Comparing the effectiveness of several modeling methods for fault prediction

Article *in* Empirical Software Engineering · June 2010

DOI: 10.1007/s10664-009-9111-2 · Source: DBLP

CITATIONS
94

READS
349

3 authors, including:

Elaine J. Weyuker
University of Central Florida
194 PUBLICATIONS **10,713** CITATIONS

SEE PROFILE

Thomas J. Ostrand
Mälardalen University
70 PUBLICATIONS **4,654** CITATIONS

SEE PROFILE

Empir Software Eng manuscript No.
(will be inserted by the editor)

Comparing the Effectiveness of Several Modeling Methods for Fault Prediction

Elaine J. Weyuker, Thomas J. Ostrand,
Robert M. Bell
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932

Received: date / Accepted: date

Abstract We compare the effectiveness of four modeling methods—negative binomial regression, recursive partitioning, random forests and Bayesian additive regression trees—for predicting the files likely to contain the most faults for 28 to 35 releases of three large industrial software systems. Predictor variables included lines of code, file age, faults in the previous release, changes in the previous two releases, and programming language. To compare the effectiveness of the different models, we use two metrics– the percent of faults contained in the top 20% of files identified by the model, and a new, more general metric, the *fault-percentile-average*. The negative binomial regression and random forests models performed significantly better than recursive partitioning and Bayesian additive regression trees, as assessed by either of the metrics. For each of the three systems, the negative binomial and random forests models identified 20% of the files in each release that contained an average of 76 to 94 percent of the faults.

Keywords empirical study · fault prediction · negative binomial · recursive partitioning · random forests · Bayesian trees · fault-percentile-average

1 Introduction

We have presented results of fault prediction models designed to facilitate efficient software testing for multiple large industrial systems [4,27]. The predictions, based on negative binomial regression models trained with readily available data from prior releases, were used to identify files deemed likely to have the most faults in the next release. The models have been quite effective—typically identifying the 20% of files in a release that contain upwards of 80% of faults.

We have often been asked whether alternatives to negative binomial regression (NBR) might improve predictive accuracy. NBR, a form of generalized linear model for count data [19], models the logarithm of the expected count as a linear function of the predictor variables. The thinking goes that NBR imposes restrictions of both linearity and additivity, and there is ample evidence in the world that complex systems do not always obey such restrictions.

System	No. of Releases	Years	KLOC	% Faults in 20% Files
Inventory	17	4	538	83%
Provisioning	9	2	438	83%
Voice Response	-	2+	329	75%

Table 1 Information for Empirical Studies

In this paper, we investigate that issue by exploring fault prediction using three additional methods that can produce very flexible sets of models with nonlinear and non-additive relationships: *recursive partitioning* (RP), *random forests* (RF), and *Bayesian additive regression trees* (BART). Recursive partitioning (also known as regression trees) explains variation in outcomes by building a tree based on successive binary splits, each on a single predictor variable [6]. Random Forests [5] and BART [7] are recent extensions of RP designed to overcome some of its perceived limitations.

We compare the predictive accuracy of RP, RF, and BART with that of NBR for three large subsystems of a very large and mature industrial system. Each of these subsystems runs as a stand-alone system and has been in the field for seven to nearly ten years, containing from 28 to 35 releases for each subsystem. Unlike our earlier papers, the focus of this paper is on the impact of the modeling methodology on the predictive accuracy obtained.

2 Our Previous Fault Prediction Work

Table 1 displays the size and age of three software systems studied previously. Each system had been in continuous use for between two and four years, and each contained hundreds of thousands of lines of code at the latest available release.

The negative binomial regression models used to predict fault likelihood for these systems were based on factors such as the file size, whether this is the first release in which a file appeared, the number of faults in earlier releases, the number of changes to the file in previous releases, and the programming language in which the file was written.

The last column of Table 1 provides an indication of the effectiveness of the models used for each study, averaged over all releases of a system. In particular, we have considered the percentage of actual faults that were contained in the 20% of the files identified by the models as likely to contain the largest numbers of faults. This average is computed by simply determining the percentage of faults identified in each release, adding these percentages and dividing by the number of releases. It is not weighted by the number of faults identified in a given release.

The table indicates that these models have been quite effective. In each case, the targeted files contained from 75% to 83% of the faults, averaged across releases. In all three systems, fault occurrence followed a Pareto-like distribution, with a relatively small percentage of files accounting for all the faults. For example, in releases 2-17 of the Inventory system, 100% of the faults were always contained in fewer than 20% of the files, and after release 10, all the faults were in fewer than 10% of the files. Details of these studies are available in [27] for the inventory and provisioning systems and in [4] for the voice response system.

