusion that may be drawn from the benchmarking experiment is that predictive performance alone does not suffice to appraise the merit of a classification model and has to be augmented by other criteria. For example, Vandecruys et al. [64] argue in favor of comprehensible classifiers and propose an Ant-Colony optimization based detection system. Similarly, Menzies et al. point out that their preferred classifier, a Naive Bayes model, is easy to interpret as well as computationally efficient [44]. Clearly, computational efficiency and transparency are desirable features of candidate classifiers and it appears to be a promising area for future research to formalize these concepts, e.g., by developing a multidimensional classifier assessment system. Meanwhile, the results observed here confirm previous findings regarding the effectiveness of RndFor for software defect prediction [24] and allow recommending this classifier for future experiments or practical applications. It is fast to train and requires only moderate parameter tuning, i.e., it is robust toward parameter settings. Furthermore, RndFor naturally assesses the relevance of individual code attributes (see [7]) and thereby provides not just an accurate but also an understandable model.

3.4 Threats to Validity

When conducting an empirical study, it is important to be aware of potential threats to the validity of the obtained results and derived conclusions. A possible source of bias relates to the data used, e.g., its measurement accuracy and representativeness if results are to be generalized. Using public domain data secures the results in so far as that they can be verified by replication and compared with findings from previous experiments. Also, several authors have argued in favor of the appropriateness and representativeness of the NASA MDP repository and/or used some of its data sets for their experiments (e.g., [24], [35], [44], [64], [67]). Therefore, we are confident that the obtained results are relevant for the software defect prediction community.

Despite the general suitability of the data, the sampling procedure might bias results and prevent generalization. We consider a split-sample setup with randomly selected test records (1/3 of the available data set). This is a well-established approach for comparative classification experiments and the size of the MDP data sets seems large enough to justify this setting. Compared to cross validation or bootstrapping, the split sample setup saves a considerable amount of computation time, which, in turn, can be invested into model selection to ensure that the classifiers are well tuned to each data set. It would be interesting to quantify possible differences between a split-sample setup and cross-validation/bootstrapping setups by means of empirical experimentation. However, this step is left for future research.

The selection of classifiers is another possible source of bias. Given the variety of available learning algorithms, there are still others that could have been considered. Our selection is guided by the aim of finding a meaningful balance between established techniques and novel approaches. We believe that the most important representatives of different domains (statistics, machine learning, and so forth) are included.

Finally, it should be noted that classification is only a single step within a multistage data mining process [18]. Especially, data preprocessing or engineering activities such as the removal of noninformative features or the discretization of continuous attributes may improve the performance of some classifiers (see, e.g., [13], [25]). For example, Menzies et al. report that their Naive Bayes classifier benefits from feature selection and a log-filter preprocessor [44]. Such techniques have an undisputed value. However, a wide range of different algorithms for feature selection, discretization, scaling, and so forth has been proposed in the data mining literature. A thorough assessment of several candidates seems computationally infeasible when considering a large number of classifiers at the same time. That is, each added individual preprocessing algorithm would multiply the computational effort of the whole study. Our view is that simple classifiers like Naive Bayes or decision trees would especially benefit from additional preprocessing activities (see [13]), whereas sophisticated techniques are well prepared to cope with, e.g., large and correlated feature sets through inbuilt regularization facilities [7], [27], [61]. As our results indicate that most simple classifiers are already competitive with more sophisticated approaches, i.e., not significantly inferior, it seems unlikely that preprocessing activities would alter our overall conclusion that most methods do not differ significantly in terms of predictive accuracy.

4 Conclusions

In this paper, we have reported on a large-scale empirical comparison of 22 classification models over 10 public domain software development data sets from the NASA MDP repository. The AUC was recommended as the primary accuracy indicator for comparative studies in software defect prediction since it separates predictive performance from class and cost distributions, which are project-specific characteristics that may be unknown or subject to change. Therefore, the AUC-based evaluation has the potential to significantly improve convergence across studies. Another contribution along this line was the discussion and application of statistical testing procedures, which are particularly appropriate for contrasting classification models.

