Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models

Baljinder Ghotra, Shane McIntosh, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
{ghotra, mcintosh, ahmed}@cs.queensu.ca

***Abstract*—Defect prediction models help software quality assurance teams to effectively allocate their limited resources to the most defect-prone software modules. A variety of classification techniques have been used to build defect prediction models ranging from simple (e.g., logistic regression) to advanced techniques (e.g., Multivariate Adaptive Regression Splines (MARS)). Surprisingly, recent research on the NASA dataset suggests that the performance of a defect prediction model is not significantly impacted by the classification technique that is used to train it. However, the dataset that is used in the prior study is both: (a) noisy, i.e., contains erroneous entries and (b) biased, i.e., only contains software developed in one setting. Hence, we set out to replicate this prior study in two experimental settings. First, we apply the replicated procedure to the same (known-to-be noisy) NASA dataset, where we derive similar results to the prior study, i.e., the impact that classification techniques have appear to be minimal. Next, we apply the replicated procedure to two new datasets: (a) the cleaned version of the NASA dataset and (b) the PROMISE dataset, which contains open source software developed in a variety of settings (e.g., Apache, GNU). The results in these new datasets show a clear, statistically distinct separation of groups of techniques, i.e., the choice of classification technique has an impact on the performance of defect prediction models. Indeed, contrary to earlier research, our results suggest that some classification techniques tend to produce defect prediction models that outperform others.**

I. Introduction

A disproportionate amount of the cost of developing software is spent on maintenance [15]. Fixing defects is a central software maintenance activity. However, before defects can be fixed, they must be detected. Software Quality Assurance (SQA) teams are dedicated to this task of defect detection.

Defect prediction models can be used to assist SQA teams with defect detection. Broadly speaking, defect prediction models are trained using software metrics (e.g., size and complexity metrics) to predict whether software modules will be defective or not in the future. Knowing which software modules are likely to be defect-prone before a system has been deployed allows practitioners to more efficiently and effectively allocate SQA effort.

To perform defect prediction studies [21, 58], researchers have explored the use of various classification techniques to train defect prediction models [12, 34]. For example, in early studies, researchers used simple techniques like logistic regression [2, 7, 47] and linear regression [24, 65] to train defect prediction models. In more recent work, researchers have used more advanced techniques like Multivariate Adaptive Regression Splines (MARS) [5, 27], Personalized Change Classification (PCC) [27], and Logistic Model Trees (LMT) [51, 53]. Ensemble methods that combine different machine learning techniques have also been explored [14, 61]. Moreover, researchers have started to develop context-sensitive techniques that are aware of the peculiarities of software defects [5, 35].

Despite recent advances in machine learning, studies suggest that the classification technique used to train defect prediction models has little impact on its performance [34, 38, 56]. For example, Lessmann *et al.* [34] conducted a study comparing the performance of 22 different classification techniques on the NASA corpus. Their results show that the performance of 17 of the 22 classification techniques are statistically indistinguishable from each other.

However, the NASA corpus that was used in the prior work is noisy and biased. Indeed, Shepperd *et al.* [57] found that the original NASA corpus contains several erroneous and implausible entries. This noise in the studied corpus may have led prior studies to draw incorrect conclusions. Furthermore, the studied corpus only contains proprietary software developed within one organization. Software developed in a different setting (e.g., open source) may lead to different conclusions.

Therefore, we set out to revisit the findings of Lessmann *et al.* [34] both in the original, known-to-be noisy setting (Section IV), and in two additional settings (Section V). We find that we can indeed replicate the initial results of Lessmann *et al.* [34] when the procedure is reapplied to the known-to-be noisy NASA corpus. On the other hand, the results of our experiment in two additional settings seem to contradict Lessmann *et al.*'s early results. We structure our extended replication by addressing the following two research questions:

RQ1 Do the defect prediction models of different classification techniques still perform similarly when trained using the cleaned NASA corpus?
No, we find a clear separation of classification techniques when models are trained using the cleaned NASA corpus [57]. The Scott-Knott statistical test [26] groups the classification techniques into four statistically distinct ranks. Our results show that models trained using LMT and statistical techniques like Simple Logistic tend to outperform models trained using clustering, rule-based, Support Vector Machines (SVM), nearest neighbour, and neural network techniques.

RQ2 Do the defect prediction models of different classification techniques still perform similarly when trained using a corpus of open source systems?
No, Scott-Knott test results show that, similar to RQ1, classification techniques like LMT, Simple Logistic (and their combination with ensemble methods) tend to produce defect prediction models that outperform models trained using clustering, rule-based, SVM, nearest neighbour, and neural network techniques.

In summary, contrary to prior results, our study shows that there are statistically significant differences in the performance of defect prediction models that are trained using different classification techniques. We observe that classification techniques are grouped into largely consistently ranks in both the cleaned NASA and PROMISE corpora. Indeed, some techniques tend to produce defect prediction models that outperform models that are trained using other techniques. Given the ease of access to such advanced techniques nowadays in the toolboxes of researchers (e.g., `R` and `Weka`), we encourage researchers to explore the various techniques available in such toolboxes instead of relying on prior findings to guide their choice of classification technique.

Paper Organization

The remainder of the paper is organized as follows. Section II describes the classification techniques that we used in our study. Section III discusses the design of our case study. Section IV discusses the results of our replication study on the original, known-to-be noisy NASA corpus, while Section V presents the results of our two research questions. Section VI discloses the threats to the validity of our study. Section VII surveys related work. Finally, Section VIII draws conclusions and describes promising directions for future work.

II. Classification Techniques

In this section, we briefly explain the eight families of classification techniques that are used in our study. Table I provides an overview of each technique.

A. Statistical Techniques

Statistical techniques are based on a probability model [31]. These techniques are used to find patterns in datasets and build diverse predictive models [4]. Instead of simple classification, statistical techniques report the probability of an instance belonging to each individual class (i.e., defective or not) [31].

In this paper, we study the Naive Bayes and Simple Logistic statistical techniques. *Naive Bayes* is a probability-based technique that assumes that all of the predictors are independent of each other. *Simple Logistic* is a generalized linear regression model that uses a logit link function.

B. Clustering Techniques

Clustering techniques divide the training data into small groups such that the similarity within groups is more than across the groups [23]. Clustering techniques use distance and similarity measures to find the similarity between two objects to group them together.

