Alacart: Poor Man's Classification Trees

By Leonardo E. Auslender
SAS Institute, Inc., Cary, NC
At regional office in Bedminster, NJ
leausl@wnt.sas.com

1. Introduction [1]

Breiman's et al 1984 monograph is the basis for CART, one of the most important **C**lassification **A**nd **R**egression **T**ree methods. Other tree methods and corresponding software include C4.5 (Quinlan, 1993), Mathsoft's S+ (Clark and Pregibon, 1992), Chaid (Kass). See Steinberg (1993) and references therein.

This paper presents a SAS macro system, Alacart [2], for classification trees ala Breiman's CART, and describes the steps involved in tree methodology. Many important details are left to the reader's initiative in reading either the 1984 monograph, or any other convenient source.

The system's output is in tabulate and in graphical forms. Most of the processing is done in the macro language, although the entire system could be written in SAS/IML . IML is utilized in Alacart to produce mostly the graphs.

The goal of the paper is to first show that it is possible to write CART code in SAS , and thus avoid the current 'black-box' syndrome. As such, it is an invaluable tool to understanding tree methods. Second and most importantly, it enables the user to explore alternative methods of splitting variables, and pruning trees, of which we will give an example.

We start by describing the CART methodology, and illustrate by way of a business case application, that uses the criterion of minimization of impurity rates. As an illustration, we also consider briefly the same application by using the Gini splitting criterion, which is the most popular in the CART literature.

2. Core of the Tree Classification Methodology.

We will concentrate on binary classification of a dependent (0,1) $Y^{(n \times 1)}$ variable. Label the 0's as 'bad', and the 1's as 'good'. Assume that the proportion of 0's is 50%. In this case of no majority, it is not possible to classify the initial data set as either good or bad according to Y. Were the proportion of "bads" 30%, say, the initial data set would be classified as "good".

Assume a corresponding data set of predictors $X^{(n, K)}$ of Y, either continuous or categorical or ordinal. For simplicity, assume the non-existence of missing values[3]. We utilize the majority rule to classify data subsets.

The tree methodology proceeds thus to separate 'goods' from 'bads'. Take the first variable in X, X_1 , in sorted order . Assume that X_1 is continuous. Consider its first value, X_{11}. Split the Y values into two sets, one set corresponding to Y values associated with values less than or equal to X_{11}, and the other set corresponding to values strictly greater. Calculate the "impurity"[4] rate of the two subsets, and their weighted sum. Repeat this exercise for each observation of X_1, and for each of the remaining predictors. Choose as your 'splitting' variable X_s at value X_{s0} the split that minimizes the impurity of the 'newly created' Y subsets.

At this moment, we have started from a unique data set, with some initial misclassification rate, and obtained two subsets, the combined misclassification of which is smaller than the original one.

Once a split is found, the process is iterated on each of the 'children' nodes. In tree terminology, the original or starting set is called the 'root' or 'parent' node, and the two subsets are called left and right nodes. The process is stopped when there is no more reduction in 'impurity'. The process is also stopped when either a minimum node size is reached (user determined, usually 5 observations), or complete purity is achieved. Once stopped, a full branch and a final node with a classification label are defined. Searching for alternative splits in the most

[1] I thank Yung-Seop "Chingoo" Lee for fruitful discussions, who however does not share in any remaining errors, which are solely mine. Alacart is a macro system with close to 3000 lines of code, plus utility macros I wrote before Noah's times. There are no plans at present to make this system publicly available.

[2] Bruce Erlichman's inventiveness deserves kudos for the name Alacart. His alternative, Magna Cart, might be used in the future.

[3] CART provides very useful tools for handling missing values. See Breiman's et al 1984 reference.

[4] There are different ways of defining impurity, impurity rates, etc. See Breiman et al 1984's monograph. We utilize misclassification rates as 'impurity' in this explanation.

recently created branch, as explained below restarts the process. A standard graphic representation is given in the appendices. [5]

2.1 Some SAS coding.

The 'engine' of the SAS code necessary to find the optimal split is given by a 'proc summary' inside a macro do loop, in which the only change is the index which denotes a different X variable. For instance;

```
%do k = 1 %to &numvar.;
 proc summary data = &indata.;
  class &&var&k.;       /* indep. Var*/
  var  &depvar.;        /* dep var */
  output out = sumout sum =;
 run;
……..
%end;
```

The information contained in the output data set 'sumout' is sufficient to determine both the variable and the value that represent the best possible split across all X variables and their corresponding values, given previous impurity.

