Robust Prediction of Fault-Proneness by Random Forests

Lan Guo† Yan Ma‡ Bojan Cukic† Harshinder Singh‡

†Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506
{lan, cukic}@csee.wvu.edu

‡Department of Statistics
West Virginia University, Morgantown, WV 26506
{yma, hsingh}@stat.wvu.edu

Abstract

Accurate prediction of fault prone modules (a module is equivalent to a C function or a C++ method) in software development process enables effective detection and identification of defects. Such prediction models are especially beneficial for large-scale systems, where verification experts need to focus their attention and resources to problem areas in the system under development.

This paper presents a novel methodology for predicting fault prone modules, based on random forests. Random forests are an extension of decision tree learning. Instead of generating one decision tree, this methodology generates hundreds or even thousands of trees using subsets of the training data. Classification decision is obtained by voting. We applied random forests in five case studies based on NASA data sets. The prediction accuracy of the proposed methodology is generally higher than that achieved by logistic regression, discriminant analysis and the algorithms in two machine learning software packages, WEKA [32] and See5 [5]. The difference in the performance of the proposed methodology over other methods is statistically significant. Further, the classification accuracy of random forests is more significant over other methods in larger data sets.

1 Introduction

As software systems are increasingly deployed in mission critical applications, it has become imperative that they operate reliably and in accordance with the requirements. Software assurance requires extensive and expensive assessment, ranging from manual inspection to automatic formal methods. Software quality models, which automatically predict fault prone modules, enable verification experts to concentrate their attention and resources at problem areas of the system under development. Thus, applying software quality models early in the software life cycle contributes to efficient defect removal and results in delivering more reliable software products.

The basic hypothesis of software quality prediction is that a module currently under development is likely to be fault prone, if a module with the similar product or process metrics in an earlier project (or release) was fault prone [20]. Therefore, the information from the previous project can be used in making a prediction for the current project, if the development environment is stable. This methodology is very useful for large-scale projects or projects with multiple releases.

Many modeling techniques have been developed and applied to software quality prediction. These include, logistic regression [7], discriminant analysis [19, 28], the discriminative power techniques [29], Optimized Set Reduction [9], artificial neural network [18], fuzzy classification [10], Bayesian Belief Networks [12] , genetic algorithms [6], classification trees [13, 17, 30, 31], and, recently, Dempster-Shafer Belief Networks [14]. For all these software quality models, there is a tradeoff between the defect detection rate and the overall prediction accuracy. Thus, a performance comparison of various models, if based on only one criterion (either the defect detection rate or the overall accuracy), may render the comparison only partially relevant. A model can be considered superior over its counterparts if it has both a higher defect detection rate, and a higher overall accuracy. About 65-75% of critical modules and non-fault prone modules were correctly predicted in [16, 17, 19]. The decision tree [30] correctly predicted

79.3% of high development effort or fault prone modules, while the trees generated from the best parameter combinations correctly identified 88.4% of those modules on the average. The discriminative power techniques correctly classified 75 of 81 fault free modules, and 21 of 31 faulty modules [29]. In one case study, within five common classification techniques: Pareto classification, classification trees, factor-based discriminant analysis, fuzzy classification, and neural network, fuzzy classification appears to yield best results with a defect detection rate of 86% [11]. Since most of these studies have been performed using different data sets, reflecting different software development environments and processes, the final judgement on "the best" fault-prone module prediction method is difficult to make. In addition, some papers do not report associated overall prediction accuracy, which makes objective comparisons even more difficult.

In this paper, we introduce a novel software quality prediction methodology based on Random Forests [3, 8]. Random forests, proposed by Breiman in 2001, are a recent extension of decision tree learning. Instead of generating one decision tree, this methodology generates hundreds or even thousands of trees using subsets of the training data. Classification decision is obtained by voting. We compare the proposed methodology with many existing approaches using the same data sets. Our approach to predicting fault prone modules has the following characteristics. (1) The methodology is general and not restricted to particular metrics or research objectives. (2) It runs efficiently on large data sets. (3) It is more robust to outliers and noise compared to other classifiers. Hence, it is especially valuable for large-scale systems. The prediction accuracy of the proposed methodology is higher as compared to logistic regression, discriminant analysis, or the algorithms employed in two data ming software packages, WEKA and See5, for the same data sets obtained from NASA. The difference in the performance of the proposed methodology over other methods is statistically significant.

The remainder of this paper is organized as follows. Section 2 describes Random Forests. Section 3 introduces the data sets used in the case studies. Section 4 defines measurement parameters used in the experiments. Section 5 outlines the major steps of the proposed methodology. Section 6 presents the experimental procedures and results. Sections 7 and 8 provide a comparison of the results obtained from the proposed methodology over related work, i.e. logistic regression, discriminant analysis, two machine learning software packages, See5 [5] and WEKA [32], and NASA's ROCKY toolset [25] for the same data sets. Section 9 summarizes the unique aspects and advantages of the proposed methodology. Finally, Section 10 concludes the paper.

2 Random Forests

A random forest is a classifier consisting of a collection of tree-structured classifiers [3]. The random forest classifies a new object from an input vector by examining the input vector on each tree in the forest. Each tree casts a unit vote at the input vector by giving a classification. The forest selects the classification having the most votes over all the trees in the forest.