In this paper, we study the K-means and Expectation Maximization clustering techniques. *K-means* divides the data into k clusters and centroids are chosen randomly in an iterative manner [22]. The value of k impacts the performance of the technique [33]. We experiment with four different k values (i.e., 2, 3, 4, and 5), and found that $k = 2$ tends to perform the best. We also study the *Expectation Maximization* [16] (EM) technique, which automatically splits a dataset into an (approximately) optimal number of clusters [5].

C. Rule-Based Techniques

Rule-based techniques transcribe decision trees using a set of rules for classification. The transcription is performed by creating a separate rule for each individual path starting from the root and ending at each leaf of the decision tree [31].

In this paper, we study the Repeated Incremental Pruning to Produce Error Reduction (Ripper) and RIpple DOwn Rules (Ridor) rule-based techniques. *Ripper* [10] is an inductive rule-based technique that creates series of rules with pruning to remove rules that lead to lower classification performance [33]. *Ridor* [18] is a rule-based decision tree technique where the decision tree consists of four nodes: a classification, a predicate function, a true branch, and a false branch. Each instance of the testing data is pushed down the tree, following the true and false branches at each node using predicate functions. The final outcome is given by the majority class of the leaf node [33].

D. Neural Networks

Neural networks are systems that change their structure according to the flow of information through the network during training [59]. Neural network techniques are repeatedly run on training instances to find a classification vector that is correct for each training set [31].

In this paper, we study the Radial Basis Functions neural network technique. *Radial Basis Functions* [8] consists of three different layers: an input layer (which consists of independent variables), output layer (which consists of the dependent variable) and the layer which connects the input and output layer to build a model [47].

E. Nearest Neighbour

Nearest neighbour (a.k.a., lazy-learning) techniques are another category of statistical techniques. Nearest neighbour learners take more time in the testing phase, while taking less time than techniques like decision trees, neural networks, and Bayesian networks during the training phase [31].

In this paper, we study the KNN nearest neighbour technique. *KNN* [11] considers the K most similar training examples to classify an instance. KNN computes the Euclidean distance to measure the distance between instances [34]. We find $K = 8$ to be the best-performing K value of the five tested options (i.e., 2, 4, 6, 8, and 16).

TABLE I
OVERVIEW OF THE STUDIED CLASSIFICATION TECHNIQUES.

Family	Technique	Abbreviation
Statistical Techniques	Naive Bayes	NB
	Simple Logistic	SL
Clustering Techniques	K-means	K-means
	Expectation Maximization	EM
Rule-Based Techniques	Repeated Incremental Pruning to Produce Error Reduction	Ripper
	Ripple Down Rules	Ridor
Neural Networks	Radial Basis Functions	RBFs
Nearest Neighbour	K-Nearest Neighbour	KNN
Support Vector Machines	Sequential Minimal Optimization	SMO
Decision Trees	J48	J48
	Logistic Model Tree using Logistic Regression	LMT
Ensemble Methods using LMT, NB, SL, SMO, and J48	Bagging	Bag+LMT, Bag+NB, Bag+SL, Bag+SMO and Bag+J48
	Adaboost	Ad+LMT, Ad+NB, Ad+SL, Ad+SMO and Ad+J48
	Rotation Forest	RF+LMT, RF+NB, RF+SL, RF+SMO, and RF+J48
	Random Subspace	Rsub+LMT, Rsub+NB, Rsub+SL, Rsub+SMO, and Rsub+J48

F. Support Vector Machines

Support Vector Machines (SVMs) use a hyperplane to separate two classes (i.e., defective or not). The number of features in the training data does not affect the complexity of an SVM model, which makes SVM a good fit for experiments where there are fewer training instances than features [31].

In this paper, we study the Sequential Minimal Optimization (SMO) SVM technique. *SMO* analytically solves the large Quadratic Programming (QP) optimization problem which occurs in SVM training by dividing the problem into a series of possible QP problems [49].

G. Decision Trees

Decision trees use feature values for the classification of instances. A feature in an instance that has to be classified is represented by each node of the decision tree, while the assumption values taken by each node is represented by each branch. The classification of instances is performed by following a path through the tree from root to leaf nodes by checking feature values against rules at each node. The root node is the node that best divides the training data [31].

In this paper, we study the Logistic Model Tree (LMT) and J48 decision tree techniques. Similar to model trees (i.e., regression trees with regression functions), *LMT* [32] is a decision tree like structure with logistic regression functions at the leaves [32]. *J48* [50] is a C4.5-based technique that uses information entropy to build the decision tree. At each node of the decision tree, a rule is chosen by C4.5 such that it divides the set of training samples into subsets effectively [54].

H. Ensemble Methods

Ensemble methods combine different base learners together to solve one problem. Models trained using ensemble methods typically generalize better than those trained using the stand-alone techniques [64].

In this paper, we study the Bagging, Adaboost, Rotation Forest, and Random Subspace ensemble methods. *Bagging* (Bootstrap Aggregating) [6] is designed to improve the stability and accuracy of machine learning algorithms. Bagging predicts an outcome multiple times from different training sets that are combined together either by uniform averaging or with voting [61]. *Adaboost* [17] performs multiple iterations each time with different example weights, and gives a final prediction through combined voting of techniques [61]. *Rotation Forest* [52] applies a feature extraction method to subsets of instances to reconstruct a full feature set for each technique in the ensemble. *Random Subspace* [25] creates a random forest of multiple decision trees using a random selection attribute approach. A subset of instances is chosen randomly from the selected attributes and assigned to the learning technique [61].

III. STUDY DESIGN

In this section, we discuss the studied corpora, and our approach to constructing and evaluating defect prediction models to address our research questions. Figure 1 provides an overview of the steps in our approach.

A. Studied Corpora

In order to replicate the case study of Lessmann *et al.* [34], we use the original, known-to-be noisy NASA corpus. Moreover, in order to perform our extended study, we use the cleaned version of the NASA corpus as provided by Shepperd *et al.* [57] (RQ1), and the PROMISE corpus [1] (RQ2). An overview of the studied corpora is shown in Table II. The studied corpora that we use in our study are available online, enabling further replication (and extension) of our results.