The overall level of predictive accuracy across all classifiers confirmed the general appropriateness of defect prediction to identify fp software modules and guide the assignment of testing resources [44]. In particular, previous findings regarding the efficacy of RndFor for defect prediction [24] were confirmed.

However, where the statistical comparison of individual models is concerned, the major conclusion is that the predictive accuracy of most methods does not differ significantly according to a Nemenyi post hoc test ($\alpha = 0.05$). This suggests that the importance of the classification model may have been overestimated in the previous research, hence illustrating the relevance of statistical hypothesis testing. Given that basic models, and especially linear ones such as LogReg, LP, and LDA, give similar results to more sophisticated classifiers, it is evident that most data sets are fairly well linearly separable. In other words, simple classifiers suffice to model the relationship between static code attributes and software defect.

Consequently, the assessment and selection of a classification model should not be based on predictive accuracy alone but should be comprised of several additional criteria like computational efficiency, ease of use, and especially comprehensibility. Comprehensible models reveal the nature of detected relationships and help improve our overall understanding of software failures and their sources, which, in turn, may enable the development of novel predictors of fault-proneness. In fact, efforts to design new software metrics and other explanatory variables appear to be a particularly promising area for future research and have the potential to achieve general accuracy improvements across all types of classifiers. We hope that the proposed framework will offer valuable guidance for appraising the potential of respective advancements.

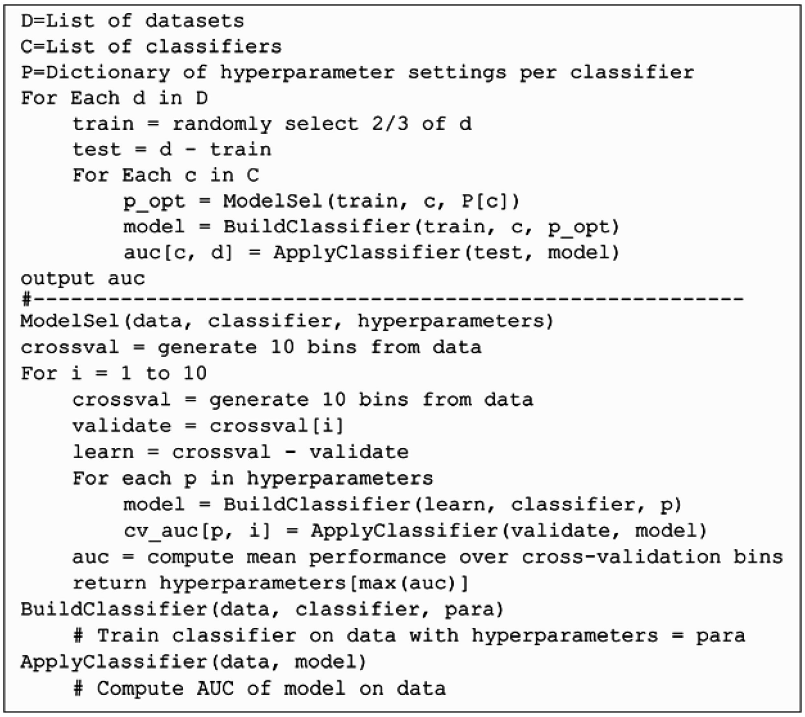
Appendix

Model Selection Methodology

This section reports hyperparameter settings that have been considered for individual classifiers during model selection. These settings may be useful for other researchers when trying to replicate the results observed within this study. It should be noted that, since a hold-out test set of 1/3 is randomly selected and removed from the overall data set, we employ 10-fold cross validation during model selection to assess individual candidate hyperparameter settings, to avoid bias because of a small training sample. The overall experimental setup has been motivated in Section 3.2 and is summarized in Fig. 3.

