

Comparing Negative Binomial and Recursive Partitioning Models for Fault Prediction

Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932
(weyuker,oststrand,rbell)@research.att.com

ABSTRACT

Two different software fault prediction models have been used to predict the $N\%$ of the files of a large software system that are likely to contain the largest numbers of faults. We used the same predictor variables in a negative binomial regression model and a recursive partitioning model, and compared their effectiveness on three large industrial software systems. The negative binomial model identified files that contain 76 to 93 percent of the faults, and recursive partitioning identified files that contain 68 to 85 percent.

Categories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging – *Diagnostics*

General Terms: Experimentation

Keywords: empirical study, software testing, fault prediction, negative binomial, recursive partition

1. INTRODUCTION

We have presented results of fault prediction models designed to facilitate efficient software testing for multiple large industrial systems [4, 19]. The predictions, based on negative binomial regression models trained with readily available data from prior releases, are used to identify files deemed likely to have the most faults in the next release. The models have been quite effective—typically identifying 20% of files 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 [14], 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 simple models.

In this paper, we begin to investigate that issue by exploring fault prediction using the method of *recursive partitioning* (also known as regression trees), which explains

variation in outcomes by building a tree based on successive binary splits, each on a single predictor variable. Recursive partitioning (RP) allows a very flexible set of models with nonlinear and nonadditive relationships. We compare prediction accuracy of RP with that of NBR for three large subsystems of a new industrial system. Although these three software subsystems are different from those presented in our earlier work cited above, the focus of this paper is on the impact of the modeling methodology.

2. OUR PREVIOUS FAULT PREDICTION WORK

Table 1 displays the size and age of the 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 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. 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 [19] 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PROMISE'08, May 12–13, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-036-4/08/05 ...\$5.00.

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

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 [20], 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 prespecified model slightly outperformed the customized one, we use it for assessing negative binomial regression in this paper.

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 [2, 11, 12].

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 [6, 22] in addition to our research described in [4, 19, 20].

In addition to fault prediction, there has been a substantial amount of preliminary work that attempted to identify the properties of entities most closely associated with 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, 7, 8, 9, 10, 16, 17, 18, 21].

Of special note to this paper are studies that utilized machine learning methods for fault prediction. These include [13] and [15], which used forms of recursive partitioning, and [9], which fit random forest models.

4. ALTERNATIVE MODELS

This section describes the two 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. 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 is itself 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 [14], using the MASS package of the R library [23].

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 to maximize the reduction in the total sum of squared errors for the two resulting nodes. Additional steps—each using the 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.

In theory, recursive partitioning can produce very flexible models. Using successive splits on a single variable, the method can build very 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 [24] 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`. In addition, the model will not perform a split on a node if the R-square value of the tree that results from the split is not reduced by at least the value of the `cp` parameter. 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 below). We did not do any pruning of the resulting trees.