[1] https://code.google.com/p/promisedata/

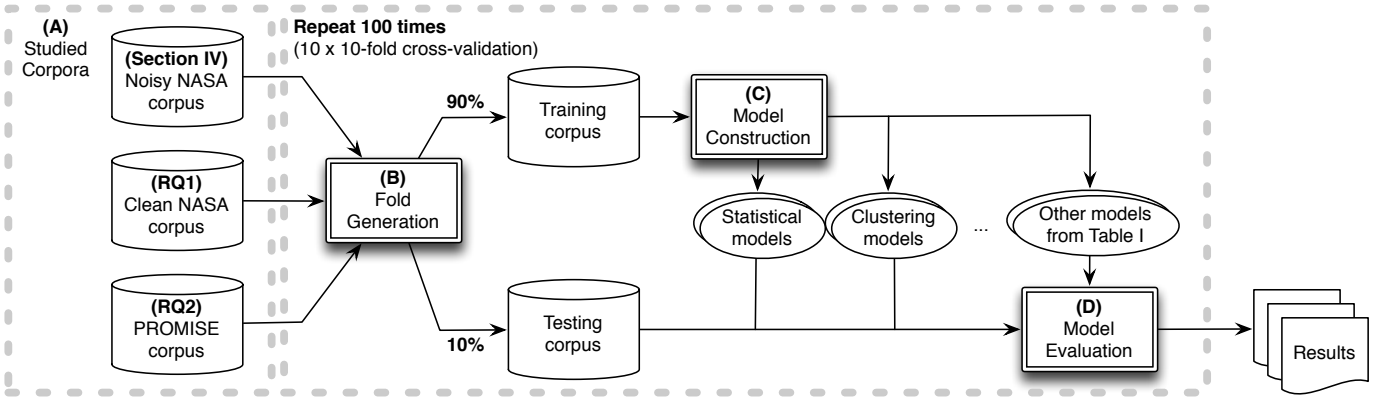

Fig. 1. An overview of our model construction and evaluation approach.

TABLE II
AN OVERVIEW OF THE STUDIED CORPORA.

Corpus	Dataset	No. of Modules	Defective	Defective (%)
Noisy NASA (Section IV)	CM1	505	48	9.5
	JM1	10,878	2,102	19.3
	KC1	2,107	325	15.4
	KC3	458	43	9.3
	KC4	125	61	48.8
	MW1	403	31	7.6
	PC1	1,107	76	6.8
	PC2	5,589	23	0.4
	PC3	1,563	160	10.2
	PC4	1,458	178	12.2
Clean NASA (RQ1)	CM1	327	42	12.8
	JM1	7,720	1,612	20.8
	KC1	1,162	294	25.3
	KC3	194	36	18.5
	MW1	250	25	10
	PC1	679	55	8.1
	PC2	722	16	2.2
	PC3	1,053	130	12.3
	PC4	1,270	176	13.8
PROMISE (RQ2)	Ant 1.7	745	166	22.3
	Camel 1.6	965	188	19.5
	Ivy 1.4	241	16	6.6
	Jedit 4	306	75	24.5
	Log4j 1	135	34	25.2
	Lucene 2.4	340	203	59.7
	Poi 3	442	281	63.6
	Tomcat 6	858	77	8.9
	Xalan 2.6	885	441	46.4
	Xerces 1.3	453	69	15.2

B. Fold Generation

The defect prediction models are trained using the 10-fold cross-validation approach, which divides an input dataset into 10 folds of equal size. Of the 10 folds, 9 are allocated to the training corpus (90%), and 1 fold is set aside for testing (10%). The training corpus is used to train the models using different classification techniques, whereas the testing corpus is used to analyze model performance. This process is repeated ten times, using each fold as the testing corpus once. To further validate our results, we repeat the entire 10-fold process ten times (100 total iterations).

C. Model Construction

In order to train our defect prediction models, we use implementations of the classification techniques of Section II provided by the WEKA [62] machine learning toolkit.

The process of training defect prediction models using clustering techniques is different than for the other techniques. Specifically, we adopt the approach of Bettenburg *et al.* in order to train models using clustering techniques [5]. The training corpus is divided into clusters and a classification model is trained within each cluster using the Naive Bayes technique. To determine which model to use in order to classify any particular row in the testing corpus, we use the Euclidean distance between each testing data point and the various clusters. The model of the cluster that is the minimum distance away is used to predict the final outcome for that row.

D. Model Evaluation

To compare the performance of defect prediction models, we use the Area Under the receiver operating characteristic Curve (AUC) [63], which plots the false positive rate (i.e., $\frac{FP}{FP+TN}$) against the true positive rate (i.e., $\frac{TP}{FN+TP}$). Larger AUC values indicate better performance. AUC values above 0.5 indicate that the model outperforms random guessing.

Scott-Knott test. We use the Scott-Knott test [26] to group classification techniques into statistically distinct ranks ($\alpha = 0.05$). The Scott-Knott test uses hierarchical cluster analysis to partition the classification techniques into ranks. The Scott-Knott test starts by dividing the classification techniques into two ranks on the basis of mean AUC values (i.e., mean AUC of ten 10-fold runs for each classification technique). If the divided ranks are statistically significantly different, then Scott-Knott recursively executes again within each rank to further divide the ranks. The test terminates when ranks can no longer be divided into statistically distinct ranks [29, 39].

We used the Scott-Knott test to overcome the confounding issue of overlapping groups that are produced by several other post hoc tests, such as Nemenyi's test [13], which was used by the original study. Nemenyi's test produces overlapping groups of classification techniques, implying that there exists no