The key element and potential trouble area in this code is the number of non-duplicate observations in the data set when the variable in question is continuous. For very large data sets, computer resources might prevent the task from being completed, in which case a subsample is advisable. For categorical variables with C categories, CART will search over $2^{C-1} - 1$ possible 2-subgroups. For C greater than 10 and many categorical variables, the computing time would be excessive.

2.2 Some programming considerations.

The careful programmer will realize by now that the process of keeping track of the progression of children nodes is in itself laborious. In Alacart, the progression of the splits is kept in a macro variable, to which new information is added. The split proceeds 'normally' by searching the '<=' side of the branch. For instance, before a new split, the information is kept in a macro variable as:

```
VARB <= 0.99521057 & VARA <= -
93.516172 & VARE <= -2.4123315
```

And after the split, the same macro variable becomes:

```
VARB <= 0.99521057 & VARA <= -
93.516172 & VARE <= -2.4123315 & VARA
<= -121.22184
```

Notice in the second macro variable the addition of VARA <= -121.22184, which is the 'typical' search. Once the final node is reached, then the process continues by first outputting the 'branch' to an ASCII file, which accumulates the different 'if-then' branches. This file is typically called 'ALGOR0.PGM', where the '0' refers to the uncertainty about the number of final nodes or branches that it will eventually contain, once no more splits are possible.

Alacart then searches for the characters '<=' in the macro variable, backwards. As soon as one set of characters '<=' is found, it is turned into '>' and the process restarts. In the code above, the last addition would be 'VARA > -121.22184'. Once no more '<=' are found in the most recently created branch, the splitting process is finished, and the number of nodes for the 'maximal' tree is determined. Thus, the first or leftmost branch contains only "<=' inequalities, and the rightmost only ">" inequalities.

In addition to the macro variable just described, alacart also creates macro variables in the same fashion for impurity rates, node classification, number of observations per node and a special link notation, which is necessary in order to determine for any node, its relative position in the overall tree.

2.3 Pruning.

The tree thus obtained could be quite large and would obtain a seemingly low overall misclassification rate. This effect is the mirror effect of model over fitting, also known as lack of generalization in the Machine Learning literature. That is, when the algorithm generated by the tree is used to 'score' other samples, the misclassification rates are larger than they could be if the tree were smaller. The pruning process, which is specific to Breiman's et al methodology, aims at remedying the over fitting problem.

Pruning, a bottom-up process, starts from the tree originally created and selectively recombines nodes and obtains a decreasing sequence of subtrees. The decision as to which final nodes to recombine depends on comparing the loss in accuracy from not splitting an intermediate node in relation to the number of final nodes that that split generates. The comparison is made across all possible intermediate node splits, and the 'minimal cost-complexity' loss in accuracy is the rule for pruning. Each subtree , represented by a set of "if then…else" statements is stored as an ASCII file called ALGORX.PGM, the x being replaced by the actual number of final nodes.

[5] *Other representations are possible, such as including number of observations going left and right, intermediate nodes classification, etc.*

The sequence of subtrees generated ends up with the root node. The decision as to which tree among the subtrees to utilize is based on either one of two methods: 1) cross-validation, or 2) a test-data set.

2.3.1 Cross-validation.

Cross-validation is the preferred method when the original data set is not 'large'. In this case, 'v' stratified samples on the dependent variable are created, without replacement. Let create 'v' data sets, each one containing (v –1) of the samples created, and 'v' test data sets, which consists of the 'left-out' sample. 'v' maximal trees are trained on the 'v' samples, and pruned.

The 'v' test data sets are then used to obtain the misclassification rates of each of the pruning subsequences. Index each pruning subsequence and corresponding misclassification rate by the number of final nodes. Thus, we obtain an array of misclassification rates by pruned subtrees, by summing over the individual 'v' corresponding rates for each final node index. Choose the size of the tree that which minimize the misclassification rate.