In general, most statistical classifiers do not require additional model selection and are estimated directly from the training data. This approach has been adopted for LARS, NB, and RVM. However, some methods (LDA, QDA, and LogReg) suffer from correlations among the attributes and require additional feature selection to produce a valid classification model. Consequently, model selection for these classifiers consists of identifying a suitable set of attributes by means of a backward feature-elimination heuristic [25].

The BayesNet classifier is a directed acyclic graph that represents the joint probability distribution of code attributes and target variable, i.e., each node in the graph represents an attribute and each arc represents a correlation or dependency. Thus, learning a BayesNet can be considered an optimization problem where a quality measure of the network structure has to be maximized. Therefore, different search techniques (K2, simulated annealing, tabu search, hill climbing, tree augmented Naive Bayes) implemented in the YALE machine learning workbench [45] have been evaluated.



```
D=List of datasets
C=List of classifiers
P=Dictionary of hyperparameter settings per classifier
For Each d in D
    train = randomly select 2/3 of d
    test = d - train
    For Each c in C
        p_opt = ModelSel(train, c, P[c])
        model = BuildClassifier(train, c, p_opt)
        auc[c, d] = ApplyClassifier(test, model)
output auc
#--------------------------------------------------------
ModelSel(data, classifier, hyperparameters)
crossval = generate 10 bins from data
For i = 1 to 10
    crossval = generate 10 bins from data
    validate = crossval[i]
    learn = crossval - validate
    For each p in hyperparameters
        model = BuildClassifier(learn, classifier, p)
        cv_auc[p, i] = ApplyClassifier(validate, model)
    auc = compute mean performance over cross-validation bins
    return hyperparameters[max(auc)]
BuildClassifier(data, classifier, para)
    # Train classifier on data with hyperparameters = para
ApplyClassifier(data, model)
    # Compute AUC of model on data
```

Fig. 3. Outline of the experimental evaluation of 22 classifiers over 10 NASA MDP data sets.

The K* classifier does not require model selection and the number of neighbors has been varied in the range $[1, 3, 5, \ldots, 15]$ for k-NN.

Model selection for neural networks requires defining the number of hidden layers as well as nodes per layer. A single hidden layer of $[4, 5, \ldots, 28]$ nodes has been considered for MLP networks whereby each individual architecture is assessed with different weight decay parameters of 0.1 and 0.2 to limit the influence of noninformative features [5]. In addition, a Bayesian learning paradigm toward neural network construction (MLP-2) has been appraised [39]. Finally, the number of cluster centers per class has been varied from 1 to 10 for RBFnet.

The major degrees of freedom of an SVM-type model are the kernel function as well as a regularization parameter, commonly denoted by C. A radial basis function kernel has been considered for SVM and LS-SVM, which is the most popular choice in the literature. Consequently, the width of the kernel function and C have been tuned by means of a multilevel grid search with exponentially refined parameter grids to achieve a broad coverage of the parameter space as well as an intensive exploration of promising regions [63]. L-SVM is a linear classifier without kernel function and requires tuning of the regularization parameter. A range from $\log(C) = [-6, -5, \ldots, 20]$ has been evaluated. The LP classifier exhibits no additional parameters and does not require model selection, whereas VP incorporates a polynomial kernel function for which degree has to be determined. Values of 1 to 6 have been studied.

Model selection for C4.5 and CART involves deciding upon a pruning strategy. We have considered unpruned trees as well as pruned trees with varying confidence level $(0.05, 0.1, \ldots, 0.7)$, each time with and without Laplacian smoothing [46] and subtree raising. The ADTree classifier is trained by a boosting-based algorithm offering the number of iterations as tuning parameter. Following [21], settings of 10 to 50 iterations have been evaluated.

With respect to ensemble classifiers, LMT generally requires determination of the number of boosting iterations. However, it has been reported that this setting is irrelevant if the final classifier is augmented by pruning [36]. Consequently, we have used the default pruning strategy with an overall number of 100 boosting iterations. Two hyperparameters have been considered for RndFor, namely, the number of trees as well as the number of attributes used to grow each individual tree. A range of [10, 50, 100, 250, 500, 1,000] trees has been assessed, as well as three different settings for the number of randomly selected attributes per tree $\left([0.5; 1, 2] \cdot \sqrt{M}\right)$, whereby M denotes the number of attributes within the respective data set (see also [7]).