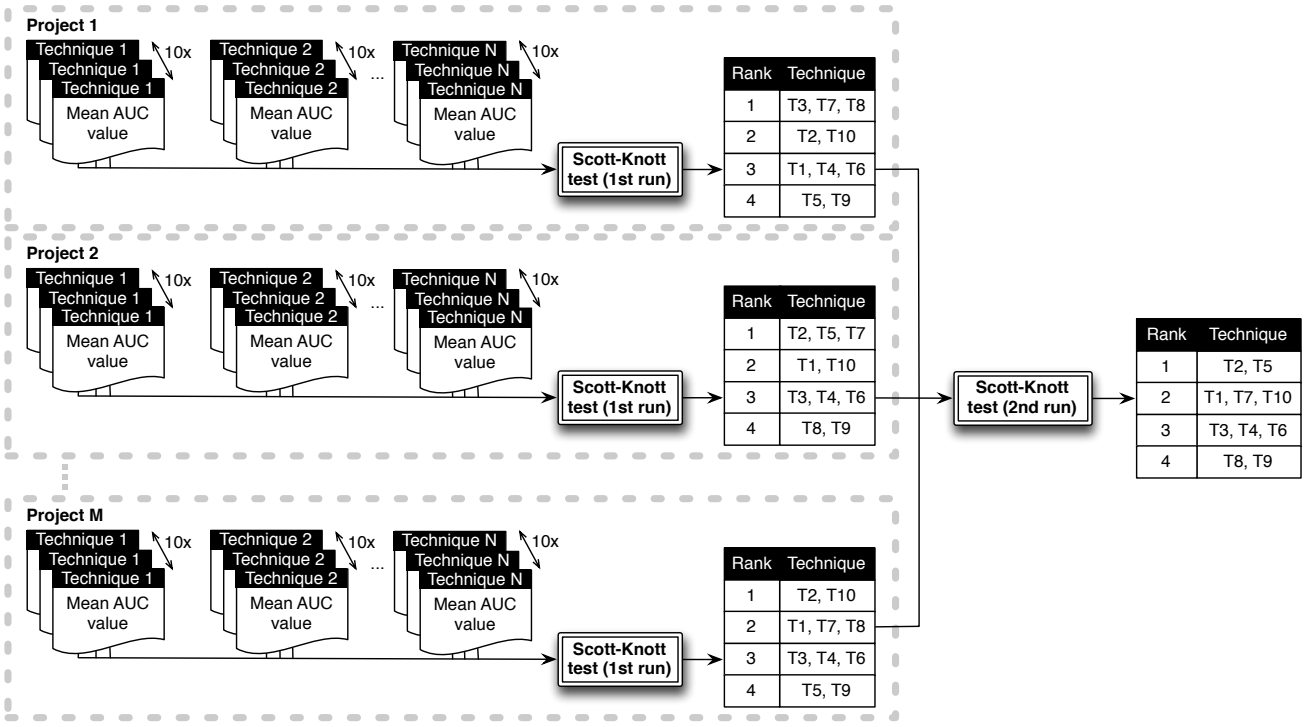

Fig. 2. Our approach for ranking classification techniques using a double Scott-Knott test.

TABLE III
METRICS USED TO TRAIN DEFECT PREDICTION MODELS IN THE NASA CORPUS.

Category	Metric	Definition	Rationale
McCabe Software Metrics	cyclomatic_complexity, cyclomatic_density, design_complexity essential complexity and pathological_complexity	Measures of the branching complexity of the software module.	Complex software modules may be more prone to defects [36].
Halstead attributes	content, difficulty, effort, error_est, length, level, prog_time, volume, num_operands, num_operators, num_unique_operands, num_unique_operators	Estimates of the complexity of reading a software module based on the vocabulary used (e.g., number of operators and operands).	Software modules that are complex to understand may increase the likelihood of incorrect maintenance, and thus, increase defect proneness [28].
LOC counts	LOC_total, LOC_blank, LOC_comment, LOC_code_and_comment, LOC_executable and Number_of_lines	Measures of the size of a software module.	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [30].
Miscellaneous	branch_count, call_pairs, condition_count, decision_count, decision_density, design_density, edge_count, essential_density, parameter_count, maintenance_severity, modified_condition_count, multiple_condition_count, global_data_density, global_data_complexity, percent_comments, normalized_cyclomatic_complexity and node_count	Metrics that are not well defined metrics in the MDP database [37].	N/A

statistically significant difference among the defect prediction models trained using many different classification techniques.

To address our research questions, Figure 2 provides an overview of our Scott-Knott test approach. To find statistically distinct ranks of classification techniques, we performed a double Scott-Knott test. The first run of the Scott-Knott test is run over each individual project. We input the 10 mean AUC values of the 10 different 10-fold cross-validation runs of each classification technique to the Scott-Knott test to find the statistical distinct ranks of classification techniques within each project. In the second run, for each classification technique, we have ten different Scott-Knott ranks (i.e., one from each project), which we input into another Scott-Knott test to find the final statistically distinct ranks of techniques.

Our approach of using a double Scott-Knott test ensures that our analysis recognizes techniques that perform well across projects, independent of their actual AUC value. For example, it might be the case that one classification technique achieves an AUC of 0.5 for project A and an AUC of 0.9 for project B. At first glance, it might seem that such a technique is not a strong performing technique. However, if an AUC of 0.5 is the best AUC value of the studied techniques for project A, and

TABLE IV
THE STUDIED TECHNIQUES RANKED ACCORDING TO OUR DOUBLE SCOTT-KNOTT TEST ON THE KNOWN-TO-BE NOISY NASA CORPUS.

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Ad+NB, EM, RBFs, Ad+SL, RF+NB, Rsub+NB, Ad+SMO, K-means, KNN, NB, SL, Bag+NB, Rsub+SL, Ad+J48, Rsub+J48, Ad+LMT, LMT, RF+SL, Bag+SL, RF+J48, Rsub+LMT,Bag+J48, RF+LMT and Bag+LMT	3.2	3.03	0.97
2	Rsub+SMO, SMO, RF+SMO, Ridor, Bag+SMO, Ripper and J48	8	7.91	1.04

AUC of 0.9 is the best AUC for project B, then that particular classification technique is the top-performing technique. The first Scott-Knott test creates the project-level ranking for each technique using the AUC values, then the second Scott-Knott test creates a global ranking of techniques using their project-level rankings.

IV. REPLICATION STUDY

In this section, we discuss the results of our replication study on the known-to-be noisy NASA corpus. The NASA corpus provides several software metrics that can be used to predict defect-prone modules. Table III describes the metrics present in the NASA corpus.

Results. **Similar to prior work, our replication using the known-to-be noisy NASA data yields a pattern of two statistically distinct groups of classification techniques**. Table IV shows that 24 of the 31 studied classification techniques appear in the first Scott-Knott group. Figure 3 shows the AUC values for each of the studied techniques.