The final tree will be taken from the original pruning sequence of the tree derived with the entire sample at the number of final nodes just described.

2.3.2 Test data set.

The test data set method is the one used in this paper, to be preferred when the size of the data set is not a constraint on the estimation process. Split the original data set into training and test subsets.

Once the maximal tree and the sequence of subtrees due to pruning are obtained, 'score' the different subtrees with the test data set and obtain the corresponding misclassification rates. Choose that subtree which minimizes the misclassification rate. While this rate decreases with the number of final nodes at the stage of tree development, it typically plateaus at some number of final nodes smaller than the maximal number of final nodes for a test data set.

3. Alacart Example.

A business application requires that customers be classified into the categories 'good' and 'bad' [6]. The data set contains customer history, such as total purchases, total payments, number of times the customer was contacted or initiated a contact, etc. Bad customers are those who tend to be late payers, determined by business analysts. Missing values were imputed using the method described in Auslender (1997).

3.1 Training Tree.

For the purpose of modeling, we split the data set into training and test data sets. The training data set contains 12504 observations, 65.4% of which are 'bad' customers. The test data set contains 8193 observations, 64.7% of which are 'bad' customers. There are 22 independent variables. The training data set is used to generate the largest tree, which is presented in appendix I. The test data set is used at the pruning stage, and generates the pruned tree, of appendix II.

The largest tree contained 21 final nodes. Let us describe the leftmost branch. The variable "current balance due" splits the root node at 105.38. Those customers whose balances are less than 105.38 go 'left' while those with more than 105.38 go 'right'. The 'lefties' are then split on "past due balances" at 90.36, and finally again split by "current due balances" at 12. The resulting customers are classified as "good". If we summarize the branch, those customers whose 'current due balance' is at most $12, and the 'past due balance' is $90.36 are considered to be 'good' customers. We can observe that the variable 'current due' reappears. The final nodes contain the labels "G" and "B", denoting that those nodes represent customers classified as either "Good" or "Bad".

3.2 Pruned Tree.

As can be seen from comparing appendices I and II, the pruned tree is a subset of the larger tree. The pruned tree contains 8 final nodes, which was derived from the next table.

The table shows how the misclassification rate changes along the sequence of pruned subtrees, scored with the training and the test subsets. The minimum misclassification rate is at 8 nodes, which produces the tree graphed in appendix II.

The interested reader should note that the change in misclassification rates is so 'smooth' that pruning at say, 6 nodes, for reasons of parsimony should not be ruled out. Were this pruned tree applied to other datasets for the same application, we would expect a 28% misclassification rate on average. The prudent analyst should bear in mind the model interpretability, predictive power, generalization and business needs and practices when considering the pruning and subsequent implementation of a tree-based model. It is worthy to mention the need to recalibrate and remodel periodically

[6] *Due to business considerations, we are not at liberty to divulge more information on the specifics of the application.*

when data updates and/or brighter models are forthcoming

% MISSCL. RATES, MIN AT 8	DEV - TRAINING DATA SET	TEST DATA SET
# FINAL NODES		
21	27.33	28.32
20	27.34	28.29
18	27.37	28.28
17	27.38	28.24
16	27.40	28.24
15	27.42	28.30
14	27.45	28.32
13	27.48	28.35
12	27.51	28.26
10	27.60	28.27
8	27.70	28.23
7	27.77	28.27
6	27.85	28.33
4	28.65	29.13
3	29.64	30.50
2	30.81	31.88
1	34.60	35.31