Each tree is grown as follows:

- If the number of cases in the training set is N, sample N cases at random, with *replacement* from the original data. This sample will be the training set for growing the tree.
- At each node, m predictors are randomly selected out of the M input variables ($m \ll M$) and the best split on these m predictors is used to split the node. The value of m is held constant during the forest growing. By default, $m = \sqrt{M}$ (to achieve near optimal results).
- Each tree is grown to the largest extent possible. There is no pruning.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This **oob (out-of-bag) data** is used as a test set to get an unbiased estimate of the classification error. Therefore, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. The out-of-bag estimates are unbiased [8].

Random forest is a good candidate for software quality prediction, especially for large-scale systems, because [3]:

- It is reported to be consistently accurate when compared with current classification algorithms.
- It runs efficiently on large data sets.
- It has an efficient method for estimating missing data and retains accuracy when a large portion of the data is missing.
- It gives estimates of which attributes are important in the classification.
- It is more robust with respect to noise compared to other methods.

3 Public Domain Defect Data

An important feature of this study is repeatability. The data sets used in the case studies are five mission critical

Project	# modules	% with defects	Language	Developed at	Notes
CM1	496	9.8%	C	Location 1	A NASA spacecraft instrument
JM1	10,885	19.3%	C	Location 2	Realtime predictive ground system: Uses simulations to generate predictions
KC1	2,109	15.5%	C++	Location 3	Storage management for receiving and processing ground data
KC2	520	20.4%	C++	Location 3	Science data processing; another part of the same project as KC1; different personnel from KC1. Shared some third-party software libraries with KC1, but no other software overlap.
PC1	1,109	6.9%	C	Location 4	Flight software for earth orbiting satellite

Total modules: 15,119

Figure 1. Data Sets Used in the Cases Studies

NASA projects [26] (see Figure 1). They are freely available via the web site of NASA's Metrics Data Program (MDP) [1].

Each data set contains 21 software metrics, including McCabe [22, 23, 24], Halstead [15], Line Count, and Branch Count. Metric descriptions are listed in Table 1. Each data set contains additional two attributes: *Error Rate* (number of defects in the module) and *Defect* (whether or not the module has any defects).

4 Measurement Parameters

In this study we are interested in predicting whether or not the module contains any defects, instead of how many defects it contains. Software metrics serve as predictors. The predicted variable is *Defect*. Figure 2 presents a defect prediction sheet.

		Defect present?	
		No	Yes
Defect Predicted?	No	TN=true negative	FN=false negative
	Yes	FP=false positive	TP=true positive
	No prediction	NP_1	NP_2

Figure 2. A defect prediction sheet

Specificity is used to define the rate of the defect module detection. In the literature, it is also referred to as *Probability of Detection (PD)* [25]:

$$Specificity(PD) = \frac{TP}{FN + TP + NP_2} \qquad (1)$$

Similarly, *Sensitivity* is defined as the portion of the correct classification of non-fault prone modules:

Table 1. Metric Descriptions of Five Data Sets

Metric Type	Metric	Definition
McCabe	v(G)	Cyclomatic Complexity
	ev(G)	Essential Complexity
	iv(G)	Design Complexity
	LOC	Lines of Code
Derived Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Count
	E	Effort
	B	Effort Estimate
	T	Programming Time
Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAnd-Comment	Lines of Code and Comment
Basic Halstead	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

$$Sensitivity = \frac{TN}{TN + FP + NP_1} \quad (2)$$

The overall prediction accuracy is measured by Acc:

$$Acc = \frac{TN + TP}{TN + FN + FP + TP + NP_1 + NP_2} \quad (3)$$

Another parameter is *Probability of False alarm (PF)*. It represents the ratio of non-fault prone modules predicted as fault prone modules:

$$PF = \frac{FP}{TN + FP + NP_1} \quad (4)$$

5 Methodology

We applied the random forest classifier to the prediction of fault prone modules. In our case studies, the five NASA data sets were collected from mission critical projects. The fault-prone modules constitute only a small portion in the data sets. As mentioned earlier, there is a tradeoff between the overall accuracy Acc and the defect detection rate PD. Random forests, trying to minimize overall error rate, will keep the error rate low on the large class while letting the smaller classes have a larger error rate. This will obviously impose problems for software quality prediction, because many fault-prone modules will be misclassified as non-fault prone ones and hence might be released into the later phase of the software life cycle.

We solve this problem by changing the default *cutoff* of random forests. *Cutoff* is a vector of length equal to the number of predicted classes. If the predicted variable has for instance two classes, the *cutoff* is a vector of length 2. Cutoff vectors are used in the classification decision by random forests. The "winning" class for an observation is the one with the maximum (positive) difference between the proportion of its votes and its cutoff value. Suppose the predicted variable has n classes $1, 2, \cdots, n$, and a random forest votes $p_1, p_2, \cdots, p_n$ for the corresponding classes. If the *cutoff* value for the corresponding class is $c_1, c_2, \cdots, c_n$, we have

$$winning\ class = i,$$

$$where\ p_i - c_i = \max(p_1 - c_1, p_2 - c_2, \cdots, p_n - c_n)$$

By default, majority voting is applied, that is, *cutoff* is (0.5, 0.5) in the case of 2 predicted classes. By generating user-defined cutoffs for random forests, different thresholds are set for the classes and the prediction errors can be balanced toward the way it is preferred. In this way, random forests can be tuned to obtain a wide range of overall accuracy Acc and defect detection rate PD, and to achieve optimal classification results.

In our methodology, we first generate a matrix for *cutoff* values, and then apply the random forest classifier with each individual *cutoff* vector in the matrix to predicting fault prone modules. Each tree in the forest is built from randomly selected two-thirds of the data set, and the remaining one-third of the data are used as the test set for validation. Experimental details are described in the next section.