There is surprising consistency among the results for these studies. Although the results for the voice response system are slightly less accurate than we observed for the

inventory and provisioning systems, they are still surprisingly good. The negative binomial regression model may have yielded less accurate results for this system because its development paradigm was different from the one for which the models were developed. The voice response system did not use a regular release schedule, which is fundamental to our prediction method, but instead used a "continuous release" approach. A fuller discussion of this issue is included in [4].

In [28], we studied whether a model with a pre-specified set of predictor variables could perform as well as a customized model designed explicitly for a given subject system. Because the pre-specified model slightly outperformed the customized one, we use it for assessing negative binomial regression in this paper. In addition, because it is pre-specified, this model more readily lends itself to automation which is necessary for building a tool, a prerequisite for industrial usability.

3 Other Prediction Work

Two main types of fault prediction goals have been studied in the literature. The first type aims at predicting whether or not a code entity such as a file or module will contain any faults in the next release. In this case every entity is categorized as either likely to be faulty or not, with no differentiation between how bad the faulty ones will be. Examples of research that fall into this category include work by Arisholm and Briand [2] and by Khoshgoftaar et al. [15,16].

The goal of the second type of research is to identify the entities most likely to contain the largest numbers of faults and sort them into decreasing order of numbers of predicted faults. Groups that have worked on this variant of prediction research include Denaro and Pezze [8] and Succi et al. [30] as well as our research described in [4,27,28].

In addition to fault prediction, there has been a substantial amount of preliminary work that attempted to identify the properties most closely associated with software entities containing faults or the ones with the highest numbers of faults or highest fault densities. Papers describing this sort of research include [1,3,9,10,12,13,21,22,25,29].

Of special note to this paper are studies that utilized machine learning methods for fault prediction. These include work by Koru and Liu [17] and by Menzies et al. [20], which used forms of recursive partitioning, and also work by Guo et al. [12], which fit random forests. The research described in each of these papers consider predictions made by a single type of model, as we did in earlier papers in which we made predictions using negative binomial regression models.

Jiang et al. [14] conducted fault prediction experiments using five different types of models, including random forests, logistic regression, bagging, boosting, and naive Bayes, but their emphasis was on comparing the effectiveness of different combinations of design and code metrics as predictor variables rather than on the differences between models. In their experiments on thirteen datasets from the NASA Metrics Data Program repository [23], random forests usually performed the best.

Lessmann et al. [18] compared predictive accuracy for 22 classifiers of fault proneness using ten data sets from the NASA Metrics Data repository. In general, they found few systematic differences among methods as measured by each method's ranks in terms of area under the ROC curve. In particular, they found no statistically significant difference among the top 17 methods.

4 Alternative Models

This section describes the four models under comparison in this paper. For each model, predictions for release N are based on training data from releases 1 to (N-1), with one observation per file for each release included in the training period. The dependent variable for each observation is the number of faults observed for the file during the particular release.

4.1 Negative Binomial Regression

All of our previous prediction modeling has used negative binomial regression (NBR), which models the logarithm of the expected number of faults as a linear combination of the predictor variables. On the positive side, the model's linearity assumption limits the number of parameters to be estimated, thereby reducing the chances of overfitting on training data and simplifying interpretation of the resulting model. At the same time, this assumption reduces the flexibility of NBR to fit more complicated relationships with nonlinear patterns or interactions (i.e., bivariate relationships that depend on the value of a third variable).

Specifically, let y_i equal the number of faults observed in file i and x_i be a vector of characteristics for that file. NBR specifies that y_i, given x_i, has a Poisson distribution with mean λ_i. To allow for additional dispersion compared with Poisson in the distribution of the number of faults for each file, the conditional mean of y_i is given by $\lambda_i = \gamma_i e^{\beta' x_i}$, where γ_i itself is a random variable drawn from a gamma distribution with mean 1 and unknown variance $\sigma^2 \geq 0$. The variance σ^2 is known as the *dispersion parameter*. The regression coefficients β and the dispersion parameter σ^2 were estimated by maximum likelihood [19], using the MASS package of the R library [32].

