Tree-Based Software Quality Estimation Models For Fault Prediction

Taghi M. Khoshgoftaar*
Naeem Seliya
Florida Atlantic University
Boca Raton, Florida USA

Abstract

Complex high-assurance software systems depend highly on reliability of their underlying software applications. Early identification of high-risk modules can assist in directing quality enhancement efforts to modules that are likely to have a high number of faults. Regression tree models are simple and effective as software quality prediction models, and timely predictions from such models can be used to achieve high software reliability.

This paper presents a case study from our comprehensive evaluation (with several large case studies) of currently available regression tree algorithms for software fault prediction. These are, CART-LS *(least squares),*S-PLUS*, and* CART-LAD *(least absolute deviation). The case study presented comprises of software design metrics collected from a large network telecommunications system consisting of almost* 13 *million lines of code. Tree models using design metrics are built to predict number of faults in modules. Besides being compared for fault prediction accuracy, algorithms are also compared based on structure and complexity of their tree models. Performance metrics, average absolute and average relative errors are used to evaluate fault prediction accuracy. Number of terminal nodes and number of independent variables utilized by a tree model are used to gauge complexity and structure of regression trees. It was observed that* S-PLUS *trees had poor predictive accuracy and were large and more complex. On the other hand,* CART-LAD *trees had good accuracy and were simple and easy to interpret.*

Keywords*: software fault prediction, regression trees,* CART, S-PLUS*, least squares, least absolute deviation*

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory , Deptartmen t of Computer Science and Engineering, Florida Atlantic Univ ersity, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/.

1. Introduction

Complex high-assurance and mission-critical software systems are heavily dependent on stability and reliability of their underlying software applications. Human safety and reduction in software development costs are the primary reasons why computer scientists and researchers are actively engaged in developing, quantifying and implementing techniques for improving software reliability. Reliability-improvement techniques include rigorous design and code reviews, automatic test-case generations, strategic assignment of key personnel, and re-engineering of high-risk areas of a system.

Early fault prediction is a proven technique in achieving high software reliability and can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults [15]. A software fault is a defect that causes a software failure in an executable product [21].Software quality models based on software metrics have been shown [13] to yield quality predictions with useful accuracy [14]. Such models can be used to predict a quality factor such as number of faults for a module or number of code churns. Software metrics such as design metrics can be collected earlier in the development life cycle than the quality factor [35]. If such software metrics for a module are supplied, one can compute a prediction of the number of faults.

Many software quality modeling techniques have been developed and used in real life software quality estimation problems. A few commonly used techniques include regression trees [7, 29, 32], artificial neural networks [30], case-based reasoning [6, 30], multiple linear regression [1], and fuzzy logic prediction [34]. The focus of this study is on software fault prediction using regression tree models.

A regression tree for software quality prediction, is a collection of decision rules represented by an abstract tree model. A set of *predictors* (independent variables)

Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS'02)
0-7695-1339-5/02 $17.00 © 2002 IEEE

are used to predict a *response variable* (dependent variable) such as number of faults. The root node and all internal nodes of the regression tree are decision nodes that use a predictor as a decision variable. The edges in the model lead either to a decision node or to a leaf node. Each leaf node is labelled with a quantity for the response variable. A software module traverses a path from the root node to a leaf node of the regression tree, according to the values of its predictors, and is assigned an appropriate value (e.x., average) to its response variable.

Regression trees are attractive due to their simplicity in model interpretation [7], and are particularly suited for effective data mining [13]. They can be constructed relatively fast compared to other prediction methods such as neural networks [18], and can be easily converted to SQL statements to access databases efficiently.

There are currently two techniques that facilitate software quality modeling using regression trees. These are CART (Salford Systems Inc.) and S-PLUS (Mathsoft Inc.). CART [2] provides two methods to build regression tree models, CART-LS (least squares) and CART-LAD (least absolute deviation). CART-LS uses the within-node mean value as its prediction for the response variable and the mean-square error as the criteria to rank different trees. CART-LAD uses the within-node median values as its prediction for the response variable and the mean-absolute deviation as the criteria to rank different trees.

A couple of studies [7, 32] have used S-PLUS [4] to build regression trees for fault prediction. Quite a few studies have been performed with CART as a software quality prediction technique [3, 11, 14]. A recent study stresses the importance of factors that affect the results of a prediction problem [26]. It that study it was indicated that there is a strong relationship between the success of a particular prediction approach and characteristics of the prediction problem. A simulation approach was adopted in [26] while evaluating common prediction techniques. Effect of characteristics such as data set size, nature of the 'cost' function, presence of outliers and distribution of the response variable (number of faults in our case) on the prediction technique were investigated.

However, it should be noted that data obtained from real world systems involve many uncontrollable characteristics and unknown factors other than those mentioned above. For example, factors such as skill of programmers and efficiency of the testing process affect the number of faults and their detection. Thereby, influencing the number of outliers. Distribution of the response variable obtained from real world systems is often very difficult to control. These unknown and uncontrollable characteristics make it difficult to quantify certain other factors that strongly affect the prediction problem.

This study is a comprehensive evaluation of regression trees built using CART-LS, S-PLUS, and CART-LAD algorithms. Our research group has performed empirical studies with CART-LS, CART-LAD, and S-PLUS regression tree algorithms using many large-scale case studies, including data collected over several project-releases [29]. Many large-scale case studies of real world software systems were used including those obtained from multiple releases of a project. In order to obtain relevant results that are not due to a lucky data split, we performed experiments using multiple data splits for a few case studies. In addition, empirical studies were also performed on the principal components (Principle Components Analysis) [17] of the data sets (for all case studies). However, in this paper we present our results for only one case study.