6 Experiments and Results

In our experiments, we used the random forest classifier in a statistical software package, R [2]. For each of the five NASA data sets, we generated user-defined cutoff vectors. Since our classification problem has two possible outcomes (fault-prone and non-fault prone), the vector contains two cutoff values, (c_1, c_2). c_1 and c_2 are the thresholds for the classes of non-fault prone and fault-prone, respectively. A random forest votes p_1 and p_2 for the corresponding classes. In principle, the random forest will classify a module, for instance, as non-fault prone, if $p_1 - c_1 > p_2 - c_2$.

For each available data set, we generated 45 cutoff vectors in the form of 45×2 matrix. All values in the matrix are of the form $k * 0.1$, where $k \in \{1, 2, ..., 9\}$. The sum of the two cutoff values in the same row can be no greater than 1, to meet the requirement for *cutoff* values in random forests. Each row in the matrix represents a pair of thresholds for one classification result. In this way, we get combinations of *cutoff* values that result in different classifications by the random forest. Through experimentation, we determined that additional cutoff vectors (with smaller increment values) give us redundant results.

Forty-five cutoff vectors provide us with 45 classification results for each data set. Each result is generated by a random forest with 500 trees (the default value in random forest literature). During the classification, 5 variables (also the default value) were randomly selected to split each node in the tree.

Predictions of fault proneness by random forests on the five NASA data sets are depicted in Figure 3. Due to the space constraints, we do not show the results of random forests on each individual data set. From Figure 3, we can observe that random forests can achieve up to 87% defect detection rate PD with generally less than 25% of false alarms PF on the five NASA data sets. The overall accuracy Acc of random forests on the five data sets is mostly within 75% to 94%. For the five data sets, the *cutoff* values of (0.8, 0.1) and (0.9, 0.1) result in high probability of detecting fault prone modules, PD, with high overall accuracy, Acc. For instance, The highest defect detection rate PD, 87%, is achieved on KC2 project with the cutoff value (0.9, 0.1). The "outlier" values of false alarms, PF, in Figure 3 are mainly due to the prediction results on JM1 data set, which is the largest project containing the noisiest data.

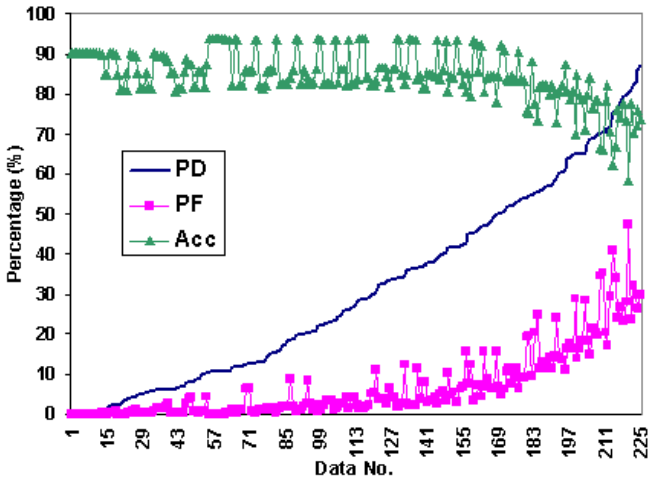

Figure 3. Prediction of random forests on the five NASA data sets (sorted by PD)

We also applied random forests to ranking the importance of attributes in the classification [3] for the five data sets. The five most important (informative) attributes for each data set are listed in Table 2. (See Table 1 for the metric descriptions). We also performed a set of experiments by using only the selected five attributes to predict fault prone modules for each data set. The prediction results by using 5 predictors are comparable to the prediction results by using all 21 predictors (shown in Figure 3) for each data set. This is not a surprising result and it has been reported in the literature. Due to the space constraints, we do not show the classification results of 5 predictors in this paper. Obviously, a reduced set of predictors will result in smaller decision trees and faster classifications.

Table 2. Five Most Important Attributes in Each Data Set

Data Set	Most Important Attributes
CM1	UniqOp, V, LOBlank, N, E
JM1	V, LOC, E, T, I
KC1	LOC, I, T, E, V
KC2	UniqOpnd, TotalOpnd, iv(G), LOBlank, LOCode
PC1	I, UniqOpnd, LOComment, T, N

We can observe from Table 2 that the metrics selected from JM1 and KC1 are the same. Both of these two projects are or relate to ground systems (refer to Figure 1). Most selected metrics in Table 2 are Halstead metrics and Line Count. Generally, important metrics for predicting fault-prone modules are different from project to project, as a consequence of different software development environments and processes. This observation suggests that software quality models should be tuned on the local project. This is consistent with the remark of Menzies *et. al.* that defect detectors built from local data perform *better than* detectors from remote projects [27]. Thus, whenever possible, software development organizations should be building and analyzing their own data repositories for defect detection.

7 Methods Compared

In this study, we compare the performance of Random Forests with other statistical methods such as logistic regression, discriminant analysis, algorithms in two machine learning software packages See5 [5] and WEKA [32], as well as NASA's ROCKY toolset [25] on the same data sets obtained from NASA.

We used 10-fold cross-validation to evaluate the prediction accuracy of the compared methods. The 10-fold cross-validation was run for at least 10 times in each experiment. The result with the least variance was chosen as the final result. In this way, we are confident that the accuracy estimation has low bias and low variance [21].