4.2 Recursive Partitioning

Recursive partitioning (RP) constructs a binary decision tree to partition training observations with the goal of producing leaf nodes that are each as homogeneous as possible with respect to the value of the dependent variable. The RP algorithm begins by splitting the entire set of observations into two nodes, based on a binary cut of a single predictor variable. The predictor variable and cut are chosen jointly to maximize the reduction in the total sum of squared errors for the two resulting nodes. Additional steps—each using that same criterion to split a single node based on cutting a single predictor variable—continue until a stopping criterion is satisfied. The prediction for any observation, in either the training data or a new data set, equals the mean of the dependent variable in the training data for the corresponding node [6].

In theory, recursive partitioning can produce very flexible models. Using successive splits on a single variable, the method can approximate arbitrary, nonlinear relationships. By default, RP tends to build models with high order interactions. If anything, the hardest models for RP to generate may be linear ones like those imposed by NBR.

We fit the RP models for this paper using the rpart package [34] in R. This implementation of recursive partitioning limits the growth of trees primarily through three parameters. The rpart model fitting function will not split any node with fewer than *minsplit* observations and will not perform a specific split that creates a node with

fewer than *minbucket* observations. We used the default values of *minsplit* = 20 and *minbucket* = 7. Finally, the *cp* parameter regulates tree growth by requiring that every split increase the R-square value of the resulting tree by at least the value of *cp*. Setting *cp* to smaller values increases the final tree size, and also increases the computation time to create the final tree. We evaluated *cp* values ranging from 0.01 down to 0.00001 (see Section 7.2). We did not do any pruning of the resulting trees.

4.3 Random Forests

The Random Forests (RF) methodology was motivated by a perceived drawback of recursive partitioning that makes it difficult to accurately model smooth or additive relationships. For example, if the dependent variable grows smoothly as a function of a particular predictor variable, modeling that relationship accurately requires multiple splits on the predictor (preferably all near the root of the tree). Similarly, modeling additive effects (i.e., effects that do not depend on the value of other predictors) may require many splits, even for a binary predictor, if the variable is not involved in one of the first few splits. In other words, predictions from recursive partitioning are highly dependent on the first few splits of the tree.

The method of random forests deals with this problem by basing predictions on the average of a large number of distinct trees [5]. First, each tree is based on perturbed data—a bootstrap sample of the original data selected with replacement—a technique called bagging. Second, to insure variety across trees in the early splits, only a sample of predictor variables is assessed, and therefore eligible for use, at each intermediate node. These two variations on standard recursive partitioning tend to produce splits that would not be found otherwise and to smooth out some of the rough edges associated with a single tree. In contrast to RP, where tree size is limited to avoid overfitting, each tree in a random forest is typically grown as large as possible, with the averaging of many trees being the protection against overfitting. We used the randomForest package in R [33] to fit random forests of size 500 trees. At each node, we sampled the default number of predictor variables, which was two. As the name implies, both bagging and sampling of predictor variables inject randomness into the final model and predictions.

4.4 Bayesian Additive Regression Trees

Bayesian additive regression trees (BART) try to address the limitations of RP by modeling the dependent variable as the *sum* of many trees plus a normally distributed error. The methodology, described in [7], specifies a full Bayesian model with prior distributions for the size of each tree, the distribution of the variables used for splitting, the split points, the magnitude of differences in means at each split, and the variance of errors. Estimation is performed using an iterative Bayesian backfitting MCMC algorithm. We used the BayesTrees package in R [31] to fit BART models to the data, and experimented with various values of the key parameters. Chipman et al. [7] found that BART, with certain default values, outperformed random forests, linear regression, neural nets, boosting, and MARS on simulated data proposed by [11], but our best results were obtained with slight variations of those defaults. We reduced the number of trees in the model to 100 from the default 200, but increased the probability of

generating larger individual trees by decreasing BART's *power* parameter to 0.95 from the default 2.0.

5 Subject Systems

We performed three empirical studies to compare the effectiveness of using recursive partitioning, random forests, Bayesian additive regression trees and negative binomial regression. Each study used a different stand-alone subsystem of a very large business maintenance system. We will refer to them as System A, System B, and System C. We studied 35 releases of Systems A and B, which represented close to ten years in the field for each of the systems. System C was begun during the eighth release of the other systems; our study of System C covers 28 releases over roughly seven years. Each of the systems was large—containing a maximum of 668, 1413, and 584 files, respectively, and 442 KLOCs, 384 KLOCs, and 329 KLOCs during the final release studied.