The (partial) 'scoring' code produced for the 8 nodes pruned tree is presented below. Scoring proceeds by 'including' the score (stored as an ALGOR8.PGM) in a data step. Note that the sentences "% Node Impurity", "Branch #" and "Node Freq" are commented out but they could be activated during the scoring run to add more information. The curious readers might want to follow the code below and compare it with appendix II.

```
/* PROGRAM ALGOR8.PGM WITH 8 FINAL NODES*/
/* METHOD MISSCL ALACART TEST */

RETAIN ROOT 1;
IF ROOT &  CURRDUE <= 105.38 &  PASTDUE <= 90.36 &  CURRDUE <= 12
THEN DO;
      NODE        =  '4_1 ';
      PRED   =  0  ;
  /*  % NODE IMPURITY = 0.0399              ; */
```

```
  /*  BRANCH #   = 1     ; */
  /*  NODE FREQ  = 81            ; */
END;
ELSE IF ROOT &  CURRDUE <= 105.38 &  PASTDUE <= 90.36 &  CURRDUE >  12
THEN DO;
      NODE        =  '4_2 ';
      PRED   =  1  ;
  /*  % NODE IMPURITY = 0.4478              ; */
  /*  BRANCH #   = 2     ; */
  /*  NODE FREQ  = 212           ; */
END;
ELSE IF ROOT &  CURRDUE <= 105.38 &  PASTDUE >  90.36
THEN DO;
      NODE        =  '3_2 ';
      PRED   =  0  ;
…….
```

3.3 Model assessment.

While the overall model assessment may be glimpsed from the misclassification rate, given the graphical representation and consequent 'easy' interpretation, it is almost certain that specific final nodes will deserve further scrutiny. Thus, we assess node behavior with the following (partial) table, titled **"Prediction Diagnostics by Node"** which refers to the 'training' tree of appendix I.

The node numbering employed has two parts. For instance, '4_2' refers to the node in appendix I located in the 4th level, and the second from the left. Thus, it is the node that is derived from: 'if currdue <= 105.38 & pastdue <= 90.36 & currdue > 12'. Notice that the node 4_3 is an intermediate node, and thus it would not appear in a complete table.

The predictive accuracy for the training (or development) data set at node 4_2 was 73.58%, while that for the test data set was slightly higher. Since 'training' and 'testing' information are similar for this node, it might be inferred that this node is stable, although further diagnostics (beyond the scope of this paper) should be run.

PRED DIAGN BY NODE		TYPE					
		DEVLOPMNT		TESTING		COMBINED	
		# OBS		# OBS		# OBS	
			% NODE OBS		% NODE OBS		% NODE OBS
4_1	PRED ACCURACY						
	BAD PRED	5	6.17	1	1.96	6	4.55
	GOOD PRED	76	93.83	50	98.04	126	95.45
	COMBINED	81	100.00	51	100.00	132	100.00
4_2	PRED ACCURACY						
	BAD PRED	56	26.42	36	26.09	92	26.29
	GOOD PRED	156	73.58	102	73.91	258	73.71
	COMBINED	212	100.00	138	100.00	350	100.00

4. Alternative splits: Gini.

Breiman et al. considered other splitting criteria, and recommend the use of the Gini criterion for binary dependent variables. While we will not detail the corresponding argument, we have produced a new set of trees based on the Gini criterion, presented in appendices III and IV.

While both the Misclassification and Gini Trees start with the same split of "current due balances" at 105.38, ensuing splits differ and the overall shape of the trees is quite different. The number of final nodes is 15 for training, and 10 for the pruned version. The misclassification rate was 30.68 for the test data set at 10 final nodes.

The structure of Alacart allows for the easy addition of different splitting methods, such as Chaid, or Loh's approach (Loh and Vanichsetakul, 1988).

5. Conclusions.

This paper has presented in a 'lay and non-exhaustive manner' the topic of methodology, programming and application of classification trees. We have also indicated the possibility of researching alternative methodologies within the tree methodology, to better solve specific problems. For instance, Mathsoft S+ software minimizes the deviance. The Chaid methodology (Kass, 1980) produces splits based on chi-square inference.

6. Bibliography.

Auslender, L.: Missing Value Imputation Method for Large Databases, in *Proceedings of the 1997 Conference of the North Eastern SAS Users Group,* Baltimore, Md.

Breiman L., Friedman J., Olshen R., Stone C., *Classification and Regression Trees*, Wadsworth International Group, Belmont, Calif., 1984.

Clark L., Pregibon D., *Tree-based Models*, in Chambers, J., Hastie T. Eds. *Statistical Models in S*, Chapman & Hall, 1992.