The tree-based algorithms are evaluated not only for prediction accuracy, but also based on the structure and complexity of their tree models. Performance metrics, average absolute error (AAE) and average relative error (ARE) are used to gauge the fault prediction accuracy of regression trees. Structure and complexity of trees are measured in terms of number of leaf nodes of the model and the number of independent variables used by the model.

The case study used in our empirical studies with CART and S-PLUS comprises of software design metrics collected from a large network telecommunications system [35]. The *fit* data set comprised of 4648 observations while the *test* data set comprised of 2324 observations. The data sets were obtained by applying an impartial data splitting technique to the collected data. 9 design metrics are used as independent variables to predict the number of faults in modules. It should be noted that for this case study only 9 software design metrics were available.

Software fault prediction models are built using the *fit* data set, while the *test* data set is used to validate the selected tree models. To our knowledge this is the first study that performs an extensive and comprehensive study of currently available regression tree modeling algorithms for predicting software faults. In-depth details of our empirical results obtained from many case studies are presented in [29]. Due to space restriction we have not provided detailed empirical findings obtained from all our case studies including those obtained from principle components analysis.

Empirical studies in this paper suggest that regression trees built with S-PLUS are large and complex as

compared to trees built using CART-LAD and CART-LS. For the case study presented in this paper, the predictive accuracy of CART-LAD was better than both CART-LS and S-PLUS. This is confirmed by statistical verification, i.e., Z-Test [1] (See Appendix). A similar result was observed in our empirical research with other case studies, including their principal components [29].

It should be noted that the presence of excess zero's and a heavy tailed fault distribution for the case study may have influenced the better prediction performance of CART-LAD. Investigation about the performance of CART-LAD and its median-approach with data that do not have lop-sided distribution remains an interesting area for further research. However, it is out of scope for this study.

The layout of the rest of the paper is as follows. The S-PLUS and CART regression tree algorithms are briefly presented in Sections 2 and 3 respectively. Section 4 describes the case study used in this paper, while Section 5 present our methodology in building, evaluating, and comparing tree models. Section 6 discusses the results of our comparative study, and Section 7 concludes this paper with final inferences and directions for future work.

2. S-PLUS Regression Trees

S-PLUS [4] is a solution for advanced data analysis, data mining, and statistical modeling. It combines an intuitive graphical user interface with an extensive data analysis environment to offer ease of use and flexibility. Hereon, we shall use S-PLUS to refer to its regression tree algorithm. The predictors are software metrics, treated by S-PLUS as ordinal measures, and are used to build regression trees to predict the response variable. The S-PLUS tree algorithm can process only numerical data. It constructs a regression tree which is a collection of decision rules determined by recursive binary partitioning of the training data set.

Decision rules can be controlled by the analyst by specifying certain parameters [4], which limit the growth of the tree model. These parameters are *minsize*, the size threshold which limits the number of observations in a leaf node and *mindev*, the uniformity threshold which limits the allowable deviance in the leaf nodes. By controlling these parameters the analyst can prune the tree model to the desired level and control overfitting [12]. However, S-PLUS provides a function that can be used to prune the tree after it has been constructed by the algorithm, without sacrificing the goodness-of-fit of the tree model [4]. In the course of the S-PLUS regression tree algorithm, modules in the *fit* data set are assigned to tree nodes. A software module is considered as an *object*

Predictors are derived from software metrics as explained below. Let x_{ij} be the j^{th} predictor's value for module i, $\mathbf{x}_i$ be the vector of predictors for module i, and y_i be the response variable, i.e., number of faults. The algorithm initially assigns all the modules in the *fit* data set to the root node. The algorithm then recursively partitions each node's modules into two subsets that are assigned to its child nodes, until a stopping criterion halts further partitioning.

The deviance of module i is minus twice the log-likelihood, scaled by the variance, which reduces to the following [4].

$$D(\mu_i; y_i) \quad = \quad (y_i - \mu_i)^2 \tag{1}$$

where, μ_i is estimated by the mean value of y over all training modules that fall in the same leaf as module i. The deviance of a node l is the sum of the deviances of all the training modules in the node [4].

$$D(\mu_l; y) \quad = \quad \sum_{i \in l} (y_i - \mu_i)^2 \tag{2}$$

The tree-building algorithm chooses the predictor whose best split maximizes the change in deviance between the deviance of the current node and the sum of the deviances of the prospective child nodes. The "best split" of a predictor partitions the current node's set of modules into two subsets choosing the cut-point that minimizes the sum of the deviances of the left and right prospective child nodes. Partitioning stops when the node deviance is less than a small fraction of the root node deviance.

$$\frac{D(\mu_l; y)}{D(\mu_{\text{root}}; y)} < mindev \tag{3}$$

or the number of modules (n_l) in the current node is less than a threshold,

$$n_l \quad < \quad minsize \tag{4}$$

where, *mindev* and *minsize* are parameters [29]. Let $L(\mathbf{x}_i)$ be the leaf that the i^{th} module falls into according to the structure of the tree. The predicted value of the response variable for module i is the mean of training modules in the leaf it falls into.

$$\hat{y}_i \quad = \quad \mu_{L(\mathbf{x}_i)} \tag{5}$$

3. CART Regression Trees

In regression, a case consists of data $(\mathbf{x}_i, y_i)$ where, $\mathbf{x}_i$ is the i^{th} measurement vector of independent variables and y_i is the i^{th} response variable. The CART [2]

regression tree algorithm partitions the input data set into terminal nodes by a sequence of recursive binary splits. Binary splits are generated by CART, based on the significant independent variables. A teach binary partition, the two subsets are as homogeneous as possible with respect to the dependent variable, which in our case is the number of faults in the software module.