The top portions of Figures 1, 2, and 3 show the numbers of faults that were actually detected in each release of Systems A, B, and C, respectively. Generally, these faults were found after a failure was observed during system test. Very few faults were identified once the system was released to the field for customer usage. For each system, the number of faults varied widely across releases—from 5 or fewer to about 120 or 130. The average number of faults per release was 48, 39, and 53 for Systems A, B, and C, respectively.

6 Evaluation Metrics

In past work, we have often assessed predictive accuracy in terms of the percentage of faults contained in the 20% of files predicted to have the most faults (see Table 1). The figure of 20% was obtained from our initial empirical studies of large systems [25], where we repeatedly found 20% of the files contained 80% or more of the reported faults. While this intuitive measure of success works well for many purposes, the relative performance of competing prediction methods can be sensitive to the arbitrary cutoff value of 20%. To provide the most stable comparisons possible in the next section, we propose a more general metric that reflects the effectiveness of the different prediction models across all values of the cutoff.

This metric is the average, over all values of m, of the percentage of faults contained in the top m files, sorted in decreasing order of predicted numbers of faults. If the predictions are good, then files with many faults will occur at or near the top of the list, and their fault counts will be included in most of the terms that contribute to the average. As a result, the overall average percentage of faults will be high. However, if the predictions are poor, and fault-laden files occur farther down in the list, then their fault counts will not contribute to the average until later in the computation, resulting in a lower value of the metric.

The following derivation shows that the metric is equivalent to the average percentile of predicted faults associated with faulty files, where the average is weighted by the actual number of faults in those files. Consequently, we call this metric the *fault-percentile-average*.

Consider a release with K files f_1, f_2,...,f_K, listed in increasing order of predicted numbers of faults, so f_K is the file predicted to have the most faults. Let n_k be the

actual number of faults in file k, and let $N = n_1 + ... + n_K$ be the total number of faults in the release. For any m in $1, ..., K$, the proportion of actual faults in the top m predicted files is

$$P_m = \frac{1}{N} \sum_{k=K-m+1}^{K} n_k$$

The desired metric is then the average of the P_m:

$$\frac{1}{K} \sum_{m=1}^{K} \frac{1}{N} \sum_{k=K-m+1}^{K} n_k$$

Pulling out the constant $\frac{1}{N}$ and interchanging the order of summation yields

$$\frac{1}{KN} \sum_{k=1}^{K} \sum_{m=K-k+1}^{K} n_k$$

Letting $j = m - K + k$,

$$= \frac{1}{KN} \sum_{k=1}^{K} n_k \sum_{j=1}^{k} 1$$

$$= \frac{1}{KN} \sum_{k=1}^{K} k(n_k)$$

There may be ties among the predicted fault counts, in which case the ordering of the actual fault counts can affect the value of P_m. In this case, we distribute the actual fault count equally among the tied files. For example, if 5 files are predicted to have the same number of faults, and together they contain a total of 3 actual faults, we assign $n_k = .6$ to each of them.

Note that each term in the last summation is the rank of the k^{th} file, weighted by the number of its faults. The summation is the sum of the weighted ranks, and $1/N$ times the sum is the average rank per fault. The entire expression can be interpreted as the average percentile of the faults for each file. Multiplying by $1/K$ normalizes the result to allow uniform comparison across systems with different numbers of files.

The fault-percentile-average can also be seen as a normalized version of the area under the predictor curve in the Alberg diagram defined by Ohlsson and Alberg [24]. For the predictor curve, the files are shown in decreasing order of predicted fault count on the x-axis, and the y-axis shows the cumulative number of observed faults.

Figures 1, 2, and 3 show the value of the fault-percentile-average for each release in Systems A, B, and C. The predictions start after data from two releases with faults have been accumulated (at Release 3 for Systems A and B, and at Release 5 for System C). The gap at Release 33 for System B reflects the absence of any faults for that release. Figure 5 summarizes the fault-percentile-average across all the releases of each system (weighted by the numbers of faults at each release). Figure 6 is a similar chart of the original 20% metric averaged over all the releases of each system.

Statements about the statistical significance of differences between modeling methods are based on t-tests for whether mean differences in percentiles, weighted by number of faults, differ from zero. For example, to compare NBR with RP, we subract the NBR

percentile from the RP percentile for each file with one or more faults (across all releases for a system). If the resulting t-statistic differs significantly from zero, we report that the methods differ significantly. White's method [35] is used to account for the weights in computing t-statistics.