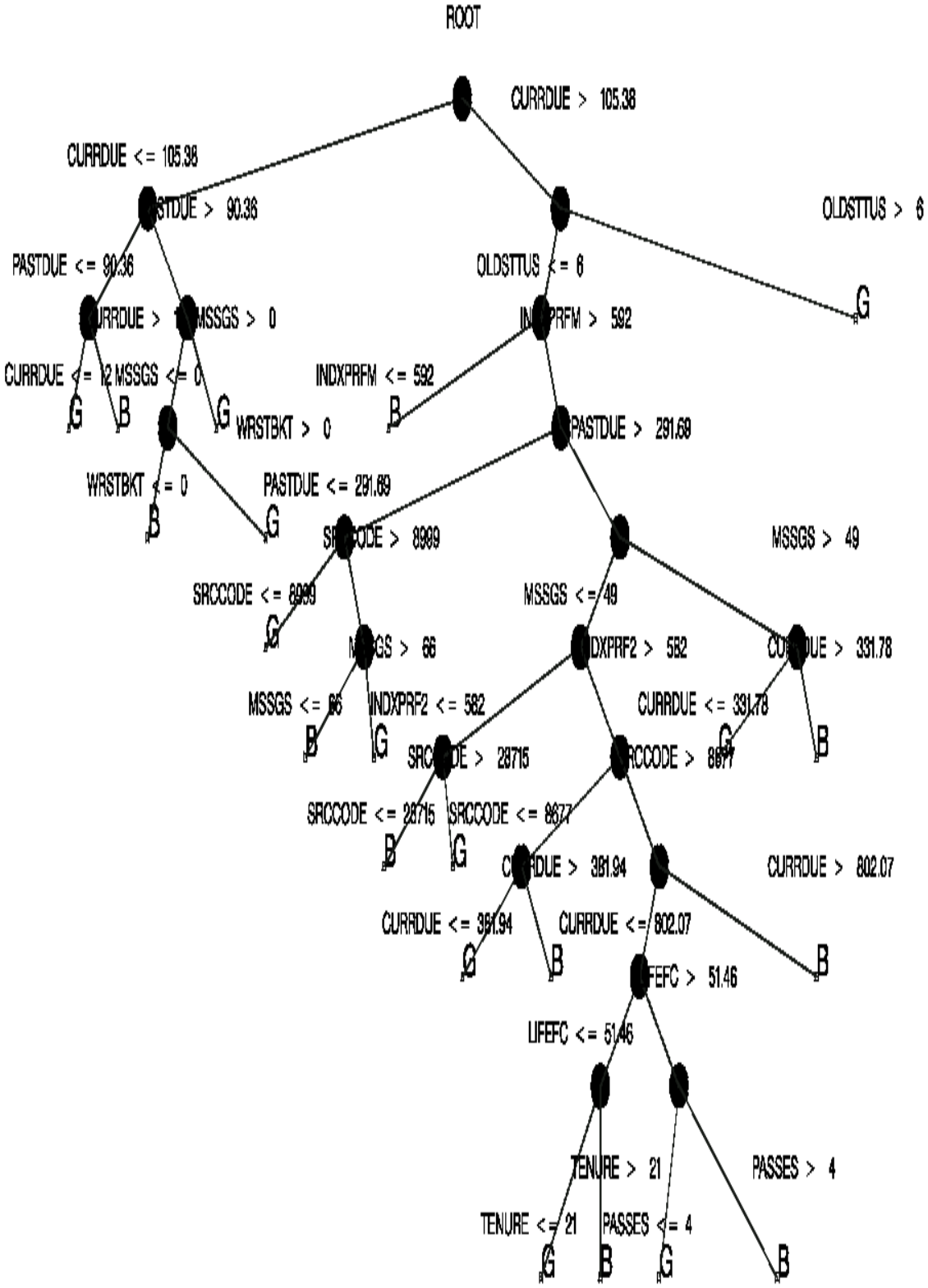
Kass G. V.: *An exploratory technique for investigating large quantities of categorical data",* Applied Statistics, 29, 119-127, 1980.

Loh W., Vanichsetakul N., Tree-structured classification *via generalized discriminant analysis*, Journal of the American Statistical Association, Sept. 1988.