Figure 4 highlights the importance of our double Scott-Knott approach. Looking at project `JM1` and project `PC4` (or `KC4`), many of the best techniques for project `JM1` perform worse than some of the worst-performing techniques on the `PC4` (or `KC4`) project. Our double Scott-Knott approach is able to account for such peculiarities by considering the rank in the second Scott-Knott iteration.

Our replication yields a similar pattern of classification techniques as was found in the previous NASA study, i.e., 24 of the 31 studied techniques end up in the same statistical group. Most classification techniques produce defect prediction models that have statistically indistinguishable performance from one another.

V. EXTENDED CASE STUDY

In this section, we present the results of the extended Lessmann *et al.* [34] replication with respect to our two research questions. Generally speaking, our goal is to study whether the previously reported results of Lessmann *et al.* [34] (and confirmed by us in Section IV) still hold in two new experimental settings, which we formulate as our two research questions. Below, we discuss our approach for addressing each research question and the results.

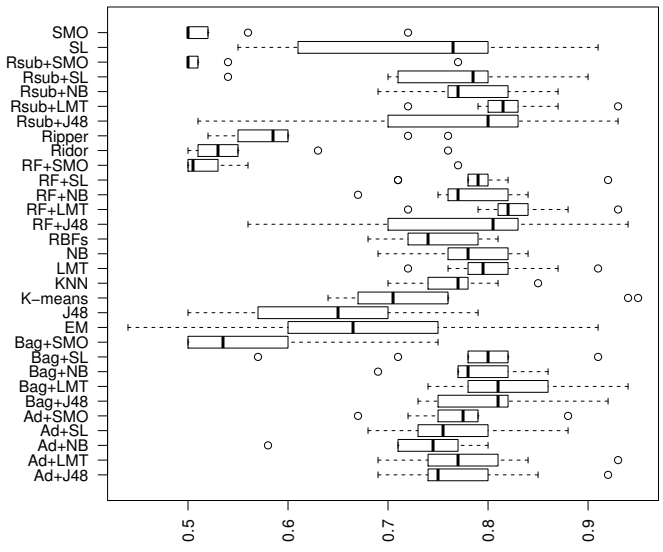

Fig. 3. AUC values for the studied classification techniques on the known-to-be noisy NASA corpus.

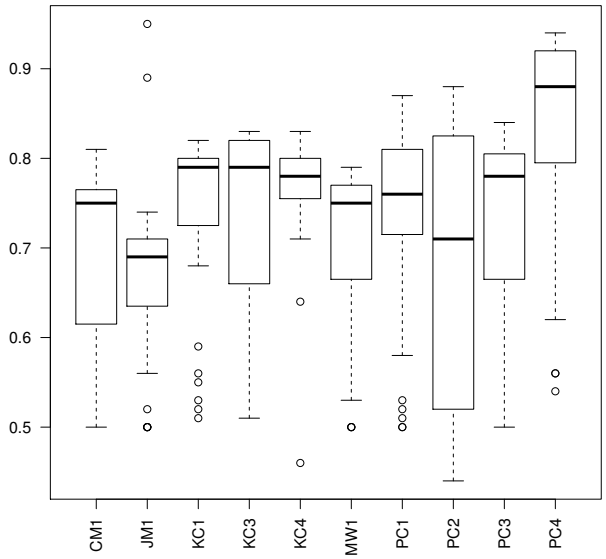

Fig. 4. AUC values of all classification techniques that are applied to the different projects in the known-to-be noisy NASA corpus.

TABLE V
DIFFERENCES IN THE NOISY AND CLEAN NASA CORPORA.

Dataset	Instances before cleaning	Instances after cleaning	Instances removed by cleaning
CM1	505	327	178
JM1	10,878	7,720	3,158
KC1	2,107	1,162	945
KC3	458	194	264
MW1	403	250	153
PC1	1,107	679	428
PC2	5,589	722	4,867
PC3	1,563	1,053	510
PC4	1,458	1,270	188

TABLE VI
CLEANING CRITERIA APPLIED TO THE NOISY NASA CORPUS BY SHEPPERD *et al.* [57].

Criterion	Data Quality Category	Explanation
1	Identical cases	Instances that have identical values for all metrics including class label.
2	Inconsistent cases	Instances that satisfy all conditions of Case 1, but where class labels differ.
3	Cases with missing values	Instances that contain one or more missing observations.
4	Cases with conflicting feature values	Instances that have 2 or more metric values that violate some referential integrity constraint. For example, LOC_TOTAL is less than Commented LOC. However, Commented LOC is a subset of LOC_TOTAL.
5	Cases with implausible values	Instances that violate some integrity constraint. For example, value of LOC=1.1.

(RQ1) Do the defect prediction models of different classification techniques still perform similarly when trained using the cleaned NASA corpus?

Approach. Our extended study starts with using the cleaned version [57] of nine of the NASA projects that were used by Lessmann *et al.* [34]. The purpose of using the cleaned version is to find whether the removal of noisy instances has an impact on the performance of defect prediction models produced using the studied classification techniques. An overview of the cleaned version of the NASA dataset is shown in Table II. We are only able to use 9 of the 10 NASA projects because the KC4 dataset does not contain any instances after cleaning. The differences in the noisy and clean NASA corpora are highlighted in Table V. The preprocessing criteria that Shepperd *et al.* [57] applied to the noisy NASA corpus are presented in Table VI.

Results. **We find four statistically distinct ranks of classification techniques when models are trained using datasets from the cleaned NASA corpus.** Table VII shows that there is a clear separation of classification techniques into four Scott-Knott ranks in the cleaned NASA corpus. This is in stark contrast with the two groups of classification techniques that Lessmann *et al.* [34] found (and that Section IV confirmed).

The techniques are divided into four distinct ranks in which ensemble techniques (i.e., RF+a base learner), decision trees (i.e., LMT), statistical techniques (i.e., Simple Logistic and Naive Bayes), neural networks (i.e., RBFs), and nearest neighbour (i.e., KNN) outperformed models trained using rule-based techniques (i.e., Ripper and Ridor), clustering techniques (i.e., K-means and EM) and SVM (i.e., SMO). Our results in Table VII show that the prediction models trained using ensemble techniques (i.e., RF+a base learner), decision trees (i.e., LMT), and statistical techniques (i.e., simple logistic) also outperformed models trained using nearest neighbour (i.e., KNN) and neural networks (i.e., RBFs). Furthermore, we find that prediction models trained using the combination of LMT and simple logistic with ensemble methods like bagging, rotation forest, and random subspace produce defect prediction models that perform the best (i.e., achieve the highest average Scott-Knott rank) when compared to other combinations, i.e, Naive Bayes, J48, and SMO with ensemble methods.