7.1 Discriminant Analysis

Discriminant analysis is a very useful tool to determine which variables discriminate between two or more naturally occurring groups. It can also be used to classify cases into two or more groups. In our study, the DISCRIM procedure (linear discriminant function) in SAS [4], a commercial statistics software, was employed as a classifier on the five NASA data sets.

7.2 Logistic Regression

Logistic regression is useful to predict a dependent variable on the basis of independents (predictors). For comparison, the LOGISTIC procedure in SAS was used as a classifier to predict fault prone modules for the five NASA data sets.

7.3 See5/C5

See5/C5 is a commercial machine learning software [5]. Its earlier version is called C4.5. There are three classifiers in See5: *DecisionTree*, *RuleSet*, and *Boosting*. When See5 is invoked with the default values of all options, it constructs a decision tree for classification. Decision trees can sometimes be quite difficult to understand. An important feature of See5 is its ability to generate classifiers called *RuleSets*

Table 3. WEKA Classifiers

Applied WEKA Classifiers	Generally Recommended	Best in (*) Our Study
DecisionStump	X	X
DecisionTable	X	
HyperPipes		X
IB1		XX
IBk	X	XX
j48.J48	X	XX
j48.PART	X	
KernelDensity		X
KStar		XXX
Logistic		XX
NaiveBayesSimple		
NaiveBayes	X	XXX
ZeroR	X	
OneR	X	
SMO	X	
VotedPerceptron		XX
VF1		XXXXX
LogitBoost		XX
NeuralNetwork		
ADTree		

(*) X: among the best on one project; XX: among the best on two projects; and so on

that consist of unordered collections of (relatively) simple if-then rules, derived from the constructed decision trees. Another innovation incorporated in See5 is *adaptive boosting*. The idea is to generate several classifiers (either decision trees or rulesets) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class. The three classifiers of See5 were used to predict fault prone modules for the five NASA data sets.

7.4 WEKA

WEKA is a collection of machine learning algorithms for solving real-world data mining problems [32]. It contains 41 algorithms for classification and numeric prediction (the most important ones are explained in [33]). Out of these 41 algorithms, a total of 20 classifiers are applicable to our data sets. For each data set, the best classifiers are selected according to the $< Acc, PD >$ pair by the domination rule. For example, we use $r =< Acc, PD >$ to represent each result. Suppose we have $r_1 =< 0.6, 0.7 >$, $r_2 =< 0.5, 0.8 >$, and $r_3 =< 0.6, 0.6 >$. In this case, r_3 is dominated by r_1, and thus r_1 and r_2 are selected as the best results. The applied WEKA classifiers, the recommended important ones [33], as well as the best performers in our case studies are listed in Table 3. It can be observed from Table 3 that, although not recommended as the most reliable classifiers in [33], some algorithms, for instance *VF1*, turn out to be the best performers in our case studies.

7.5 ROCKY

ROCKY is a defect detector toolset used in experimental selection of modules for software inspection at NASA IV&V facility in Fairmont, West Virginia [25]. ROCKY detectors are built by exhaustively exploring all singleton rules of the form:

$$attribute \geq threshold$$

where *attribute* is every numeric attribute present in a data set, and *threshold* is certain percentile value of the corresponding *attribute*. ROCKY was applied to predicting fault prone modules in KC2 and JM1 data sets with all McCabe and Halstead metrics. Predictions based on individual metrics were presented in [25].

8 Evaluation

In this section, we compare the performance of Random Forests with that of other statistical methods on the five data sets. In the following comparison, the criteria are overall accuracy *(Acc)* and defect detection rate *(PD)*. The Random Forests results are chosen for comparison by the following rules. For a pair of results, the predictions by the Random Forest and the compared method, if one prediction is dominant over the other (see §7.4 for domination rule), this pair of results is selected for evaluation. Otherwise, the pair with the closest distance between the compared criterion, either Acc or PD, is selected. For clarity, we use $< Acc_{RF}, PD_{RF} >$ to represent a result from Random Forest prediction, and $< Acc_{other}, PD_{other} >$ to represent a result from other compared methods. We define $Distance_{Acc}$, $Distance_{PD}$, and $Distance$ as follows:

$$Distance_{Acc} = |Acc_{RF} - Acc_{other}|$$

$$Distance_{PD} = |PD_{RF} - PD_{other}|$$

$$Distance = \min(Distance_{Acc}, Distance_{PD})$$

The pair of results that have the minimal $Distance$ are chosen for evaluation.

We tested statistical significance of the classification results in order to compare Random Forests with other methods. The normal distribution test was performed to establish whether classification improvements of Random Forests hold at the 0.05 level of significance (95% confidence level). Tables 4, 5, 6, 7 and 8 present comparisons on each of the five available NASA data sets. In these tables, *Yes* means that the classification performance difference is significant and *No* means the difference is insignificant at 0.05 level. A $(+)$ sign means the Random Forest prediction is better than the compared method; a $(-)$ sign means that the Random Forest prediction is worse than the compared method.

8.1 JM1 Project

JM1 is the largest data set in our study. It contains information on $10,883$ modules. Due to the noise and outliers unavoidable in such a large data set, current algorithms applied on JM1 have lower prediction accuracy and defect detection rate than on the other four data sets in this study. Random forests outperform all the compared methods on JM1 (see Table 4). Random Forests have both higher overall accuracy Acc and higher defect detection rate PD than almost all the compared methods. The dominance of Random Forests over the other methods is statistically significant. Actually, the difference between the performance of Random Forests and that of other methods is very large in many cases, indicated by the large Z values ($|Z| \geq 1.96$ corresponds to the significant difference at 0.05 level). For *VF1* and *HiperPipes* algorithms of WEKA, Random Forests achieve higher overall accuracy Acc, while the two WEKA classifiers reach higher defect detection rate PD. There is no double $(-)$ sign in Table 4, which means that Random Forests are not surpassed in accuracy by any method compared on JM1.