CART regression tree algorithms initially build a large tree [2, 3], and then prune it backwards using *cross validation* to avoid overfitted trees [1, 2]. Starting with the *fit* data set (learning sample L), three elements are necessary to determine a regression tree predictor:

1. A way to select a split at every internal node.
2. A rule for determining when a node is terminal.
3. A rule for assigning a value $\hat{y}(t)$ to every terminal node. $\hat{y}(t)$ is the predicted value of the response variable for terminal node t.

CART provides three waysto estimate accuracy of regression trees; resubstitution estimate, test sample estimate and v-$fold$ cross validation estimate. Empirical studies in this paper use the v-$fold$ cross validation estimate to evaluate regression tree models and control overfitting [12]. In a v-$fold$ cross validation estimate, the learning sample L (*fit* data set) is divided into v subsets of approximately equal size. (v-1) subsets are used as *fit* data sets while one remaining subset is used as a *test* data set. v such trials are performed such that each subset of the learning sample is used once as a *test* data set. The average error over these v trials, gives the cross validation error estimate. In our empirical studies we have used the 10-$fold$ cross validation estimate approach.

Certain parameters can be controlled when building regression trees with CART. These include number of terminal nodes, depth or level ofregression tree, and node size before splitting. In the following two sections we present the essential differences between theCART-LS (*Least Squares*) and CART-LAD (*Least Absolute Deviation*) methods. Further in-depth mathematical details including, tree pruning methods and standard error estimates of the CART regression tree algorithms can be found in [2].

3.1 Least Squares Method

This method generates regression trees using the within node mean value observed in each terminal node as its predicted value. Ranking of regression trees of different sizes is evaluated based on the *mean square error* estimate. Given a learning sample L consisting of $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$, L is used to construct regression trees and also to estimate their accuracy.

The resubstitution error estimate for the CART-LS method, as a measure of accuracy for a regression tree T is given by,

$$R(T) = \frac{1}{N} \sum_{t \in T} \sum_{(\mathbf{x}_n, y_n) \in t} (y_n - \bar{y}(t))^2 \qquad (6)$$

where, N is the total number of cases in thelearning sample, t is a terminal node in the regression tree T, and $\bar{y}(t)$ is the mean value of response variables in t.

Given any set of possible splits $\mathbf{S}$ of a current terminal node t, the best split $\check{s}$ of t is that split in $\mathbf{S}$ which most decreases $R(T)$. For any split s of t into t_L and t_R, let $\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$. Then the best split $\check{s}$ is given by,

$$\Delta R(\check{s}, t) = max[\Delta R(s,t)] \qquad (7)$$

Thus, a regression tree is formed by iteratively splitting nodes so as to maximize the decrease in $R(T)$. The best split at a node is that split which most successfully separates the high response values from the low ones.

When using the test sample estimate, a *fit* data set is used to build regression trees, while a *test* data set is used to evaluate the accuracy of the tree models. The test sample error estimate for the CART-LS method is given by,

$$R^{ts}(T) = \frac{1}{N_2} \sum_{(\mathbf{x}_n, y_n) \in L_2} (y_n - d(\mathbf{x}_n))^2 \qquad (8)$$

where, L_2 is the test data set with N_2 cases. $d(\mathbf{x}_n)$ denotes the predictor corresponding to the $\mathbf{x}_n$ measurement vector of independent variables. The learning sample L_1 is used to build regression trees, while the test sample L_2 is used to evaluate the accuracy of trees.

The v-$fold$ cross validation error estimate for the CART-LS method is given by,

$$R^{cv}(T) = \frac{1}{N} \sum_{v} \sum_{(\mathbf{x}_n, y_n) \in L_v} (y_n - d_v(\mathbf{x}_n))^2 \qquad (9)$$

where, $d_v(\mathbf{x}_n)$ denotes the predictor for the v^{th} trial of the cross validation and L_v is the v^{th} subset of the learning sample L.

Let $\bar{y}$ be a sample mean of $y_1, ..., y_N$, and set $R(\bar{y})$ as,

$$R(\bar{y}) = \frac{1}{N} \sum_{n} (y_n - \bar{y})^2 \qquad (10)$$

Then the mean square relative error estimates for resubstitution, test sample and v-$fold$ cross validation are given by, $R(T)/R(\bar{y})$, $R^{ts}(T)/R(\bar{y})$ and $R^{cv}(T)/R(\bar{y})$ respectively.

3.2 Least Absolute Deviation Method

This method generates regression trees using the within node median value observed in each terminal node as its predicted value. Ranking of regression trees of different sizes is evaluated based on the *mean absolute deviation* estimate.

The resubstitution error estimate $R(T)$ for the CART-LAD method is given by,

$$R(T) = \frac{1}{N} \sum_{t \in T} \sum_{(\mathbf{x}_n, y_n) \in t} |y_n - \tilde{y}(t)| \qquad (11)$$

where, N is the total number of cases in the learning sample, t is a terminal node in the regression tree T and $\tilde{y}(t)$ is the median value of the y values in node t. The splitting criteria is analogous to that mentioned for the CART-LS method.