TABLE VII
THE STUDIED TECHNIQUES RANKED ACCORDING TO THE DOUBLE SCOTT-KNOTT TEST ON THE CLEAN NASA CORPUS.

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Bag+J48, LMT, RF+J48, SL, Bag+SL, RF+SL, Rsub+SL, RF+LMT, Rsub+LMT, and Bag+LMT	1.22	1.24	0.15
2	RBFs, Ad+NB, KNN, Ad+SL, RF+NB, Rsub+NB, NB, Bag+NB, Ad+SMO, Ad+J48, Ad+LMT, and Rsub+J48	2.61	2.62	0.51
3	Ripper, EM, J48, and K-means	5.72	5.58	0.67
4	Rsub+SMO, RF+SMO, SMO, Ridor, and Bag+SMO	7.66	7.4	0.42

Since four statistically distinct ranks of classification techniques emerge, we conclude that the used classification technique has a statistically significant impact on the performance of defect prediction models trained using the datasets from the cleaned NASA corpus.

(RQ2) Do the defect prediction models of different classification techniques still perform similarly when trained using a corpus of open source systems?

Approach. In an effort to further generalize our extended study results, we replicate the Lessmann *et al.* study [34] using another ten datasets from the PROMISE corpus. The selected PROMISE datasets were used in several previous studies [47, 48]. The PROMISE datasets provide different metrics than the NASA corpus. Table II provides an overview of the PROMISE corpus. Table VIII describes the metrics provided by the PROMISE corpus.

Results. **We find that, similar to our results of RQ1, a pattern of four statistically distinct ranks of classification techniques emerges in the PROMISE corpus as well**.

TABLE VIII
METRICS USED TO TRAIN DEFECT PREDICTION MODELS IN THE PROMISE CORPUS.

Category	Metric	Definition	Rationale
Size	loc	Measures of the size of a software module.	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [30].
CK	wmc, dit, cbo, noc, lcom, rfc, ic, cbm, amc, lcom3	Measures of the complexity of a class within an object-oriented system design.	More complex classes may be more prone to defects [19].
McCabe's Complexity	max_cc and avg_cc	Measures of the branching complexity of the software module.	Complex software modules may be more prone to defects [36].
QMOOD	dam, moa, mfa, cam, npm	Measures of the behavioural and structural design properties of classes, objects, and the relations between them [55].	Higher QMOOD metrics imply higher class quality. On the other hand, lower QMOOD metrics imply lower class quality [2], which may lead to defects.
Martin's metrics	ca, ce	Measures of the relationship between software components, including calls as well as number of instances [45].	Higher values indicate low encapsulation, reusability, and maintainability [45], which may lead to defects.

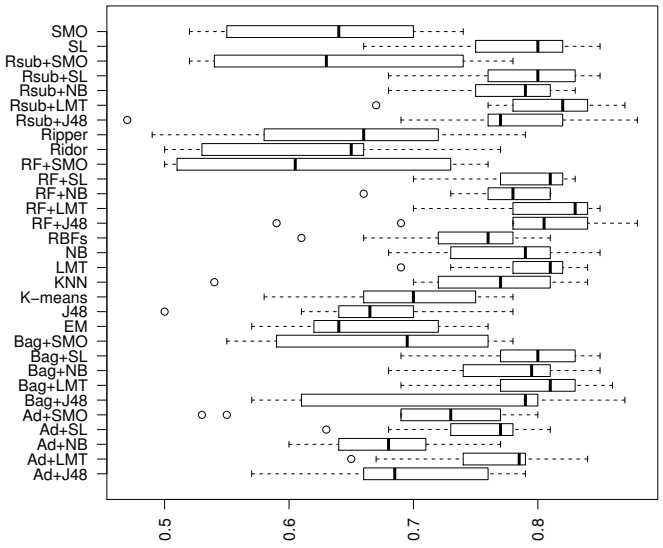

Fig. 5. AUC values of the classification techniques on the PROMISE corpus.

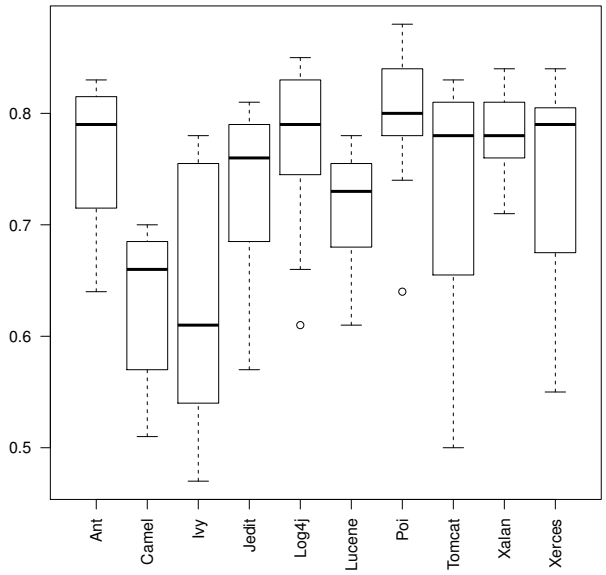

Fig. 6. AUC values of all classification techniques that are applied to the different projects in the PROMISE corpus.

Table IX shows the four statistically distinct ranks generated by applying the double Scott-Knott test.

The results show that prediction models trained using decision trees, statistical techniques, K-nearest neighbour, and neural networks outperform models trained using clustering techniques, rule-based techniques, and SVM. Furthermore, we find that when ensemble methods are used to combine LMT and simple logistic, the best performance is achieved when compared to the other classification techniques. Figure 5 shows the mean AUC values of techniques on the PROMISE dataset. We observe that the mean AUC value of the best (RF+LMT) and the worst (RF+SMO) classification techniques in the PROMISE corpus have higher values when compared to best (Bag+LMT) and worst (Rsub+SMO) classification techniques in the cleaned NASA corpus.