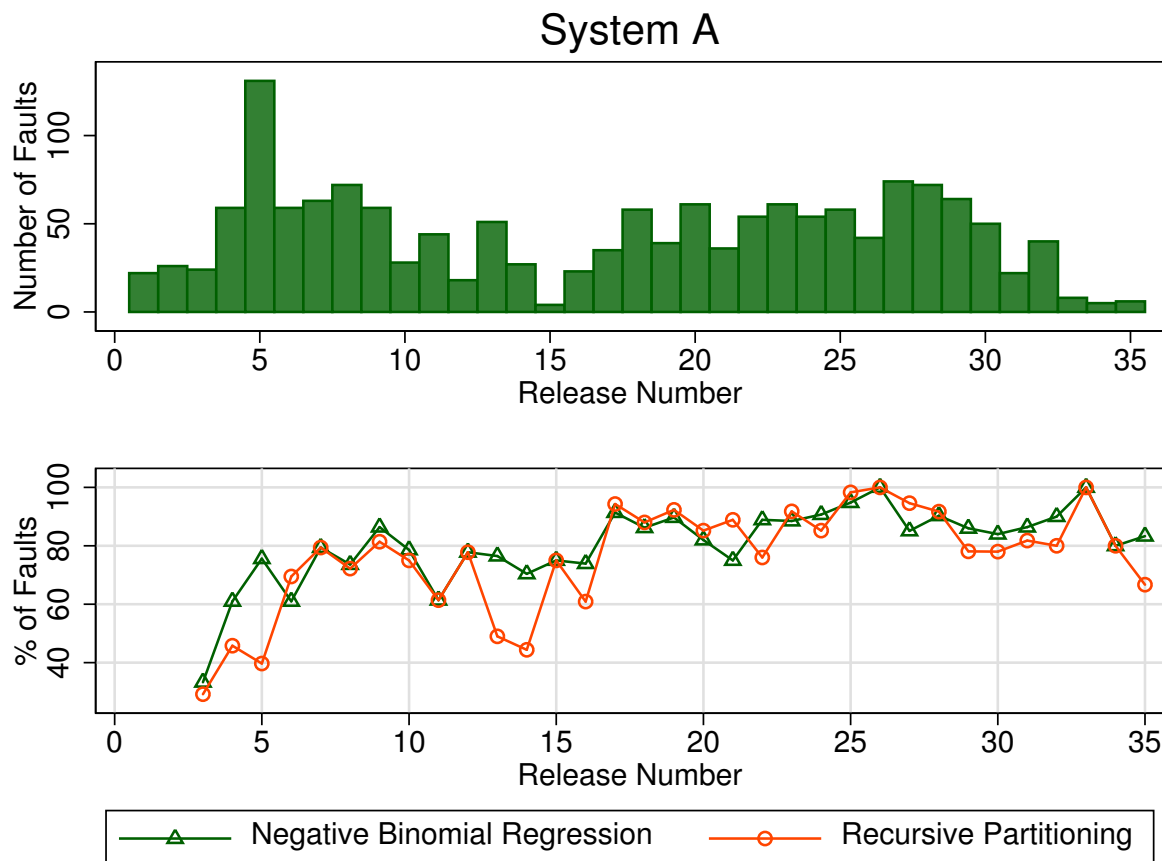


Figure 1: Number of Faults and Percentage of Faults in Top 20% of Files, by Release Number for System A

5. SUBJECT SYSTEMS

We performed three empirical studies to compare the effectiveness of using recursive partitioning versus the negative binomial regression model. 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 approximately nine years in the field. System C was begun during the 10th release of the other systems; our study of System C covers 26 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 portion of Figures 1, 2, or 3 shows the number 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. RESULTS

6.1 Examples of Each Type of Model

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. All predictors other than dummy variables for languages and releases (not shown) were very statistically significant. The greatest number of faults is 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, 19].

Figure 4 displays the tree fit to 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 (13,305 of 13,366 observations in the training data) fell to the left, while files with four or 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