The test sample error estimate for the CART-LAD method is given by,

$$R^{ts}(T) = \frac{1}{N_2} \sum_{(\mathbf{x}_n, y_n) \in L_2} |y_n - d(\mathbf{x}_n)| \qquad (12)$$

The $v\text{-}fold$ cross validation error estimate for the CART-LAD method is given by,

$$R^{cv}(T) = \frac{1}{N} \sum_{v} \sum_{(\mathbf{x}_n, y_n) \in L_v} |y_n - d_v(\mathbf{x}_n)| \qquad (13)$$

Let $\tilde{y}$ be a sample median of $y_1, ..., y_N$, and set $R(\tilde{y})$ as,

$$R(\tilde{y}) = \frac{1}{N} \sum_{n} |y_n - \tilde{y}| \qquad (14)$$

Then the absolute relative error estimates for resubstitution, test sample and $v\text{-}fold$ cross validation are given by, $R(T)/R(\tilde{y})$, $R^{ts}(T)/R(\tilde{y})$ and $R^{cv}(T)/R(\tilde{y})$ respectively.

4. System Description

We studied a very large-scale telecommunications software system consisting of almost 13 million lines of code within procedures overall. The system was programmed in PROTEL, a high-level language similar to the C programming language. A *module* was defined as a set of source code files that work together to perform a function. In some software development environments, models based on metrics of product attributes alone do not give useful accuracy. Information regarding the reuse history of each module can be helpful in software quality prediction modeling [15, 16].

Table 1. System Profile: Changed Modules

Application	Telecommunications
Language	PROTEL
Lines of Code	7.0 million
Executable Statements	6.0 million
Control Flow Graph Edges	2.0 million
Source Files	18.0 thousand
Changed Modules	7.0 thousand
Design Metrics	9
Quality Metric	Number of faults

Table 2. Fault Distribution

Faults	Modules
0	3537
1	1777
2	700
3	332
4	183
5	103
≥ 6	340
Mean	1.25
Median	0
Std*	2.49

* Std is the Standard Deviation

In the case study presented here, we were able to determine which modules were reused and changed from the previous release. We limited the scope of the case study to this subset of modules, summarized in Table 1, representing about seven million lines of code.

A software *fault* for our case study is defined as a reported problem that caused a change in the code. Data on faults in each module were collected (by EMERALD)

Table 3. Software Design Metrics

Symbol	Description
Call Graph Metrics	
UCT	Unique procedure calls
TCT	Total calls to others
NDI	Distinct files included
Control Flow Graph Metrics	
VG	McCabe's cyclomatic complexity
NL	Number of loops
$IFTH$	Number of if-then structures
$NELTOT$	Total nesting level
$PSCTOT$	Total number of vertices within the span of loops or if-then structures
$RLSTOT$	Total edges plus vertices within loop structures

during unit test, system test, beta test, and operations. Enhanced measurement for early risk assessment of latent defects (EMERALD) was a joint project of Nortel and Bell Canada for improving reliability of telecommunications software products. The fault distribution and characteristics of the entire data set for our case study is presented in Table 2. Majority of the modules have faults less than 3 and only relatively few modules have a higher number of faults. This is a strong data characteristic that influences the performance of a prediction technique [26]. Software metrics were collected on almost 7000 changed modules using the DATRIX software analyzer [22, 25], a key component of EMERALD. 9 software design metrics were made available, as listed in Table 3.

Software design metrics allow predicting number of faults in modules well before the testing phase. A number of studies have evaluated software design metrics as inputs to software quality models [20, 35]. A *call graph* depicts dependencies among procedures. Call graph metrics can be collected as early as the high-level design phase. A *control flow graph* depicts the flow of control of an algorithm with a graph made of edges and vertices. Control flow graph metrics can be collected as soon as the detailed design of algorithms is complete. An impartial data splitting [29] was used to obtain the *training* or *fit* (4648 observations) and *test* (2324 observations) data sets.

Threats to external validity are conditions that limit generalization of case study results. The software engineering community demands that the subject of an empirical study be [33]: (1) developed by a group, not an individual, (2) developed by professionals, not students, (3) developed in an industrial environment, not simulated, and (4) large enough to be comparable to real industry projects. The case studies used in our work fulfill these criteria, thereby, avoiding threats-to-validity due to artificial settings and simulations.

5. Methodology

5.1 Building Regression Tree Models

For the case study described in Section 4, regression trees with CART-LS, S-PLUS, and CART-LAD algorithms were built using the following methodology. The response variable, i.e, number of faults, was predicted using 9 predictors, i.e., software design metrics. A brief description of our empirical approach is presented below, however, a more detailed description of our experiments is elaborated in [29].

1. *Formatting Data*:
   Regression tree modeling using CART-LS and CART-LAD algorithms, requires the data sets be converted to the SYSTAT file format [31]. The training (*fit*) and validation (*test*) data sets were converted to this format prior to building tree models.

2. *Developing Models*:
   The training data set is used to build different models. CART and S-PLUS algorithms allow the analyst to vary certain parameters while constructing regression trees. In the case of CART, parameters include number of terminal nodes, depth of regression trees, and minimum node size before further splitting. The first two by default are automatically forecasted by the algorithm depending on the case study. The third parameter is set to 10 by default. In our experiments with CART-LS and CART-LAD [29], we have used default values for these parameters. In the case of S-PLUS, parameters include minimum node deviation and minimum node size before further splitting. In our experiments with S-PLUS [29], we built regression trees using minimum node size values of 10, 20, 30, 40, 50, 70, and 100. For each minimum node size we varied minimum node deviation with values 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.15, and 0.2.

3. *Selecting Models*:
   In the case of CART-LS and CART-LAD, models are selected based on the lowest cross-validation relative error values[1]. These cross-validation values are based on the *fit* data set. In the case of S-PLUS, AAE and ARE values of models built are computed for the *fit* data set and the model with the lowest values is selected as the final model.