Various other metrics have been used by research groups as ways of assessing the effectiveness of fault prediction approaches. Among these metrics are *Accuracy*, *Precision*, and *Recall*, as well as the percentages of *Type I* (False Positives) and *Type II* (False Negatives) misclassification rates.

In [26], Ostrand and Weyuker discuss the pros and cons of each of these metrics, and conclude that Accuracy, Precision and Recall all have significant deficiencies as a primary assessment metric. They further argued that while it is desirable to have few false positives (non-faulty files incorrectly identified as being faulty) because it leads to an inefficient use of resources, it is the false negative rate that is particularly important. Their argument was that when a file is incorrectly determined to be non-faulty, the file will receive less scrutiny than it should and therefore faults are likely to go undetected. Khoshgoftaar et al. [16] and Jiang et al. [14] offer a similar argument.

We believe that the combination of the two metrics used in this paper provides a good way of assessing fault prediction methods, and balances the various issues discussed in [26].

7 Prediction Results for the Four Modeling Methods

This section reports results of fitting four different prediction models. Except as noted below, we made the same predictor variables available to each model: lines of code, age of the file (i.e., number of previous releases), recent history of changes and faults (set to zero for new files), programming language, and release number.

7.1 Negative Binomial Regression

Table 2 shows results of the negative binomial regression model fit to 13,366 observations from releases 1 to 26 of System A (sample chosen arbitrarily). Several predictor variables were transformed (logarithm or square root) to reduce skewness and to improve the fit (this was not done for the other three models, which are invariant to transformation of predictor variables). In addition, based on experience with previous systems, we categorized file age into four groups: new, age 1, age 2-4, and age 5 or more. All predictors other than dummy variables for languages and releases (not shown) were very statistically significant. Greater numbers of faults are associated with larger files, newer files, and files with recent changes or faults. These results generally agree with those observed for systems studied previously [4, 27].

7.2 Recursive Partitioning

Figure 4 displays the tree fit by recursive partitioning to the same data for releases 1 to 26 of System A, using a value of $cp = 0.01$. The first split was based on the number of faults in the prior release. Files with three or fewer faults in the prior release (13,305 of 13,366 observations in the training data) fell to the left, while files with four or

Variable	Coefficient	Std. Error	z-statistic
Log(KLOC)	0.48	0.04	11.14
New file	2.40	0.22	10.98
Age = 1	0.87	0.21	4.12
Age = 2-4	0.33	0.15	2.21
(Prior Changes)$^{1/2}$	0.53	0.06	9.41
(Prior Prior Changes)$^{1/2}$	0.29	0.05	5.78
(Prior Faults)$^{1/2}$	0.45	0.09	5.06
Language 1	0.66	0.25	2.69
Language 2	0.36	0.26	1.38
Language 3	0.01	0.24	0.03
Language 4	-0.05	0.28	-0.17
Language 5	-0.81	0.25	-3.31
Language 6	$-\infty$	NA	$-\infty$

Table 2 Coefficients for NBR Model Fit to Releases 1 to 26 of System A

more faults in the prior release (61 observations) fell to the right. Subsequent splits were based on lines of code, prior faults (again), prior changes (twice), programming language (twice), and file age. For numeric splitting variables, the split was based on an ordinal cut. In contrast, for the nominal variable *programming language*, the split involved dichotomizing the list of categories. For example, the notation "prog_lang=bgj" at the furthest right split indicates that files programmed in the second, seventh, and tenth languages (ordered alphabetically) fell to the left with all others falling right.

In total, eight splits were made, resulting in nine terminal nodes. The number below each node equals the mean number of faults in the training set for that node. Those numbers are used for subsequent predictions.

Note that in the tree of Figure 4, even with eight splits, the far left node (mean = 0.04687) contains 95.0% of the training data, leading to inadequate discrimination among the vast majority of files. This occurred because no further split of the left node was able to improve the overall R-square by the *cp* criterion of 0.01 or more. In general, larger values of *cp* tend to leave too many files in a single node, leading to inadequate discrimination around the cut point demarcating the top 20% of files (we use linear interpolation to account for the fact that RP does not break ties within nodes for determining the top 20% of files).