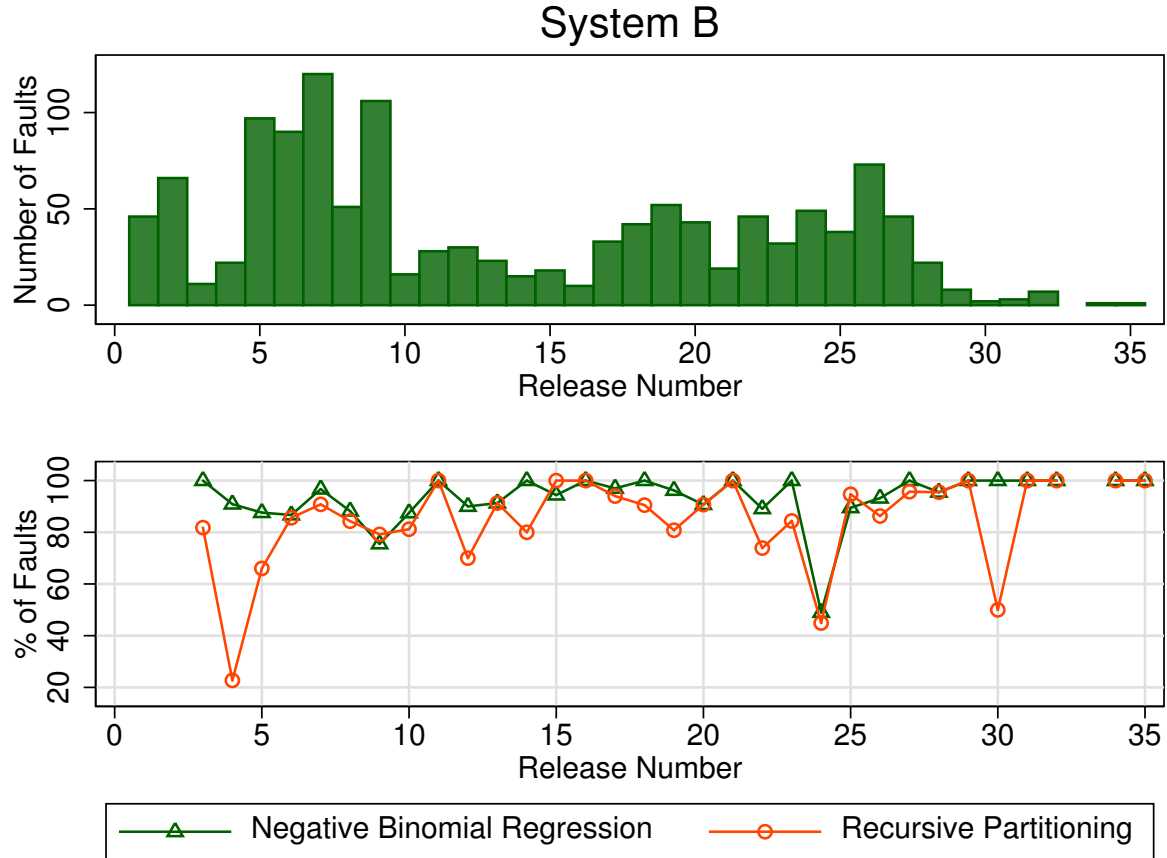


Figure 2: Number of Faults and Percentage of Faults in Top 20% of Files, by Release Number for System B

that files programmed in the second, seventh, and tenth languages (ordered alphabetically) fell to the left with all others falling right.

A total of 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.

Even with eight splits, the far left node (mean = 0.04687) contained 95.0% of the training data because no split was able to improve the overall R-square by the cp criterion of 0.01 or more. Decreasing the cp parameter allowed the tree to continue growing. Using successively smaller values of $cp = 0.005, 0.001, 0.0005,$ and 0.0001 produced trees with 15, 45, 74, and 171 nodes.

6.2 Comparison of Predictive Accuracy

To make a fair comparison of predictive accuracy for negative binomial regression versus resursive partitioning, we needed to investigate how large to set cp for the latter method. We tested value of $cp = 0.01, 0.005, 0.001, 0.0005, 0.0001,$ and 0.00001 for all three systems.

In general, decreasing cp in order to build larger trees improved our measure of predictive accuracy, up to a point. As illustrated above, using $cp = 0.01$ tended 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). For two of three systems, $cp = 0.0005$ identified the greatest percentage of faults, averaged across releases, and it was second to $cp = 0.001$ for the third system. Performance degraded for $cp = 0.0001$ and below suggesting substantial overfitting.

The lower portions of Figures 1 to 3 compare predictive accuracy of NBR and RP for the three systems. The figures display for each release, starting with the third, the percentages of faults included in the 20% of the files predicted to contain the largest numbers of faults, using each prediction method. For RP, all results are based on using $cp = 0.005$.

The figures show that NBR generally outperformed RP on this measure for all three systems. Although results were certainly not uniform across releases for any of the systems, almost all substantial differences favored NBR.

Table 3 summarizes our findings in terms of unweighted averages over all releases for the two different prediction models. These averages favor NBR by between 4.4 and 8.6 percentage points. However, we note that much of these differences are due to very large shortfalls for RP on a small number of releases that often have very few faults.