Figure 6 again emphasizes the importance of using the double Scott-Knott approach in our analysis. The best performing technique for the `Camel` project is well below the worst-performing technique for other projects, such as `Poi` and `Xalan`.

We observed some common patterns in the ranks of both the NASA and PROMISE corpora. Our results shows that LMT when combined with ensemble methods (i.e., bagging, random subspace, and rotation forest) achieve top-rank performance in both corpora. Also, ensembles of simple logistic along with stand-alone LMT and simple logistic appear in the top Scott-Knott rank in both corpora. Furthermore, clustering techniques (i.e., EM and K-means), rule-based techniques (Ripper and Ridor), and SVM (i.e., SMO) appear in the lowest Scott-Knott rank in both corpora.

TABLE IX
THE STUDIED TECHNIQUES RANKED ACCORDING TO THE DOUBLE SCOTT-KNOTT TEST ON THE PROMISE CORPUS.

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Rsub+J48, SL, Rsub+SL, Bag+SL, LMT, RF+SL, RF+J48, Bag+LMT, Rsub+LMT, and RF+LMT	1.7	1.63	0.33
2	RBFs, Bag+J48, Ad+SL, KNN, RF+NB, Ad+LMT, NB, Rsub+NB, and Bag+NB	2.8	2.84	0.41
3	Ripper, EM, J48, Ad+NB, Bag+SMO, Ad+J48, Ad+SMO, and K-means	5.1	5.13	0.46
4	RF+SMO, Ridor, SMO,and Rsub+SMO	6.5	6.45	0.25

Complementing our results of the cleaned NASA study, we again find four statistically distinct ranks of classification techniques, which leads us to conclude that the used classification technique has a statistically significant impact on the performance of defect prediction models trained using the datasets of the PROMISE corpus as well.

VI. THREATS TO VALIDITY

In this section, we discuss the threats to the validity of our case study.

A. Construct Validity

The datasets that we analyze are part of two corpora (i.e., NASA and PROMISE), which each provide a different set of predictor metrics. Since the metrics vary, this is another point of variation between the studied corpora, which may explain some of the differences that we observe when drawing comparisons across the corpora. Nevertheless, our main findings still hold, i.e., there are statistically significant differences between the performance of defect prediction models trained using various classification techniques within the cleaned NASA dataset and the PROMISE one as well.

B. Internal Validity

The tuning of the parameters of the K-means ($k = 2$) and KNN ($K = 8$) techniques that we used may influence our results. For example, this tuning may impact the performance of models trained using these classification techniques when compared to default parameters. To combat this bias, we experimented with several configurations (k = 2, 3, 4, and 5; and K = 2, 4, 6, 8, and 16) that achieved similar results. Nonetheless, a more carefully controlled experiment of the influence of configuration parameters is needed.

C. External Validity

We performed our experiments on 29 datasets, which may threaten the generalization of the results of our study. However, these datasets are part of the popular NASA and PROMISE corpora, which has been used by several studies that benchmark defect prediction models in the past [34, 47, 48].

The consistency of our results are based on two studied data corpora. The results may vary in other corpora that we have not analyzed. On the other hand, the two corpora contain software systems that are developed in both proprietary and open source paradigms, which helps to counter potential bias. Nonetheless, additional replication studies may prove fruitful.

VII. RELATED WORK

In order to train defect prediction models, defective software modules are predicted using software metrics, such as object-oriented metrics (e.g., Chidamber and Kemerer (CK) metrics [9]), process metrics [3, 24, 40, 42] (e.g., number of changes, recent activity), product metrics [20, 41, 43, 44, 60] (e.g., lines of code and complexity), historical metrics [20, 46] (e.g., code change), and structural metrics [1] (e.g., cyclomatic complexity or coupling). In this paper, we used several types of metrics based on their availability in the studied corpora. A promising direction for future research would be to explore the impact that different metrics have on our findings (e.g., perhaps our findings are sensitive to the type of metrics that are available for training our models).

There exists several evaluation metrics to measure the performance of classification models, such as accuracy [61], AUC [34, 47], error sum, median error, error variance, and correlation [5]. In this paper, we use the AUC because it is threshold-independent.

In addition to Lessmann *et al.* [34], there have been several studies that compare the performance of classification techniques across projects in search of the top-performing classification techniques. Panichella *et al.* [47] conducted a study to compare the performance of statistical, neural networks, and decision trees classification techniques using the PROMISE corpus. Bettenburg *et al.* [5] conducted a study using the PROMISE corpus in order to compare the performance of classification models that are trained using clustering and statistical techniques, reporting that clustering techniques tend to perform better than models that are trained using statistical techniques. However, Lemon [33] found contradictory results to Bettenburg *et al.* [5] when building classification models on NASA datasets using clustering and statistical techniques.

The key differences between such prior work and this paper are the following:

1) To the best of our knowledge, this is the first study to replicate a prior analysis using the cleaned NASA corpus.
2) Most prior studies use either the NASA or the PROMISE corpus — they rarely combine both corpora together for a single analysis, hence the generalization of prior results is hindered.
3) Most prior studies use a post hoc statistical tests to compare techniques. However, the results of such tests are not as clear cut as the Scott-Knott tests, which produces non-overlapping ranks.
4) To the best of our knowledge, this study is the first to extrapolate project-specific analyses to a corpus-level by

ranking classification techniques at the project-level and lifting it using a double Scott-Knott test.

5) To the best of our knowledge, this is the first study to use several recent classification techniques (e.g., LMT), while ensuring that we capture techniques from a variety of families.

VIII. Conclusions

A previous study on the NASA corpus by Lessmann *et al.* [34] showed that the performance of defect prediction models that are trained using various classification techniques do not vary much, implying that the use of any technique should be sufficient when predicting defects. In this paper, we revisit this very impactful finding, since it has influenced a large amount of follow-up literature after its publication — many researchers opt not to investigate the performance of various classification techniques on their datasets.

Our replication of Lessmann *et al.* [34] yields the same results even when using a more advanced statistical ranking technique (i.e., the Scott-Knott test). However, when using a cleaned version of the NASA corpus and when using another corpus (the PROMISE dataset, which provides different metrics than the NASA corpus does), we found that the results differ from those of Lessmann *et al.* [34], i.e., classification techniques produce defect prediction models with significantly different performance.