To make a fair comparison of recursive partitioning versus the other methods, we needed to determine an optimal value of the *cp* parameter. We tested values of $cp = 0.01$, 0.005, 0.001, 0.0005, 0.0001, and 0.00001 for all three systems. Decreasing *cp* allows more node splitting, and permits the tree to continue growing, which improved our measure of predictive accuracy, up to a point. On the example data, using successively smaller values of $cp = 0.005$, 0.001, 0.0005, and 0.0001 produced trees with 15, 45, 74, and 171 terminal nodes. Of the 73 splits in the tree produced by $cp = 0.0005$, 27 occurred on LOC, with no other predictor used more than 10 times. Although prior faults had the fewest splits (3), we note that two of those splits occurred at or near the root, making this variable particularly influential on predictions.

For two of the three systems, $cp = 0.0005$ identified the greatest percentage of faults in the top 20% of files, averaged across releases, and it was second to $cp = 0.001$ for the third system. Consequently, all results for RP are based on using $cp = 0.0005$.

7.3 Random Forests

For RF we considered forests of various sizes including a single tree, 20 trees, 100 trees and 500 trees. On average, we obtained the best results when using 500 trees. The results included here and below reflect this version of the RF model. (Jiang et al. [14] also obtained the best random forests results with 500 trees.)

A drawback of random forests is the difficulty of displaying the results and the consequent challenge of interpreting the model. One tool to help measure which predictor variables matter most is an "importance" measure, which indicates how much individual predictors contribute to the reduction in the mean squared prediction error. This measure is returned by R's random forests predictor function, along with the predicted fault values for each release. For releases 1 to 26 of System A, the highest importance values occurred for prior faults and LOC.

7.4 Bayesian Trees

For BART we considered models that were sums of 40 trees, 100 trees, and 200 trees. We generally observed the best results with 100 trees. Results with 200 trees were virtually the same or even slightly poorer than for 100, and the computation times were significantly longer. All reported results are based on models with 100 trees.

In contrast to the other methods, the BART models did not include programming language as a predictor variable because the available software required all predictors to be numeric. For 100 trees fit to releases 1 to 26 of System A, on average (across MCMC samples) there were 530 total splits on six predictor variables. The most splits occurred for LOC (111) and prior faults (101).

7.5 Comparison of Predictive Accuracy

The lower portion of Figure 1 compares predictive accuracy of NBR, RF, RP and BART for System A. For each release starting with the third, the figure displays, by modeling method, the fault-percentile-average (i.e., the mean percentile of predicted number of faults for files that had faults, weighted by number of faults). With occasional exceptions, results were uniformly best for RF and NBR—with the mean percentile reaching 80 by release 5 and often exceeding 90 after release 16. Performance was notably worse for releases 3 and 4, for which there were fewer than 75 faults on which to train.

Results for both RP and BART were much more erratic and often substantially inferior to those for RF and NBR. Some, but not all, of the worst results for BART occurred for releases with fewer than 10 faults.

Figures 2 and 3 illustrate similar patterns for Systems B and C. Again, results are almost uniformly as good or better for both NBR and RF than for either RP and BART. RP appears to generally outperform BART for these two systems.

In contrast with System A, NBR and RF perform very well for System B starting with release 3, perhaps because there were already more than 100 faults for training release 3. For System C, none of the methods reached 80% until release 10, again likely due to the small numbers of faults in the releases preceding 10.

Figure 5 summarizes results for the fault-percentile-average metric. The mean percentiles across releases (weighted by number of faults) for NBR and RF were almost identical for all three systems. The differences were not statistically significant for any of the three systems. Performance of RP and BART was significantly worse than either NBR or RF for all three systems ($p < 0.001$), and BART was significantly worse than RP for Systems B ($p < 0.001$) and C ($p < 0.01$). Figure 6 shows corresponding results for the original metric of percentage of faults in the top 20% of files based on predicted numbers of faults. Reported values are unweighted averages over all releases of a system.

	% Correlation of Percentiles		
Methods	**System A**	**System B**	**System C**
NBR - RF	.78	.88	.78
NBR - RP	.59	.73	.68
RP - RF	.61	.70	.64
RF - BART	.60	.61	.61
NBR - BART	.52	.61	.51
RP - BART	.45	.52	.43

Table 3 Correlations of Percentiles for Pairs of Methods, by System

Table 3 shows correlations of percentiles for all six pairs of methods, by system. For a single release, this calculation would be equivalent to computing Spearman correlations of raw predictions. The actual calculation allows aggregating information across all releases of a given system in order to produce more stable correlations.

For each system, the correlation between NBR and RF is the highest of all the pairs by a wide margin, ranging from 0.78 for two systems and 0.88 for the other. This suggests that RF is approximately replicating NBR, or vice versa. In contrast, RP and BART rank the files substantially differently from NBR and RF, and even more so from each other; the correlation between RP and BART is the lowest for all six pairs.