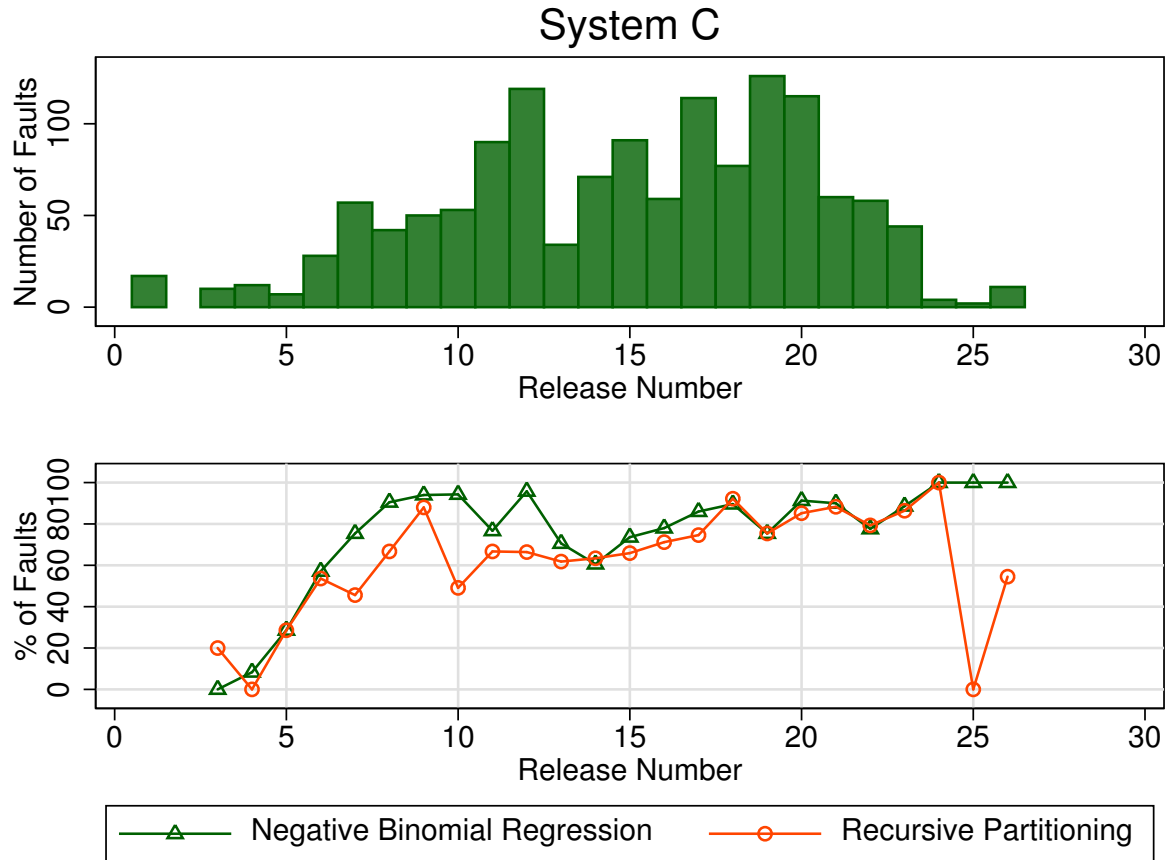


Figure 3: Number of Faults and Percentage of Faults in Top 20% of Files, by Release Number for System C

7. CONCLUSIONS

For each of the three large industrial systems we studied in this paper, the predictive accuracy of recursive partitioning was noticeably inferior, on average, to that of negative binomial regression. 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 is an inefficient method for estimating those relationships.

A more promising alternative may be random forests, which averages across an ensemble of trees (RP models) fit to perturbations of the original data [5]. This methodology may work better for fitting linear and additive relationships, while maintaining the flexibility of recursive partitioning, where needed. We plan to evaluate random forests and see how they compare to the results obtained by using negative binomial regression and recursive partitioning models.

8. 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] G. Denaro and M. Pezze. An Empirical Evaluation of Fault-Proneness Models. *Proc. International Conf on Software Engineering (ICSE2002)*, Miami, USA, May 2002.
- [7] 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.