Improvements of random forest predictions over other methods listed in Table 4 are depicted in Figure 4. From Figure 4, we can observe that Random Forests achieve both higher overall accuracy Acc and defect detection rate PD than almost all other methods compared. For instance, Random Forests have 18.5% higher defect detection rate than the *RuleSet* classifier of See5, without compromising the overall accuracy. Considering the size of JM1 project, Random Forests correctly identify 390 more fault-prone modules than the *RuleSet* classifier. In another comparison, Random Forests have 15% higher defect detection rate and 6.3% higher overall accuracy than the *VotedPerceptron* classifier of WEKA. It amounts to 315 more fault-prone modules and 685 more modules in total that are correctly classified by random forest predictions. The advantage of Random Forests is clearly manifested in this case study.

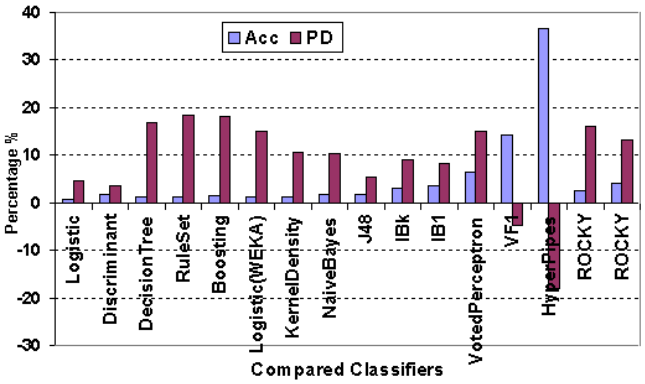

Figure 4. Improvement of random forest predictions over other methods on JM1

8.2 PC1 Project

PC1 is a fairly large data set with 1,109 modules. Random Forests have higher overall accuracy Acc and/or defect detection rate PD than the compared methods (see Table 5). Compared with the WEKA classifier *VF1*, Random Forests achieve higher overall accuracy Acc, while the WEKA classifier reaches higher defect detection rate PD. The difference in most comparisons is statistically significant. Random Forests outperform almost all the compared methods on PC1.

Improvements of random forest predictions over other methods listed in Table 5 are depicted in Figure 5. From Figure 5, we can observe that Random Forests achieve both higher overall accuracy Acc and/or defect detection rate PD than almost all other methods compared. For instance, Random Forests have 23.4 - 27.8% higher defect detection rate than the *Logistic* classifier of SAS as well as the *DecisionTree* and *Ruleset* classifiers of See5, with same or higher overall accuracy. Recall that PC1 is a safety critical project containing only 6.9% of modules with defects (refer to Figure 1). It is of great practical importance that random forests correctly identify over one-forth of total fault-prone modules more than other methods without compromising the overall accuracy.

8.3 KC1 Project

KC1 has 2,109 modules. Random Forests have higher overall accuracy Acc and/or higher defect detection rate PD than almost all the compared methods (see Table 6). The difference in most comparisons is statistically significant. Random Forests are only outperformed by the WEKA classifier *KStar* on KC1. The difference between Random Forests and *KStar* is not significant.

Table 4. Statistical Difference Test of Random Forests vs. Other Methods Using 0.05 Level of Significance on JM1

Method Compared	Software	Acc	$Acc(RF)$	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	$PD(RF)$	$\|Z_{PD}\|$	Z_{PD} Significant?
Logistic	SAS	0.658	0.664	0.93	No $(+)$	0.654	0.701	3.26	Yes $(+)$
Discriminant	SAS	0.736	0.755	3.22	Yes $(+)$	0.509	0.545	2.34	Yes $(+)$
DecisionTree	See5	0.811	0.822	2.10	Yes $(+)$	0.131	0.300	13.33	Yes $(+)$
RuleSet	See5	0.810	0.822	2.28	Yes $(+)$	0.115	0.300	14.79	Yes $(+)$
Boosting	See5	0.808	0.822	2.66	Yes $(+)$	0.118	0.300	14.51	Yes $(+)$
Logistic	WEKA	0.813	0.824	2.10	Yes $(+)$	0.117	0.266	12.28	Yes $(+)$
KernelDensity	WEKA	0.810	0.822	2.28	Yes $(+)$	0.194	0.300	7.97	Yes $(+)$
NaiveBayes	WEKA	0.803	0.822	3.59	Yes $(+)$	0.197	0.300	7.73	Yes $(+)$
j48.J48	WEKA	0.802	0.822	3.78	Yes $(+)$	0.247	0.300	3.86	Yes $(+)$
IBk	WEKA	0.761	0.793	5.67	Yes $(+)$	0.369	0.459	5.92	Yes $(+)$
IB1	WEKA	0.756	0.793	6.53	Yes $(+)$	0.376	0.459	5.45	Yes $(+)$
VotedPerceptron	WEKA	0.560	0.623	9.45	Yes $(+)$	0.604	0.754	10.42	Yes $(+)$
VF1	WEKA	0.418	0.561	21.10	Yes $(+)$	0.868	0.821	4.21	Yes $(-)$
HyperPipes	WEKA	0.195	0.561	55.67	Yes $(+)$	1.000	0.821	20.33	Yes $(-)$
ROCKY	ROCKY	0.752	0.778	4.52	Yes $(+)$	0.338	0.499	10.58	Yes $(+)$
ROCKY	ROCKY	0.540	0.581	6.09	Yes $(+)$	0.671	0.805	9.88	Yes $(+)$