Like any empirical study, our study has several limitations (*cf.* Section VI). However, we would like to emphasize that we do not seek to claim the generalization of our results. Instead, the key message of our study is that there are datasets where there are statistically significant differences between the performance of models that are trained using various classification techniques. Hence, we recommend that software engineering researchers experiment with the various available techniques instead of relying on specific techniques, assuming that other techniques are not likely to lead to statistically significant improvements in their reported results. Given the ubiquity of advanced techniques in commonly-used analysis toolkits (e.g., `R` and `Weka`), we believe that our suggestion is a rather simple and low-cost suggestion to follow.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17.

[2] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[3] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in *Proceedings of the International Workshop on Principles of Software Evolution*, 2007, pp. 11–18.

[4] A. Berson, S. Smith, and K. Thearling, "An overview of data mining techniques," *Building Data Mining Application for CRM*, 2004.

[5] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Think locally, act globally: Improving defect and effort prediction models," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 2012, pp. 60–69.

[6] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[7] L. C. Briand, J. Wüst, and H. Lounis, "Replicated case studies for investigating quality factors in object-oriented designs," *Empirical software engineering*, vol. 6, no. 1, pp. 11–58, 2001.

[8] M. D. Buhmann, "Radial basis functions," *Acta Numerica 2000*, vol. 9, pp. 1–38, 2000.

[9] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[10] W. W. Cohen, "Fast Effective Rule Induction," in *Proceedings of the International Conference on Machine Learning*, 1995, pp. 115–123.

[11] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[12] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[13] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[14] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[15] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT professional*, vol. 2, no. 3, pp. 17–23, 2000.

[16] C. Fraley and A. E. Raftery, "Bayesian regularization for normal mixture estimation and model-based clustering," *Journal of Classification*, vol. 24, no. 2, pp. 155–181, 2007.

[17] Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm," in *Proceedings of the International Conference on Machine Learning*, 1996, pp. 148–156.

[18] B. R. Gaines and P. Compton, "Induction of ripple-down rules applied to modeling large databases," *Journal of Intelligent Information Systems*, vol. 5, no. 3, pp. 211–228, 1995.

[19] B. Goel and Y. Singh, "Empirical investigation of metrics for fault prediction on object-oriented software," in *Computer and Information Science*. Springer, 2008, pp.

255–265.

[20] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[21] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[22] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Advances in Neural Information Processing Systems*, 2004, pp. 281–288.

[23] K. Hammouda and F. Karray, "A comparative study of data clustering techniques," *Fakhreddine Karray University of Waterloo, Ontario, Canada*, 2000.

[24] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 78–88.

[25] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[26] E. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: A package for performing the scott-knott clustering algorithm in r," *Trends in Applied and Computational Mathematics*, vol. 15, no. 1, pp. 003–017, 2014.

[27] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013*. IEEE, 2013, pp. 279–289.

[28] K. Kaur, K. Minhas, N. Mehan, and N. Kakkar, "Static and dynamic complexity analysis of software metrics," *World Academy of Science, Engineering and Technology*, vol. 56, p. 2009, 2009.

[29] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: a case study of android game apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 610–620.

[30] A. G. Koru, D. Zhang, K. El Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 293–304, 2009.

[31] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[32] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine Learning*, vol. 59, no. 1-2, pp. 161–205, 2005.

[33] B. Lemon, "The effect of locality based learning on software defect prediction," Ph.D. dissertation, West Virginia University, 2010.

[34] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[35] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 343–351.

[36] T. Menzies, J. S. Di Stefano, M. Chapman, and K. McGill, "Metrics that matter," in *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*. IEEE Computer Society, 2002, p. 51.

[37] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[38] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[39] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 537–551, 2013.

[40] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 181–190.

[41] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 580–586.

[42] ——, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 284–292.

[43] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 452–461.

[44] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 886–894, 1996.

[45] M. F. Oliveira, R. M. Redin, L. Carro, L. d. C. Lamb, and F. R. Wagner, "Software quality metrics and their impact on embedded software," in *Proceedings of the 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, 2008*. IEEE Computer Society, 2008, pp. 68–77.

[46] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software

systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.

[47] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*. IEEE, 2014, pp. 164–173.

[48] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 409–418.

[49] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in kernel methods*. MIT Press, 1999, pp. 185–208.

[50] J. R. Quinlan, *C4. 5: programs for machine learning*. Morgan kaufmann, 1993, vol. 1.

[51] J. Ratzinger, T. Sigmund, and H. C. Gall, "On the relation of refactorings and software defect prediction," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008, pp. 35–38.

[52] J. Rodríguez, L. Kuncheva, and C. Alonso, "Rotation forest: A new classifier ensemble method." *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[53] P. S. Sandhu, S. Kumar, and H. Singh, "Intelligence system for software maintenance severity prediction," *Journal of Computer Science*, vol. 3, no. 5, p. 281, 2007.

[54] L. Sehgal, N. Mohan, and P. S. Sandhu, "Quality prediction of function based software using decision tree approach," in *Proceedings of the International Conference on Computer Engineering and Multimedia Technologies (ICCEMT)*, 2012, pp. 43–47.

[55] A. Shaik, C. Reddy, B. Manda, C. Prakashini, and K. Deepthi, "Metrics for object oriented design software systems: a survey," *Journal of Emerging Trends in Engineering and Applied Sciences*, vol. 1, no. 2, pp. 190–198, 2010.

[56] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, 2014.

[57] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, no. 9, pp. 1208–1215, 2013.

[58] E. Shihab, "An exploration of challenges limiting pragmatic software defect prediction," Ph.D. dissertation, Queens University, 2012.

[59] Y. Singh and A. S. Chauhan, "Neural networks in data mining," *Journal of Theoretical and Applied Information Technology*, vol. 5, no. 6, pp. 36–42, 2009.

[60] R. Subramanyam and M. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[61] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *Journal of Information & Computational Science*, vol. 8, no. 16, pp. 4241–4254, 2011.

[62] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[63] S. Wu and P. Flach, "A scored auc metric for classifier evaluation and selection," 2005.

[64] Z.-H. Zhou, "Ensemble learning," in *Encyclopedia of Biometrics*. Springer, 2009, pp. 270–273.

[65] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," in *Proceedings of the International Workshop on Predictor Models in Software Engineering*, 2007.