8 Conclusions

For each of the three large industrial systems we studied in this paper, NBR and RF substantially and significantly outperformed RP and BART. There was not a statistically significant difference between NBR and RF for any of the systems. This suggests that the linear and additive assumptions behind negative binomial regression are probably realistic for these systems. Given much greater execution times for fitting RF and the non deterministic results, NBR offers distinct advantages at little cost for these systems. Of course, there is no guarantee that comparisons for other systems with different characteristics would come out the same.

We believe that RP performed poorly because of the difficulties we discussed above (Section 4.3) in fitting what appears to be basically a linear and additive relationship on a logarithmic scale. For a condition (e.g., new files) that is associated with more faults uniformly across values of other predictors, the best RP model would need to incorporate that condition in every path of the tree. Unless the condition induces a split very near the top of the tree, which can only occur for a few predictors, this would require a large number of splits lower down. Even with thousands of observations for

most training sets, there seems to be insufficient information to determine the correct structure. A contributing factor may have been insufficient tuning of the model fitting procedure. However, we doubt that the shortfall associated with RP could be overcome by tweaking either the model (e.g., alternatives like C4.5) or specific parameters. Instead, to the extent that relationships are stable across the range of other predictors, recursive partitioning appears to be an inefficient method for estimating those relationships.

We hypothesize that the very poor performance of BART results from the failure of two assumptions. Most obviously, the distribution of faults is discrete and very skewed with the vast majority of observations being zero, far from BART's assumption of normal errors with constant variance. However, a larger problem may be BART's additive structure on the raw scale. If, as hypothesized by NBR, relationships are additive on a logarithmic scale, that could produce large interactions on the raw scale that BART is poorly suited to handle. Chipman et al. [7] outline an extension of BART for binary classification, which might work much better for data like ours, even though our outcome is not naturally binary. However, we do not have access to software for this extension.

At first glance, our findings may seem at odds with those of Lessmann et al. [18], who found relatively few statistically significant differences in predictive accuracy among 22 classifiers of fault proneness. However, random forests and CART, possibly the classifier closest to RP, were ranked first and last among the 22 classifiers in [18], a difference that was statistically significant. Furthermore, Lessmann et al. did not evaluate BART or anything similar, and we believe that its poor performance for our problem was explained by the circumstances detailed above. Consequently, there is little evidence of disagreement between the two studies.

References

1. E.N. Adams. Optimizing Preventive Service of Software Products. *IBM J. Res. Develop.*, Vol 28, No 1, Jan 1984, pp. 2-14.
2. E. Arisholm and L.C. Briand. Predicting Fault-prone Components in a Java Legacy System. *Proc. ACM/IEEE ISESE*, Rio de Janeiro, 2006.
3. V.R. Basili and B.T. Perricone. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, Vol 27, No 1, Jan 1984, pp. 42-52.
4. R.M. Bell, T.J. Ostrand, and E.J. Weyuker. Looking for Bugs in All the Right Places. *Proc. ACM/International Symposium on Software Testing and Analysis (ISSTA2006)*, Portland, Maine, July 2006, pp. 61-71.
5. L. Breiman. Random Forests. *Machine Learning*, Vol. 45, 2001, pp. 5-32.
6. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.
7. H.A. Chipman, E.I. George, and R.E. McCulloch. BART: Bayesian Additive Regression Trees. http://arxiv.org/abs/0806.3286v1, June 2008.
8. G. Denaro and M. Pezze. An Empirical Evaluation of Fault-Proneness Models. *Proc. International Conf. on Software Engineering (ICSE2002)*, Miami, USA, May 2002.
9. S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, A. Mockus. Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Trans. on Software Engineering*, Vol 27, No. 1, Jan 2001, pp. 1-12.
10. N.E. Fenton and N. Ohlsson. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Trans. on Software Engineering*, Vol 26, No 8, Aug 2000, pp. 797-814.
11. J.H. Friedman. Multivariate Adaptive Regression Splines. *Annals of Statistics*, Vol. 19, 1991, pp. 1-67.