References

[1] C. Andersson, "A Replicated Empirical Study of a Selection Method for Software Reliability Growth Models," *Empirical Software Eng.*, vol. 12, no. 2, pp. 161-182, 2007.

[2] C. Andersson and P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Software Eng.*, vol. 33, no. 5, pp. 273-286, May 2007.

[3] B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen, "Benchmarking State-of-the-Art Classification Algorithms for Credit Scoring," *J. Operational Research Soc.*, vol. 54, no. 6, pp. 627-635, 2003.

[4] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751-761, Oct. 1996.

[5] C.M. Bishop, *Neural Networks for Pattern Recognition.* Oxford Univ. Press, 1995.

[6] A.P. Bradley, "The Use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145-1159, 1997.

[7] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[8] L.C. Briand, V.R. Basili, and C.J. Hetmanski, "Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components," *IEEE Trans. Software Eng.,* vol. 19, no. 11, pp. 1028-1044, Nov. 1993.

[9] L.C. Briand, W.L. Melo, and J. Wüst, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects," *IEEE Trans. Software Eng.,* vol. 28, no. 7, pp. 706-720, July 2002.

[10] M. Chapman, P. Callis, and W. Jackson, "Metrics Data Program," *NASA IV and V Facility,* http://mdp.ivv.nasa.gov/, 2004.

[11] J.G. Cleary and L.E. Trigg, "K*: An Instance-Based Learner Using an Entropic Distance Measure," *Proc. 12th Int'l Conf. Machine Learning,* 1995.

[12] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *J. Machine Learning Research,* vol. 7, pp. 1-30, 2006.

[13] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," *Proc. 12th Int'l Conf. Machine Learning,* 1995.

[14] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification,* second ed. Wiley, 2001.

[15] K. El-Emam, S. Benlarbi, N. Goel, and S.N. Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High-Risk Software Components," *J. Systems and Software,* vol. 55, no. 3, pp. 301-320, 2001.

[16] K. El-Emam, W. Melo, and J.C. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," *J. Systems and Software,* vol. 56, no. 1, pp. 63-75, 2001.

[17] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters,* vol. 27, no. 8, pp. 861-874, 2006.

[18] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases: An Overview," *AI Magazine,* vol. 17, no. 3, pp. 37-54, 1996.

[19] N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Software Eng.,* vol. 25, no. 5, pp. 675-689, Sept./Oct. 1999.

[20] N.E. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Trans. Software Eng.,* vol. 26, no. 8, pp. 797-814, Aug. 2000.

[21] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," *Proc. 16th Int'l Conf. Machine Learning,* 1999.

[22] Y. Freund and R.E. Schapire, "Large Margin Classification Using the Perceptron Algorithm," *Machine Learning,* vol. 37, no. 3, pp. 277-296, 1999.

[23] K. Ganesan, T.M. Khoshgoftaar, and E.B. Allen, "Case-Based Software Quality Prediction," *Int'l J. Software Eng. and Knowledge Eng.,* vol. 10, no. 2, pp. 139-152, 2000.

[24] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust Prediction of Fault-Proneness by Random Forests," *Proc. 15th Int'l Symp. Software Reliability Eng.,* 2004.

[25] M.A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 6, pp. 1437-1447, Nov./Dec. 2003.

[26] M.H. Halstead, *Elements of Software Science.* Elsevier, 1977.

[27] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2002.

[28] T.M. Khoshgoftaar and E.B. Allen, "Logistic Regression Modeling of Software Quality," *Int'l J. Reliability, Quality and Safety Eng.,* vol. 6, no. 4, pp. 303-317, 1999.

[29] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, and S.J. Aud, "Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System," *IEEE Trans. Neural Networks,* vol. 8, no. 4, pp. 902-909, 1997.