Table 5. Statistical Difference Test of Random Forests vs. Other Methods Using 0.05 Level of Significance on PC1

Method Compared	Software	Acc	$Acc(RF)$	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	$PD(RF)$	$\|Z_{PD}\|$	Z_{PD} Significant?
Logistic	SAS	0.885	0.921	2.86	Yes $(+)$	0.234	0.506	3.50	Yes $(+)$
Discriminant	SAS	0.849	0.921	5.31	Yes $(+)$	0.429	0.506	0.96	No $(+)$
DecisionTree	See5	0.929	0.932	0.28	No $(+)$	0.221	0.455	3.07	Yes $(+)$
RuleSet	See5	0.932	0.932	0.00	No	0.177	0.455	3.71	Yes $(+)$
Boosting	See5	0.940	0.937	0.29	No $(-)$	0.130	0.338	3.05	Yes $(+)$
J48	WEKA	0.934	0.934	0.00	No	0.247	0.416	2.23	Yes $(+)$
IB1	WEKA	0.914	0.921	0.60	No $(+)$	0.416	0.506	1.12	No $(+)$
KStar	WEKA	0.921	0.921	0.00	No	0.273	0.506	2.96	Yes $(+)$
NaiveBayes	WEKA	0.887	0.921	2.72	Yes $(+)$	0.299	0.506	2.62	Yes $(+)$
VF1	WEKA	0.193	0.822	29.63	Yes $(+)$	0.883	0.714	2.61	Yes $(-)$

Table 6. Statistical Difference Test of Random Forests vs. Other Methods Using 0.05 Level of Significance on KC1

Method Compared	Software	Acc	$Acc(RF)$	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	$PD(RF)$	$\|Z_{PD}\|$	Z_{PD} Significant?
Logistic	SAS	0.711	0.722	0.79	No (+)	0.752	0.831	2.48	Yes (+)
Discriminant	SAS	0.790	0.795	0.40	No (+)	0.638	0.672	0.91	No (+)
DecisionTree	See5	0.848	0.862	1.29	No (+)	0.193	0.258	1.99	Yes (+)
RuleSet	See5	0.852	0.862	0.93	Yes (+)	0.187	0.258	2.18	Yes (+)
Boosting	See5	0.862	0.862	0.00	No	0.169	0.258	2.77	Yes (+)
KStar	WEKA	0.855	0.854	0.09	No (−)	0.509	0.460	1.25	No (−)
VF1	WEKA	0.195	0.722	34.35	Yes (+)	0.957	0.831	5.25	Yes (−)

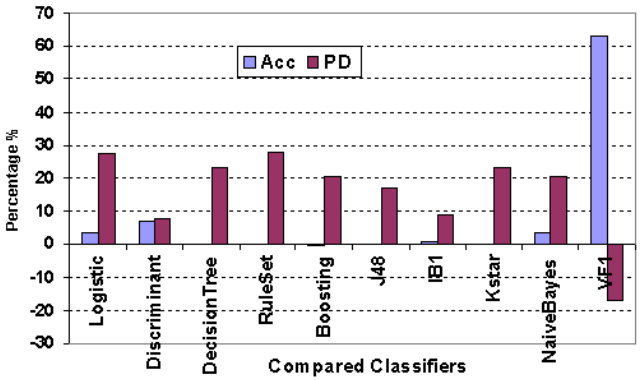

Figure 5. Improvement of random forest predictions over other methods on PC1

8.4 KC2 Project

KC2 contains 520 modules. Random Forests are significantly better than logistic regression of SAS, *VotedPerceptron* of WEKA, and NASA's ROCKY toolset on KC2. The difference between the performance of Random Forests and the rest of methods is not significant on KC2. See results in Table 7.

8.5 CM1 Project

CM1 is the smallest data set in this study, as it contains 498 modules. From the results in Table 8, there is no significant advantage of Random Forests over the compared methods. In fact, Random Forests are outperformed by *Decision Tree* of See5, WEKA classifiers *Logit Boost, Logistic Regression, KStar, and Naive Bayes*. The difference is, however, not significant.

9 Discussion

An analysis of several recent projects revealed that 20% of the modules are responsible for 80% of the malfunctions of the whole project [10]. Three data sets in our case studies demonstrate that similar 80/20 rule holds true for static artifacts, code defects (see Figure 1). The other two data sets in this study are safety critical projects. These projects understandably contain a smaller proportion of fault prone modules (7% to 10%). The goal of software quality prediction is to identify these critical modules as early as possible in the development life cycle.

This paper compares many statistical methods and machine learning algorithms. Some of them are not suitable for software quality prediction because of their low defect detection rate *PD*. Others, able to achieve over 60% defect detection rate, are candidate methods for software quality prediction. These methods include: logistic regression and discriminant analysis of SAS, the *VF1, VotedPerceptron* classifiers of WEKA, as well as NASA's ROCKY toolset. Compared with these methods, Random Forests can achieve higher overall prediction accuracy Acc and defect detection rate *PD* (*Specificity*). Random Forests work especially well on large data sets such as JM1, PC1 and KC1.