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	-∞	NA	-∞

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

System	% Faults In 20%	
	Neg Bin Reg	Rec Part
System A	80.5	76.1
System B	93.4	84.8
System C	76.1	67.9

Table 3: Average Percentage of Faults in Top 20% of Files Identified by NBR and by RP with $cp = 0.0005$

- [8] 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.
- [9] L. Guo, Y. Ma, B. Cukic, H. Singh. Robust Prediction of Fault-Proneness by Random Forests. *Proc. ISSRE 2004*, Saint-Malo, France, Nov. 2004.
- [10] L. Hatton. Reexamining the Fault Density - Component Size Connection. *IEEE Software*, March/April 1997, pp. 89-97.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] P. McCullagh and J.A. Nelder. *Generalized Linear Models*, Second Edition, Chapman and Hall, London, 1989.
- [15] T. Menzies, J.S. Di Stefano, C. Cunanen, and R. Chapman. Mining Repositories to Assist in Project Planning and Resource Allocation. *Innternational Workshop on Mining Software Repositories*, May 2004.
- [16] K-H. Moller 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] M. Pighin and A. Marzona. An Empirical Analysis of Fault Persistence Through Software Releases. *Proc. IEEE/ACM ISESE 2003*, pp. 206-212.
- [22] 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.
- [23] The R Project for Statistical Computing. <http://www.r-project.org/>
- [24] The rpart Package. <http://cran.r-project.org/doc/packages/rpart.pdf>

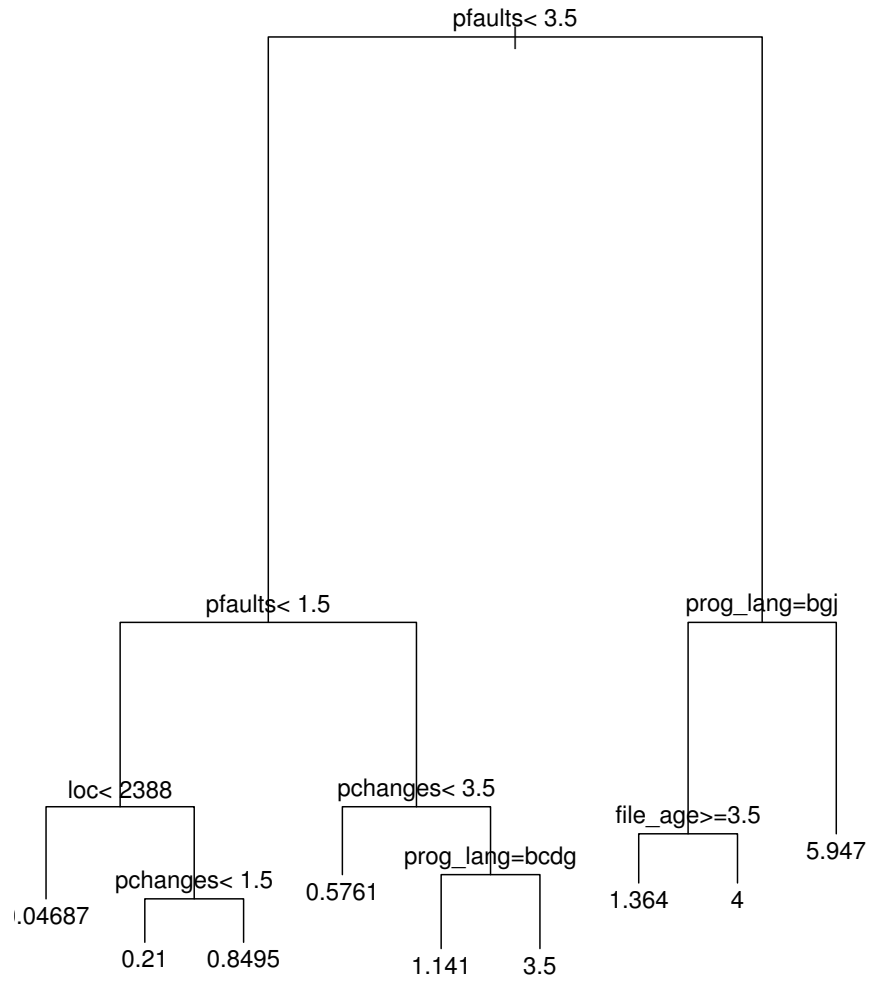


Figure 4: Sample Recursive Partition for Releases 1-26 for System A, using $cp = 0.01$