4. *Validating Models*:
   The *test* data set is used to evaluate fault prediction accuracy of models selected in step 3. Performance metrics, AAE and ARE for the *test* data set are computed, and are used to observe prediction accuracy of tree models.

5.2 Performance Metrics

Fault prediction accuracy of the tree models is determined by estimating commonly used performance metrics: average absolute error (AAE) and average relative error (ARE), given by,

$$AAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (15)$$

[1] Not to be confused with ARE. Please refer to Section 3 for details.

Table 4. Regression Tree Models

Metric	Data Set	CAR T-LAD	S-PLUS	CART-LS
TN	Fit	4	15	14
CVRE	Fit	0.8960	na	0.8310
IVBM	Fit	9	9	9
IVUM	Fit	2	5	3
AAE	Test	1.1308	1.2889	1.2894
ARE	Test	0.3943	0.6845	0.6853

Table 5. Z-Test Results (Test data): p-values

Z-Test	AAE	ARE
CAR T-LADvs S-PLUS	0.00570	< 0.00003
CAR T-LADvs CAR T-LS	0.00480	< 0.00003
S-PLUS vs CAR T-LS	0.49600	0.48800

$$ARE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i + 1}\right| \quad (16)$$

where, n is the number of modules in the *test* data set. The denominator in ARE has a one added to avoid division by zero [19]. Our study compares fault prediction models using both AAE and ARE, since the effectiveness of one over the other is out of scope for this paper.

6. Empirical Results

Regression trees were built using the 9 software design metrics (Table 3) as independent variables. Table 4 summarizes the regression trees built for the case study. Some notations of Table 4 are as follows, *TN*: number of leaf or terminal nodes, *CVRE*: cross validation relative error (only for CART trees), *IVBM*: independent variables used to build the model, and *IVUM*: independent variables used by the model.

Figures 1, 2 and 3 represent the tree models built using CART-LS, CART-LAD, and S-PLUS algorithms respectively. The diamond shaped nodes are decision nodes and the independent variable shown above them is the decision variable for that node. A rectangular node is a leaf node and the value inside it is the predicted number of faults for modules that fall into the node. The values next to the outgoing edges of a decision node indicate the threshold values of the decision variable at that node. The number of modules within a node are indicated outside the node.

RLSTOT (total number of edges plus vertices within loop structures of a module) and *VG* (McCabe's complexity metric) are used by both the CART models. In addition to these two independent variables the CART-LS model also uses *NL*: number of loops in a module. These three metrics are control flow graph design metrics, while none of the call graph metrics of Table 3 are used by either of the two models.

The CART-LS model indicates that modules with $RLSTOT < 260$ will have approximately 1 fault and those with $RLSTOT > 259$ will have 3 or more number of faults. Modules with very high *VG* values will have at least 5 or more number of faults. The CART-LAD model indicates that modules with $VG < 54$ will have no faults and on the other hand, modules with $VG > 952$ will have a high number of faults. The S-PLUS tree model (Figure 3) indicates that modules with $NL > 14$ and $IFTH > 1375$ (number of if-then constructs) will have the highest number of faults. We observe that *VG* and *RLSTOT* are common to all three models and hence, are the relatively more important attributes for the system. In-depth explanation of the structure and internals of the tree models for the case study are presented in [29].

We observe from Table 4 that the S-PLUS tree model is much larger than the CART tree models. It has 15 terminal nodes as compared to the 4 leaf nodes of the two CART models. However, it should be noted that CART has been shown to over-prune tree models [14, 29]. Larger trees are relatively harder to interpret and analyze. Smaller trees are preferred by analysts since they are usually less complex, easier to interpret, and attractive to the eye. As a measure of complexity we recorded the number of independent variables used by the tree models. Hence, from Table 4, we can observe that CART-LS and CART-LAD tree models are less complex than the S-PLUS tree model. The S-PLUS model utilizes 5 out of the 9 independent variables used in building models, as compared to 3 variables used by the CART-LS model and only 2 variables used by the CART-LAD model. A very similar observation about the structure and complexity of the CART and S-PLUS trees was made from our results obtained with other case studies [29].

Fault prediction error estimates, AAE and ARE, for the tree models are recorded in Table 4. The values shown are only for the *test* data set, since the *fit* data set was used to build models. The CART-LAD model has lower AAE and ARE values than both CART-LS and S-PLUS models. To determine if this difference is of significance, we performed a *Z*-Test (5% significance level) for CART-LAD vs. CART-LS and CART-LAD vs. S-PLUS. The tests were performed for both AAE and ARE as response values. The alternate hypothesis for the *Z*-Test was that the response variables (AAE and ARE) of the CART-LAD model are better than those for the CART-LS and S-PLUS models. The p-values for the tests are presented in Table 5, and it is observed from