The work presented in this paper is, to the best of our knowledge, the first attempt to apply Random Forests to software quality prediction. This novel methodology has the following unique aspects and advantages:

1. It can be tuned to output a wide spectrum of defect detection rate PD and overall accuracy Acc. It allows software verification experts to choose their preferred range according to their project schedule and budget constraints. In our experiments, changing the *cutoff* values of random forests results in different classification, while changing the number of trees in the forest or the number of variables used in splitting a node do not result in different classification results.

Table 7. Statistical Difference Test of Random Forests vs. Other Methods Using 0.05 Level of Significance on KC2

Method Compared	Software	Acc	$Acc(RF)$	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	$PD(RF)$	$\|Z_{PD}\|$	Z_{PD} Significant?
Logistic	SAS	0.685	0.769	3.04	Yes $(+)$	0.849	0.849	0.00	No
Logistic	SAS	0.829	0.829	0.00	No	0.198	0.566	5.51	Yes $(+)$
Discriminant	SAS	0.821	0.823	0.08	No $(+)$	0.632	0.575	0.85	No $(-)$
DecisionTree	See5	0.819	0.819	0.00	No	0.547	0.557	0.15	No $(+)$
RuleSet	See5	0.836	0.842	0.26	No $(+)$	0.500	0.519	0.28	No $(+)$
Boosting	See5	0.835	0.842	0.31	No $(+)$	0.434	0.519	1.24	No $(+)$
LogitBoost	WEKA	0.852	0.842	0.45	No $(-)$	0.500	0.519	0.28	No $(+)$
IBk	WEKA	0.812	0.819	0.71	No $(+)$	0.509	0.557	0.83	No $(+)$
DecisionStump	WEKA	0.808	0.806	0.08	No $(-)$	0.642	0.604	0.57	No $(-)$
VotedPerceptron	WEKA	0.367	0.762	12.85	Yes $(+)$	0.849	0.858	0.23	No $(+)$
VF1	WEKA	0.586	0.737	5.15	Yes $(+)$	0.887	0.868	0.42	No $(-)$
ROCKY	ROCKY	0.727	0.737	0.36	No $(+)$	0.722	0.868	2.63	Yes $(+)$

Table 8. Statistical Difference Test of Random Forests vs. Other Methods Using 0.05 Level of Significance on CM1

Method Compared	Software	Acc	$Acc(RF)$	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	$PD(RF)$	$\|Z_{PD}\|$	Z_{PD} Significant?
Logistic	SAS	0.657	0.671	0.47	No $(+)$	0.776	0.755	0.25	No $(-)$
Discriminant	SAS	0.841	0.845	0.17	No $(+)$	0.694	0.204	4.88	Yes $(-)$
DecisionTree	See5	0.906	0.904	0.11	No $(-)$	0.204	0.040	2.48	Yes $(-)$
RuleSet	See5	0.882	0.884	0.10	No $(+)$	0.102	0.060	0.76	No $(-)$
Boosting	See5	0.892	0.892	0.00	No	0.020	0.060	1.01	No $(+)$
LogitBoost	WEKA	0.898	0.898	0.00	No	0.102	0.040	1.19	No $(-)$
Logistic	WEKA	0.894	0.894	0.00	No	0.163	0.060	1.62	No $(-)$
KStar	WEKA	0.859	0.857	0.09	No $(-)$	0.204	0.122	1.10	No $(-)$
NaiveBayes	WEKA	0.849	0.845	0.18	No $(-)$	0.306	0.204	1.16	No $(-)$
VF1	WEKA	0.339	0.671	10.48	Yes $(+)$	0.898	0.755	1.87	No $(-)$

2. It can be tuned to achieve optimal results by changing cutoff values. In this study, the optimal results have the maximal overall accuracy and defect detection rate. In fact, user-defined cutoffs are important when the classification cost is very different for each class. Therefore, if *cost* is taken into consideration in a future project, this methodology can be tuned to achieve optimal results for the new requirements.

3. It runs efficiently on large data sets. It takes several seconds to generate a random forest with 500 trees on CM1, KC1, KC2, and PC1 data sets, on a Windows XP machine with a 1.60 GHz Pentium 4 processor and 256 MB of RAM. For the largest data set JM1, it takes 7 minutes to generate a forest with 500 trees on a server with a 248MHz Ultra SPARC-II sun4u processor and 2 GB of RAM.

4. It generally has higher overall prediction accuracy Acc and defect detection rate *PD* than logistic regression, discriminant analysis, NASA's ROCKY toolset, as well as the algorithms in two machine learning software packages WEKA and See5. The difference between the performance of the proposed methodology and other methods is statistically significant.

5. It is more robust to noise and outliers than other methods. Consequently, the classification accuracy of random forests is more significant over other methods in larger data sets. As mentioned before, Random Forests can correctly identify 390 more fault-prone modules or 685 more modules in total than other methods on JM1. This result may imply considerable cost savings during software inspection, and a great improvement on the ability to deliver more reliable software products.

6. It helps in identifying most important software quality attributes in the prediction datasets.

10 Conclusions and Future Work

This paper contributes a novel methodology to the framework of software quality prediction. This methodology is based on Random Forests. The methodology presented in this paper is meaningful for real-world applications in software quality prediction. Firstly, it is more robust in respect to noise than other methods. Thus, it works especially well for large-scale software systems. Secondly, It can be tuned to output a wide spectrum of defect detection rate PD and overall accuracy Acc, which allows software verification experts to choose their preferred range according to their time and budget constraints. Our study indicates that the proposed methodology generally achieves higher prediction accuracy and defect detection rate than the compared methods on the five NASA data sets. The overall accuracy of our methodology is within 75% to 94%, and the defect detection rate is up to 87% in our case studies. We believe that applying our methodology in software development life cycle entails efficient defect removal and the ability to deliver more reliable software products.