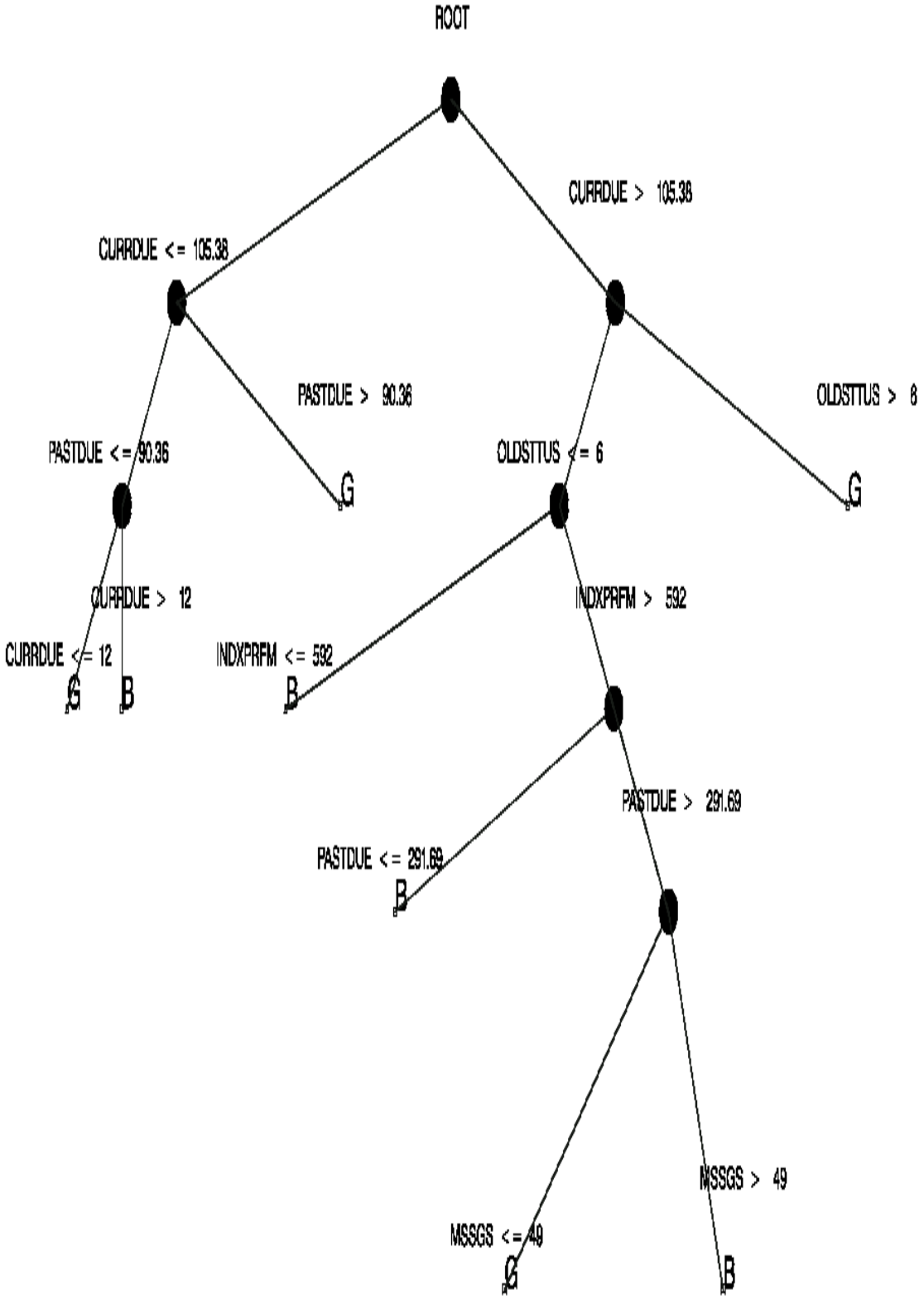
Quinlan J. R.: *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

Steinberg D., Kutner S., *Practical Aspects of Tree-Structured Data Analysis: Model Building Strategies for Classification Problems*, American Statistical Association, 1993 Proceedings of the Statistical computing Section.