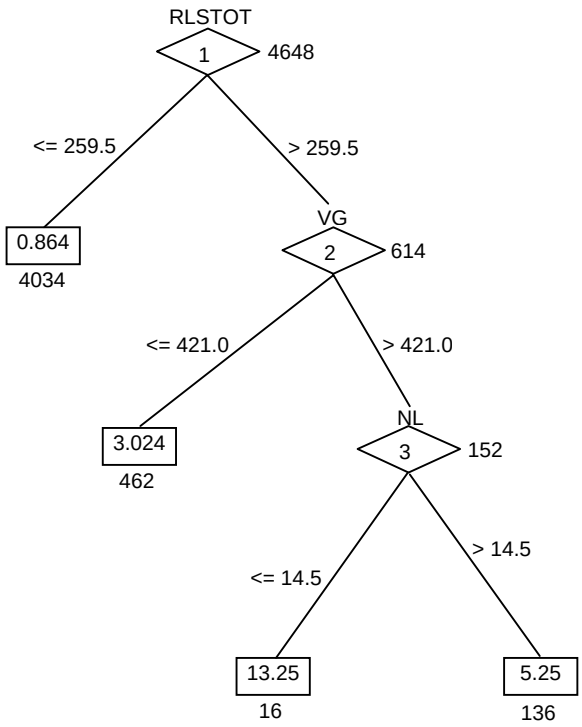

Figure 1. CART-LS Tree Model

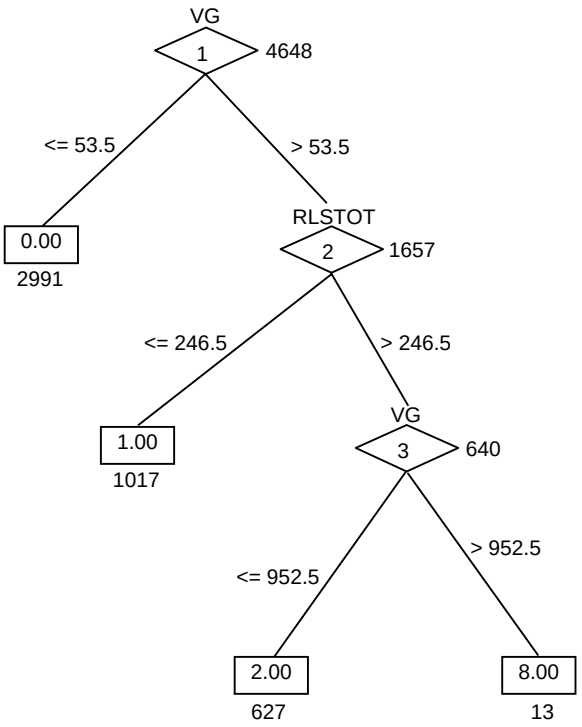

Figure 2. CART-LAD Tree Model

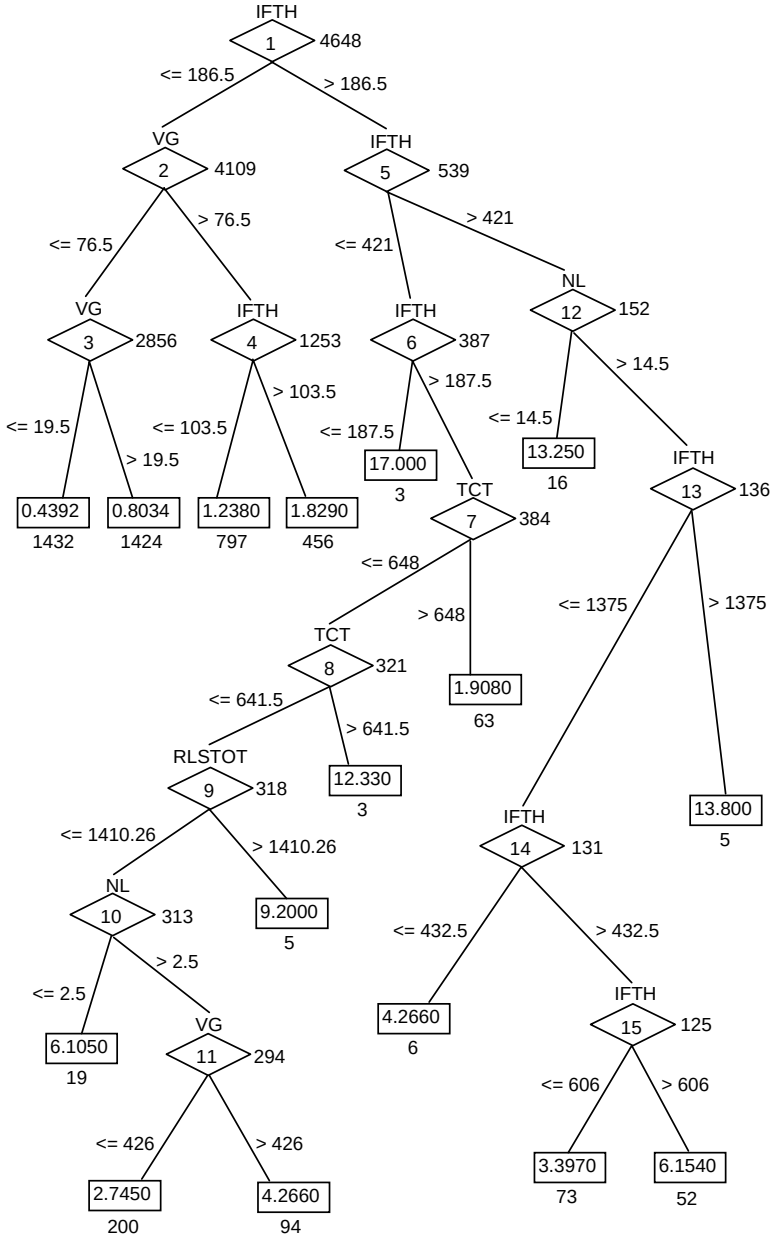

Figure 3. S-PLUS Tree Model

the very low p-values that CART-LAD has much better predictive accuracy than both CART-LS and S-PLUS.