An important feature of this study is that it has compared results from different machine learners on several data sets. The results in our case studies provide a "proving ground" for new methods. Thus, new methods can be evaluated based on previous ones in a clear and consistent manner.

The software metrics collected in the five NASA projects are mainly the McCabe and Halstead metrics. Many other software metrics such as process metrics and design metrics were proved important in software quality prediction [30], so we hope to improve the performance of our methodology if information on such metrics is available for a future study.

References

[1] http://mdp.ivv.nasa.gov

[2] http://www.r-project.org

[3] http://www.stat.berkeley.edu/users/breiman/RandomForests

[4] http://www.sas.com

[5] http://www.rulequest.com/see5-info.html

[6] D. Azar, S. Bouktif, B. Kégl, H. Sahraoui, D. Precup. Combining And Adapting Software Quality Predictive Models By Genetic Algorithms. *Proc. 17th IEEE International Conference on Automated Software Engineering (ASE2002)*, p285, 2002.

[7] V. R. Basili, L. C. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Software Eng.*, 22(10):751–761, Oct. 1996.

[8] L. Breiman. Random Forests. *Machine Learning*, Vol. 45, pp5–32, 2001.

[9] L. C. Briand, V. R. Basili, and C. J. Hetmanski. Developing Interpretable Models with Optimaized Set Reduction for Identifying High-Risk Software Components. *IEEE Trans. Software Eng.*, 19(11):1028–1044, Nov. 1993.

[10] C. Ebert. Classification techniques for metric-based software development. *Software Quality Journal*, 5(4):255–272, Dec. 1996.

[11] C. Ebert and E. Baisch. Industrial Application of Criticality Prediction in Software Development. *Proc. the Nineth International Symposium on Software Reliability Eng.*, pp80, Nov. 1998.

[12] N. Fenton and M. Neil. Software Metrics and Risk. *Proc. 2nd European Software Measurement Conference*, TI-KVIV, Amsterdam, Oct. 1999.

[13] S. S. Gokhale and M. R. Lyu. Regression tree modeling for the prediction of software quality. *Proc. the Third ISSAT International Conference on Reliability and Quality in Design*, pp31–36, Anaheim, CA, Mar. 1997.

[14] L. Guo, B. Cukic, and H. Singh. Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks. *Proc. 18th*

IEEE International Conference on Automated Software Engineering (ASE 2003), Oct. 2003.

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

[16] J. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, J. Maryland. Emerald: Software Metrics and Models on the Desktop. *IEEE Software*, pp. 56–60, Sep. 1996.

[17] T. M. Khoshgoftaar and N. Seliya. Tree-Based Software Quality Estimation Models For Fault Prediction. *Proc. the Eighth IIIE Symposium on Software Metrics (METRICS'02)*, pp203, Jun. 2002.

[18] T. M. Khoshgoftaar, and D. L. Lanning. A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software*, 29(1):85–91, Apr. 1995.

[19] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Early quality prediction: A case study in telecommunications. *IEEE Software*, 13(1):65–71, Jan. 1996.

[20] T. M. Khoshgoftaar, E. B. Allen, F. D. Ross, R. Munik oti, N. Goel, and A. Nandi. Predicting Fault-Prone Modules with Case-Based Reasoning. *Proc. the Eighth International Symposium on Software Engineering (ISSRE'97)*, pp27, Nov. 1997.

[21] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Internation Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[22] McCabe and Associates, "Software metrics: McCabe metrics". http://www.mccabe.com/metrics.php, 2004.

[23] T. McCabe. A Complexity Measure. *IEEE Trans. Software Eng.*, 2(4):308–320, Dec. 1976.

[24] T. J. McCabe and C. W. Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12):1415–1425, Dec. 1989.

[25] T. Menzies, J. D. Stefano, K. Ammar, R. M. Chapman, K. McGill, P. Callis, and J. Davis. When Can We Test Less? *IEEE Metrics'03*, 2003. http://menzies.us/pdf/03metrics.pdf

[26] T. Menzies and J. D. Stefano. How Good is Your Blind Spot Sampling Policy? *2004 IEEE Conference on High Assurance Software Engineering (HASE04)*, 2004.

[27] T. Menzies, J. D. Stefanom C. Cunanan, and R. M. Chapman. Mining Repositories to Assist in project Planning and Resource Allocation. *Submitted to the International Workshop on Mining Software Repositories*, 2004. http://menzies.us/pdf/04msrdefects.pdf

[28] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Trans. Software Eng.*, 18(5):423–433, May 1992.

[29] N. F. Schneidewind. Methodology For Validating Software Metrics. *IEEE Trans. Software Eng.*, 18(5):410–422, May 1992.

[30] R. W. Selby and A. A. Porter. Learning from examples: Generation and evaluation of decision trees for software resource analysis. *IEEE Trans. Software Eng.*, 14(12):1743–1756, Dec. 1988.

[31] J. Troster and J. Tian. Measurement and defect modeling for a legacy software system. *Annuals of Software Eng.*, 1:95–118, 1995.

[32] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, October 1999. http://www.cs.waikato.ac.nz/ml/weka/

[33] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java Implementations. http://www.cs.waikato.ac.nz/∼ihw/papers/99IHW-ETF-LT-MJ-GH-SJC-Weka.pdf, 1999.