[30] T.M. Khoshgoftaar, E.B. Allen, W.D. Jones, and J.P. Hudepohl, "Classification-Tree Models of Software-Quality over Multiple Releases," *IEEE Trans. Reliability,* vol. 49, no. 1, pp. 4-11, 2000.

[31] T.M. Khoshgoftaar, A.S. Pandya, and D.L. Lanning, "Application of Neural Networks for Predicting Faults," *Annals of Software Eng.,* vol. 1, no. 1, pp. 141-154, 1995.

[32] T.M. Khoshgoftaar and N. Seliya, "Analogy-Based Practical Classification Rules for Software Quality Estimation," *Empirical Software Eng.,* vol. 8, no. 4, pp. 325-350, 2003.

[33] T.M. Khoshgoftaar and N. Seliya, "Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study," *Empirical Software Eng.,* vol. 9, no. 3, pp. 229-257, 2004.

[34] T.M. Khoshgoftaar, N. Seliya, and N. Sundaresh, "An Empirical Study of Predicting Software Faults with Case-Based Reasoning," *Software Quality J.,* vol. 14, no. 2, pp. 85-111, 2006.

[35] A.G. Koru and H. Liz, "An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures," *Proc. Workshop Predictor Models in Software Eng.,* 2005.

[36] N. Landwehr, M. Hall, and F. Eibe, "Logistic Model Trees," *Machine Learning,* vol. 59, no. 1, pp. 161-205, 2005.

[37] F. Lanubile and G. Visaggio, "Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned," *J. Systems and Software,* vol. 38, no. 3, pp. 225-234, 1997.

[38] J. Li, G. Ruhe, A. Al-Emran, and M. Richter, "A Flexible Method for Software Effort Estimation by Analogy," *Empirical Software Eng.,* vol. 12, no. 1, pp. 65-106, 2007.

[39] D.J.C. MacKay, "The Evidence Framework Applied to Classification Networks," *Neural Computation,* vol. 4, no. 5, pp. 720-736, 1992.

[40] O.L. Mangasarian and D.R. Musicant, "Lagrangian Support Vector Machines," *J. Machine Learning Research,* vol. 1, pp. 161-177, 2001.

[41] T.J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.,* vol. 2, no. 4, pp. 308-320, 1976.

[42] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'," *IEEE Trans. Software Eng.,* vol. 33, no. 9, pp. 637-640, Sept. 2007.

[43] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, "Assessing Predictors of Software Defects," *Proc. Workshop Predictive Software Models,* 2004.

[44] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Software Eng.,* vol. 33, no. 1, pp. 2-13, Jan. 2007.

[45] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler, "YALE: Rapid Prototyping for Complex Data Mining Tasks," *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* 2006.

[46] J. Mingers, "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Machine Learning,* vol. 4, no. 2, pp. 227-243, 1989.

[47] J.C. Munson and T.M. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Trans. Software Eng.,* vol. 18, no. 5, pp. 423-433, May 1992.

[48] I. Myrtveit and E. Stensrud, "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models," *IEEE Trans. Software Eng.,* vol. 25, no. 4, pp. 510-525, July/Aug. 1999.

[49] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models," *IEEE Trans. Software Eng.,* vol. 31, no. 5, pp. 380-391, May 2005.

[50] M.C. Ohlsson and P. Runeson, "Experience from Replicating Empirical Studies on Prediction Models," *Proc. Eighth Int'l Software Metrics Symp.,* 2002.

[51] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Trans. Software Eng.,* vol. 22, no. 12, pp. 886-894, Dec. 1996.

[52] N. Ohlsson, A.C. Eriksson, and M. Helander, "Early Risk-Management by Identification of Fault Prone Modules," *Empirical Software Eng.,* vol. 2, no. 2, pp. 166-173, 1997.

[53] A.A. Porter and R.W. Selby, "Evaluating Techniques for Generating Metric-Based Classification Trees," *J. Systems and Software,* vol. 12, no. 3, pp. 209-218, 1990.