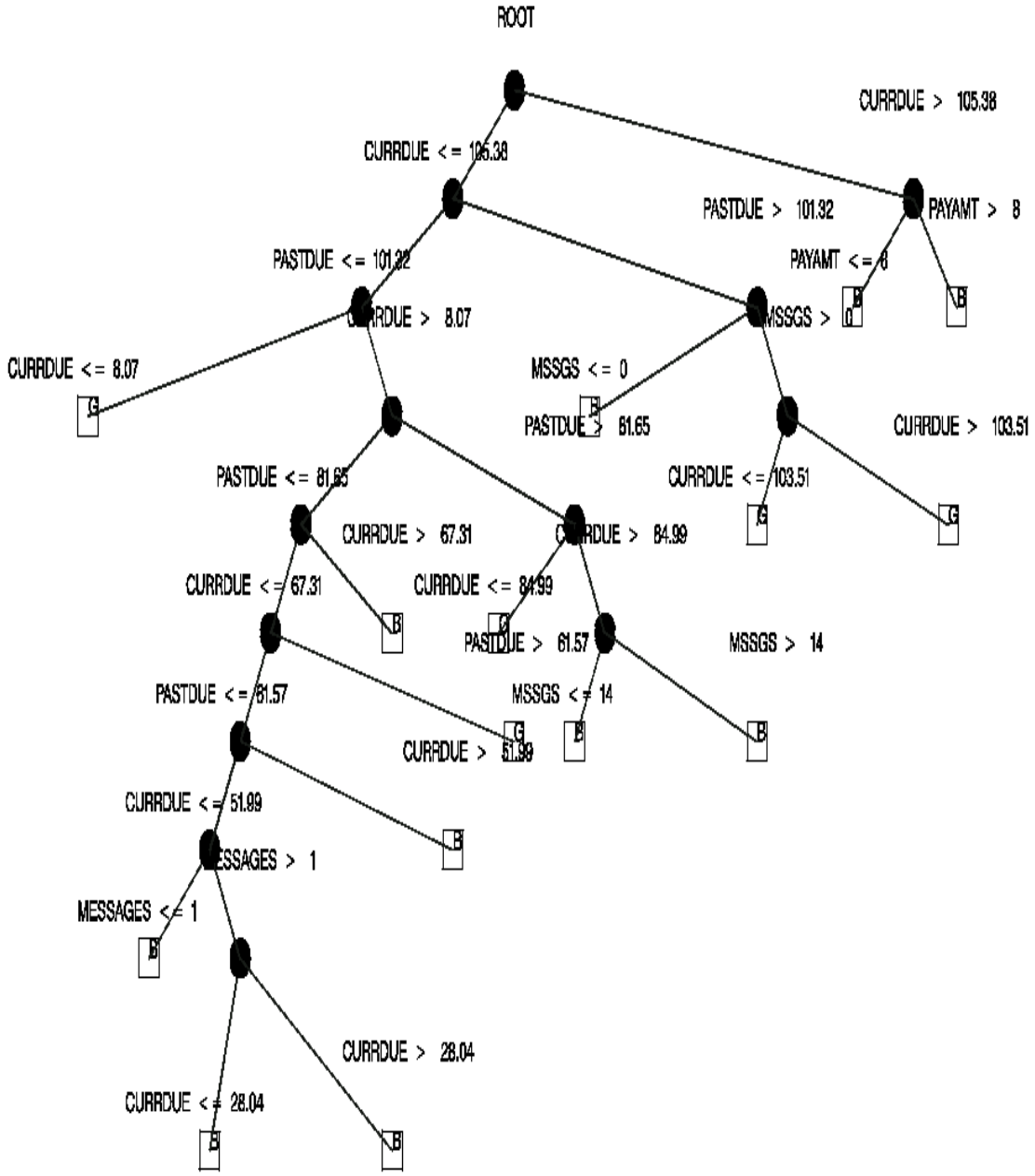
APPENDIX I: MISCL METHOD – LARGEST TREE CREATED



APPENDIX II: MISCL METHOD – PRUNED TREE



APPENDIX III: GINI METHOD – LARGEST TREE CREATED



APPENDIX IV: GINI METHOD – PRUNED TREE