Prediction accuracy of S-PLUS and CART-LS were very similar (Table 4). This is verified by the high p-values for both AAE ($p = 0.496$) and ARE ($p = 0.488$) in Table 5. Thus, for our case study we conclude that the CART-LAD tree model has the best predictive accuracy and is also simple and easy to interpret and analyze. On the other hand, we observe that the S-PLUS tree model is relatively larger and more complex than the CART models. The results obtained in our studies does not imply that CART-LAD will always yield good fault prediction. The case study, empirical evaluation environment, and other factors mentioned in [26] are important attributes of any prediction problem.

7. Conclusion

Complex high-assurance and mission-critical software systems are heavily dependent on stability and reliability of their underlying software applications. Early fault prediction techniques based on software metrics are being used to predict software quality before its deployment and operations phase. Regression trees can serve as simple, attractive, and efficient software quality models that predict number of faults in modules. Tree models are attractive to analysts and scientists because of their white-box feature [4], i.e., internal details of how the model works are clearly depicted.

In this paper we have presented a case study from our comprehensive evaluation of currently available regression tree algorithms for software quality modeling, i.e., CART-LS, S-PLUS, and CART-LAD. Many large-scale case studies of real world software systems were used including those obtained from multiple releases of a project. In order to obtain relevant results that are not due to a lucky data split, we performed experiments using multiple data splits for a few case studies. In addition, empirical studies were also performed using principal components of the data sets (for all case studies). However, due to limited space results for only one case study was presented (See [29] for other case studies). Software metrics (9 design metrics) collected from a large scale network telecommunications system were used as our case study which comprised of almost 7000 observations. Regression trees were built to predict the number of faults in modules, and performance metrics AAE and ARE were used to evaluate fault prediction accuracy of trees.

It was observed that tree models built with S-PLUS were larger and more complex than those built with CART-LS and CART-LAD. The S-PLUS tree model had a high number of leaf nodes (15) and utilized 5 out of the 9 independent variables used. On the other hand, both CART-LS and CART-LAD yielded simpler and more compact prediction trees. Both of them had only 4 leaf nodes. Out of the 9 metrics used in model building, the CART-LS model involved 3 independent variables while the CART-LAD model utilized only 2 independent variables. Fault prediction accuracy (AAE and ARE) of CART-LAD was better than both S-PLUS and CART-LS. This was statistically verified in our study by obtaining p-values from the Z-Test. Predictive accuracy of S-PLUS and CART-LS were statistically similar, i.e., Z-Test yielded very high p-values for both AAE and ARE. A similar inference about the prediction accuracy and complexity of CART-LAD, S-PLUS, and CART-LS tree models was observed with other case studies investigated.

Future work will involve comparing tree-based prediction algorithms with other (non tree-based) prediction algorithms, like neural networks, multiple linear regression and case-based reasoning. Investigation about the performance of CART-LAD and its median-approach with data that do not have lop-sided distribution remains an interesting area for further research.

Acknowledgments

We thank John P. Hudepohl, Wendell D. Jones and the EMERALD team for collecting the case-study data. This work was supported in part by Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division, and Center Software Initiative for the NASA software Independent Verification and Validation Facility at Fairmont, West Virginia.

References

[1] M. L. Berenson, D. M. Levine, M. Goldstein. *Intermediate Statistical Methods And Applications*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1983, ISBN 0-13-470781-8.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification And Regression Trees*. Wadsworth International Group, Belmont, California, 2nd. edition, 1984.

[3] L. C. Briand, Tristen Langley, and Isabella Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. *International Conference on Software Engineering*, ACM, pages 113-206, Limerick Ireland, Sept. 2000.

[4] L. A. Clark and D. Pregibon. Tree-based models. In J. M. Chambers and T. J. Hastie, editors, *Statistical Models in S*, pages 377–419. Wadsworth, Pacific Grove, California, 1992.

[5] G. R. Finnie, G. E. Wittig, and J. M. Desharnais A comparison of software effort estimation techniques: Using function points with neural networks,case-based reasoning and regression models. *Journal of Systems and Software*, 39:281–289, 1997.

[6] K. Ganesan, T. M. Khoshgofaar, and E. B. Allen. Case-based software quality prediction. *International Journal of Software Engineering and Knowledge Engineering*, 10(2):139–152, 2000.

[7] S. S. Gokhale and M. R. Lyu. Regression tree modeling for the prediction of software quality. In H. Pham, editor, *Proceedings of the Third ISSAT International Conference on Reliability and Quality in Design*, pages 31–36, Anaheim, CA, Mar. 1997. International Society of Science and Applied Technologies.

[8] A. R. Gray and S. G. Macdonnel Software metrics data analysis - Exploring the relative performance of some commonly used modeling techniques. *Journal of Empirical Software Engineering*, 4:297–316, 1999.

[9] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand. EMERALD: Software metrics and models on the desktop. *IEEE Software*, 13(5):56–60, Sept. 1996.

[10] W. D. Jones, J. P. Hudepohl, T. M. Khoshgoftaar, and E. B. Allen. Application of a usage profile in software quality models. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, pages 148–157, Amsterdam, Netherlands, Mar. 1999. IEEE Computer Society.

[11] T. M. Khoshgoftaar and E. B. Allen. Modeling software quality with classification trees. In *Recent Advances in Reliability and Quality Engineering*, Hoang Pham Editor, World Scientific, Ch. 15, 247–270, 2001.

[12] T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Controlling overfitting in software quality models: Experiments with regression trees and classification. In *Proceedings of the Seventh International Software Metrics Symposium*, pages 190–198, London, England UK, April 2001. IEEE Computer Society.

[13] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Data mining for predictors of software quality. *International Journal of Software Engineering and Knowledge Engineering*, 9(5):547–563, 1999.

[14] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Accuracy of software quality models over multiple releases. *Annals of Software Engineering*, 9:103–116, 2000.