[54] F. Provost and T. Fawcett, "Robust Classification for Imprecise Environments," *Machine Learning,* vol. 42, no. 3, pp. 203-231, 2001.

[55] J.B. Robert, "A Priori Tests in Repeated Measures Designs: Effects of Nonsphericity," *Psychometrika,* vol. 46, no. 3, pp. 241-255, 1981.

[56] J. Sayyad Shirabad and T.J. Menzies, "The PROMISE Repository of Software Engineering Databases," School of Information Technology and Eng., Univ. of Ottawa, http://promise.site.uottawa.ca/SERepository, 2005.

[57] N.F. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Trans. Software Eng.,* vol. 18, no. 5, pp. 410-422, May 1992.

[58] R.W. Selby and A.A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis," *IEEE Trans. Software Eng.,* vol. 14, no. 12, pp. 1743-1756, Dec. 1988.

[59] M. Shepperd and G. Kadoda, "Comparing Software Prediction Techniques Using Simulation," *IEEE Trans. Software Eng.,* vol. 27, no. 11, pp. 1014-1022, Nov. 2001.

[60] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Trans. Software Eng.,* vol. 23, no. 11, pp. 736-743, Nov. 1997.

[61] J.A.K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters,* vol. 9, no. 3, pp. 293-300, 1999.

[62] M.E. Tipping, "The Relevance Vector Machine," *Advances in Neural Information Processing Systems 12,* S.A. Solla, T.K. Leen, and K.-R. Müller, eds., pp. 652-658, MIT Press, 2000.

[63] T. Van Gestel, J.A.K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle, "Benchmarking Least Squares Support Vector Machine Classifiers," *Machine Learning,* vol. 54, no. 1, pp. 5-32, 2004.

[64] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M.D. Backer, and R. Haesen, "Mining Software Repositories for Comprehensible Software Fault Prediction Models," *J. Systems and Software,* vol. 81, no. 5, pp. 823-839, 2008.

[65] J.H. Zar, *Biostatistical Analysis,* fourth ed. Prentice Hall, 1999.

[66] H. Zhang and X. Zhang, "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'," *IEEE Trans. Software Eng.,* vol. 33, no. 9, pp. 635-637, Sept. 2007.

[67] S. Zhong, T.M. Khoshgoftaar, and N. Seliya, "Analyzing Software Measurement Data with Clustering Techniques," *IEEE Intelligent Systems,* vol. 19, no. 2, pp. 20-27, Mar./Apr. 2004.

Stefan Lessmann received the MSc and PhD degrees in business administration from the University of Hamburg, Germany, in 2001 and 2007, respectively, where he is currently a lecturer in information systems. His research interests include the development and application of predictive methods in various domains ranging from customer relationship management and empirical software engineering to financial markets. He is a student member of the IEEE.

Bart Baesens received the MSc and PhD degrees in applied economic sciences from the Katholieke Universiteit Leuven (K.U.Leuven), Belgium, in 1998 and 2003, respectively. He is currently an assistant professor at K.U.Leuven and the Vlerick Leuven Ghent Management School, Leuven, and a lecturer at the University of Southampton, United Kingdom. His research interests include classification, rule extraction, neural networks, support vector machines, data mining, and credit scoring.

Christophe Mues received the PhD degree in applied economic sciences from the Katholieke Universiteit Leuven (K.U.Leuven), Belgium, in 2002. He is a lecturer (assistant professor) in the School of Management at the University of Southampton, United Kingdom. Prior to his appointment at the University of Southampton, he was a researcher at K.U.Leuven. His research interests include the verification and validation of knowledge-based systems, decision tables and diagrams, and data mining techniques and applications in various areas ranging from credit scoring and credit risk management to the software engineering domain.

Swantje Pietsch received the MSc degree in business management with majors in information technology, industrial management, and logistics from the University of Hamburg, Germany, in 2006. She is currently an external PhD student at the Institute of Information Systems at the University of Hamburg. Since January 2007, she has been a business analyst at Shell. Her research focuses on novel data mining methods for solving complex classification and regression tasks.