12. L. Guo, Y. Ma, B. Cukic, H. Singh. Robust Prediction of Fault-Proneness by Random Forests. *Proc. ISSRE 2004*, Saint-Malo, France, Nov. 2004.
13. L. Hatton. Reexamining the Fault Density - Component Size Connection. *IEEE Software*, March/April 1997, pp. 89-97.
14. Yue Jiang, Bojan Cukic, Yan Ma. Techniques for evaluating fault prediction models. *Empir Software Eng* (2008), Vol 13, pp. 561-595.
15. T.M. Khoshgoftaar, E.B. Allen, J. Deng. Using Regression Trees to Classify Fault-Prone Software Modules. *IEEE Trans. on Reliability*, Vol 51, No. 4, Dec 2002, pp. 455-462.
16. T.M. Khoshgoftaar, E.B. Allen, K.S. Kalaichelvan, N. Goel. Early Quality Prediction: A Case Study in Telecommunications. *IEEE Software*, Jan 1996, pp. 65-71.
17. A.G. Koru and H. Liu. An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures. *2005 Promise Workshop*, May 15, 2005.
18. S. Lessmann, B. Baesens, C. Ues, S. Pietsch. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, Vol. 34, No. 4, July/August 2008, pp. 485-496.
19. P. McCullagh and J.A. Nelder. *Generalized Linear Models*, Second Edition, Chapman and Hall, London, 1989.
20. T. Menzies, J.S. Di Stefano, C. Cunanan, and R. Chapman. Mining Repositories to Assist in Project Planning and Resource Allocation. *International Workshop on Mining Software Repositories*, May 2004.
21. K-H. Moeller and D.J. Paulish. An Empirical Investigation of Software Fault Distribution. *Proc. IEEE First International Software Metrics Symposium*, Baltimore, Md., May 21-22, 1993, pp. 82-90.
22. J.C. Munson and T.M. Khoshgoftaar. The Detection of Fault-Prone Programs. *IEEE Trans. on Software Engineering*, Vol 18, No 5, May 1992, pp. 423-433.
23. Metrics Data Program. NASA IV&V facility, ed. Jay Long. http://MDP.ivv.nasa.gov.
24. N. Ohlsson and H. Alberg. Predicting Fault-Prone Software Modules in Telephone Switches. *IEEE Trans. on Software Engineering*, Vol 22, No 12, December 1996, pp. 886-894.
25. T. Ostrand and E.J. Weyuker. The Distribution of Faults in a Large Industrial Software System. *Proc. ACM/International Symposium on Software Testing and Analysis (ISSTA2002)*, Rome, Italy, July 2002, pp. 55-64.
26. T. Ostrand and E.J. Weyuker. How to Measure Success of Software Prediction Models. *Proc. Fourth International Workshop on Software Quality Assurance*, Dubrovnik, Croatia, Sept 2007.
27. T.J. Ostrand, E.J. Weyuker, and R.M. Bell. Predicting the Location and Number of Faults in Large Software Systems. *IEEE Trans. on Software Engineering*, Vol 31, No 4, April 2005.
28. T.J. Ostrand, E.J. Weyuker, and R.M. Bell. Automating Algorithms for the Identification of Fault-Prone Files. *Proc. ACM/International Symposium on Software Testing and Analysis (ISSTA07)*, London, England, July 2007.
29. M. Pighin and A. Marzona. An Empirical Analysis of Fault Persistence Through Software Releases. *Proc. IEEE/ACM ISESE 2003*, pp. 206-212.
30. G. Succi, W. Pedrycz, M. Stefanovic, and J. Miller. Practical Assessment of the Models for Identification of Defect-prone Classes in Object-oriented Commercial Systems Using Design Metrics. *Journal of Systems and Software*, Vol 65, No 1, Jan 2003, pp. 1 - 12.
31. The BayesTree Package. http://cran.r-project.org/web/packages/BayesTree
32. The R Project for Statistical Computing. http://www.r-project.org/
33. The randomForest Package. http://cran.r-project.org/web/packages/randomForest
34. The rpart Package. http://cran.r-project.org/web/packages/rpart
35. H. White. A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity. *Econometrics*, Vol. 48, pp. 817-838.

Fig. 1 Number of Faults and Average Percentile of Faults for System A, by Release Number

Fig. 2 Number of Faults and Average Percentile of Faults for System B, by Release Number

Fig. 3 Number of Faults and Average Percentile of Faults for System C, by Release Number

Fig. 4 Sample Recursive Partition for Releases 1-26 for System A, using $cp = 0.01$

Fig. 5 Average Percentile of Faults for Each System, by Prediction Method

Fig. 6 Average Percentage of Faults in Top 20% of Files Each System, by Prediction Method