[15] 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.

[16] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, N. Goel, J. P. Hudepohl, and J. Mayrand. Detection of fault-prone program modules in a very large telecommunications system. In *Proceedings of the Sixth International Symposium on Software Reliability Engineering*, pages 24–33, Toulouse, France, Oct. 1995. IEEE Computer Society.

[17] T. M. Khoshgoftaar, E. B. Allen, and R. Shan. Improving tree-based models of software quality with principal components analysis. In *Proceedings of the Eleventh International Symposium on Software Reliability Engineering*, pages 198–209, San Jose, California USA, October 2000. IEEE Computer Society.

[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, J. C. Munson, B. B. Bhattachary a, and G. D. Richardson. Predictive modeling techniques of software quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, Nov. 1992.

[20] B. A. Kitchenham, L. M. Pickard, and S. J. Linkman. An evaluation of some design metrics. *Software Engineering Journal*, 5(1):50–58, Jan. 1990.

[21] M. R. Lyu. Introduction. In M. R. Lyu, editor, *Handbook of Software Reliability Engineering*, chapter 1, pages 3-25. McGraw-Hill, New York, 1996.

[22] J. Mayrand and F. Coallier. System aquisition based on software product assessment. In *Proceedings of the Eighteenth International Conference*

on Software Engineering, pages 210–219, Berlin, March 1996. IEEE Computer Society.

[23] J. C. Munson and T. M. Khoshgoftaar. The dimensionality of program complexity. In *Proceedings of the Eleventh International Conference on Software Engineering*, pages 245–253, Pittsburgh, Pennsylvania USA, May 1989. IEEE Computer Society.

[24] N. Ohlsson, M. Zhao, and M. Helander. Application of multivariate analysis for software fault prediction. *Software Quality Journal*, 7:51–66, 1998.

[25] P. N. Robillard, D. Coupal, and F. Coallier. Profiling software through the use of metrics. *Software-Practice and Experience*, 21(5):507–518, May 1991.

[26] M. Shepperd and G. Kadoda. Using simulation to evaluate prediction techniques. In *Proceedings of the Seventh International Software Metrics Symposium*, pages 349–359, London, England UK, April 2001. IEEE Computer Society.

[27] N. F. Schneidewind. Software metrics validation: Space Shuttle flight software example. *Annals of Software Engineering*, 1:287–309, 1995.

[28] N. F. Schneidewind. Software metrics model for integrating quality control and prediction. In *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pages 402–415, Albuquerque, NM USA, Nov. 1997. IEEE Computer Society.

[29] N. Seliya. Software fault prediction using tree based models. Master's thesis, Florida Atlantic University, Boca Raton, Florida USA, August 2001. Advised by Taghi M. Khoshgoftaar.

[30] N. Sundaresh. An empirical study of analogy based software fault prediction. Master's thesis, Florida Atlantic University, Boca Raton, Florida USA, May 2001. Advised by T. M. Khoshgoftaar.

[31] D. Steinberg and P. Colla. *CART: A supplementary module for SYSTAT* Salford Systems, San Diego, CA, 1995.

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

[33] L. G. Votta and A. A. Porter Experimental Software Engineering: A report on the state of the art. In *Proceedings of the 17th. International Conference on Software Engineering*, pages, 277–279, Seattle, WA USA, Apr. 1995.

[34] Z. Xu. Fuzzy logic techniquesfor software reliability engineering. Ph.D. dissertation, Florida Atlantic University, Boca Raton, Florida USA, May 2001. Advised by T. M. Khoshgoftaar.

[35] W. M. Zage and D. M. Zage. Evaluating design metrics on large-scale software. *IEEE Software*, 10:75–80, July 1993.

Appendix

A. Z-Test

Z-Test is a statistical approach to determine whether differences between two methods or results, is of any important significance. Hypothesis testing is concerned with the testing of certain specified values for two population parameters. A *null* hypothesis, H_0, is tested against its compliment, the *alternate* hypothesis, H_A. Hypotheses are usually set up to determine if the population supports the hypothesis as specified by H_A.

Once the hypothesis H_0 and H_A are stated, a *significance level* 'α' is selected, which facilitates accessing the degree of significance between the hypothesized parameter value and the corresponding statistic to be obtained from sample information. Significance levels of $\alpha = 0.05$ or $\alpha = 0.01$ have been the traditional values used by mathematicians and statisticians. Let's denote Y_1 and Y_2 be two populations (or methods) of size n_1 and n_2, respectively. *Z-Test* of two populations involves the following steps:

State Hypotheses: The *null* hypothesis and the *alternate* hypothesis are defined as follows,

$$H_0 : \mu(Y_1) \leq \mu(Y_2) \tag{17}$$

$$H_A : \mu(Y_1) > \mu(Y_2) \tag{18}$$

In the above equations, $\mu(Y_1)$ and $\mu(Y_2)$ are the sample means of the populations, Y_1 and Y_2, respectively.

Compute Z Value:

$$Z = \frac{\bar{y}_1 - \bar{y}_2}{\sqrt{{s_1}^2/n_1 + {s_2}^2/n_2}} \tag{19}$$

In the above equation, $\bar{y}_1$ and $\bar{y}_2$, are the sample means while s_1 and s_2 are the sample standard deviations of the populations Y_1 and Y_2, respectively.

Decision Making The *null* hypothesis H_0 is rejected if the condition $Z \geq Z_{1-\alpha}$ is true, otherwise it is accepted (i.e., $Z < Z_{1-\alpha}$). The two populations are of significant difference if H_0 is rejected.