CART

Applied Statistics And Programming (ASAP)
Seminar Series
Session 1

September 5, 2002
(/project/consulting/ASAP/SESS1/)

Table of contents

Please make
2 copies
Thanks
Jessl

Introduction to R

(http://www.R-project.org)

- R was initially written by Robert Gentleman and Ross Ihaka — also known as ``R & R" — of the Statistics Department of the University of Auckland.

- R can be regarded as an implementation of S/Splus, or a free version of S/Splus. Documentation for S/Splus can typically be used with R.

- Difference between R and OUR Splus (windows or unix)
  R – free, easy to update, can manage huge data set, ugly graphics window, no trillis plots
  S+ -- outdated, bugs in linear mixed effect models in both windows and unix version, excellent graphical tools

- Difference in philosophy between S/R and SAS
  S/R – minimal output, intermediate results stored in objects for subsequent interrogation by further functions.
  SAS – pages of output, intermediate results output to data sets for further 'proc' or 'data' steps.

- Most classical statistics and much of the latest methodology is available for use with R. People tend to develop new methodology in R, e.g. Berkeley statisticians develop microarray analytical tools in R.

- R on CCEB machines – only UNIX version, interactive or batch mode

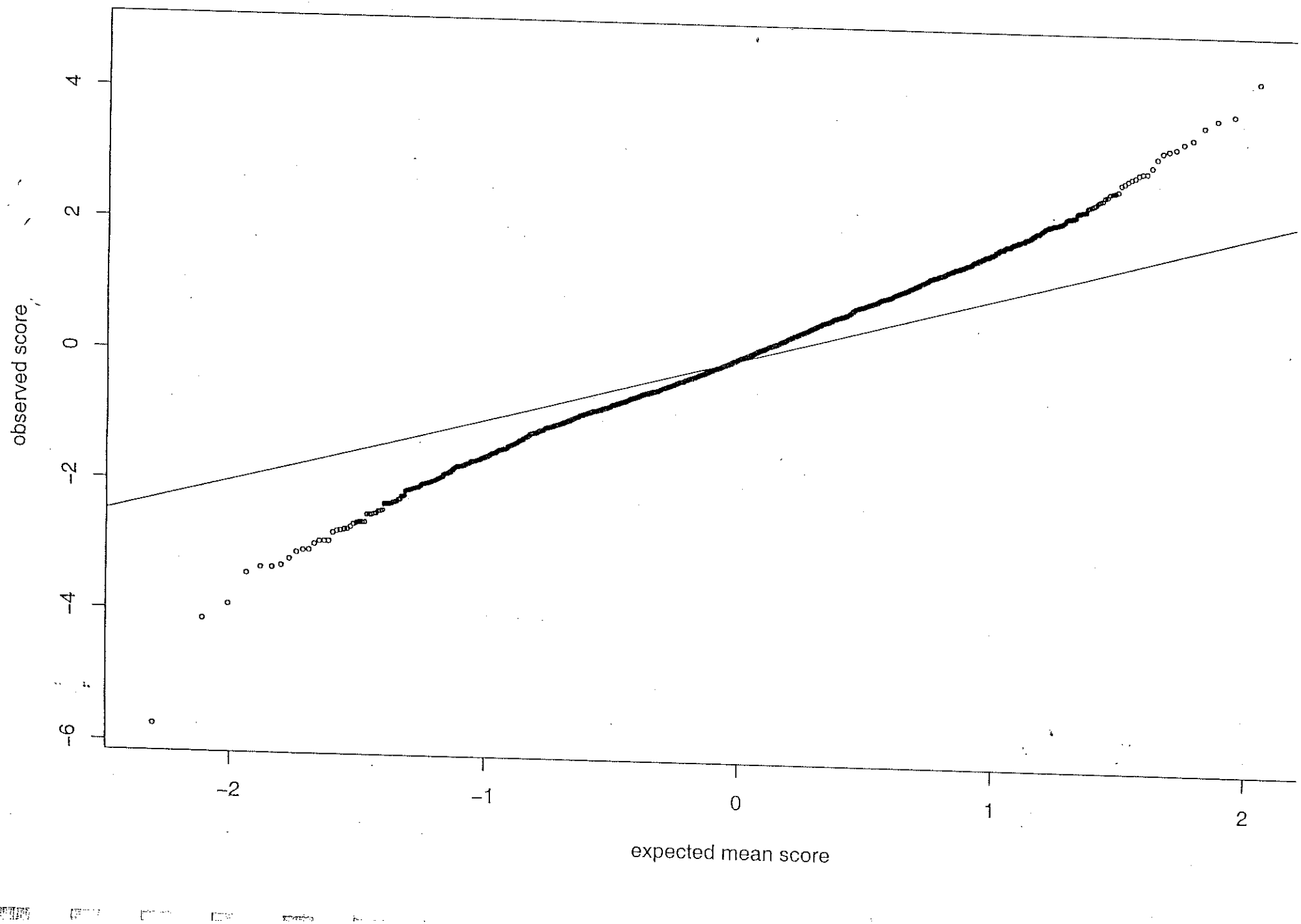
- To import data from other system, load in package 'foreign' using command
  Library(foreign)
  Function read.xport reads file in SAS Transport format and return a list of data frames

- In R, to fit linear mixed effects models, load in package 'nlme' using command library(nlme)

- Good Splus book: William N. Venables and Brian D. Ripley. Modern Applied Statistics with S-Plus. Third Edition. Springer, 1999. ISBN 0-387-98825-4.

File 1: sam.r

```
#############################################################################
#Purpose: Implement SAM (by Chu, et. al.) in R with statistic beta
#        to get FDR (false discovery rate)
#Note: beta.hat ~ r_i, beta.se ~ s_i
#Input Data: test statistics and its permutation
# (betas.obs and betas.permute by Mary 5/13/02)
#Output: FDR
#Last Updated: 5/30/02     by: Lan Zhou
#############################################################################

sam_function(beta.obs, beta.sd.obs, beta.perm, beta.sd.perm,
       genename=1:length(beta.obs), d0=seq(0,2, by=0.1)){
#beta.obs: observed statistics, dim ngene x 1 (1 beta for each gene)
#beta.sd.obs: sd for beta.obs, dim ngene x 1 (1 beta.sd for each gene)
#beta.perm: permutated beta, dim ngene x nperm
#beta.sd.perm: permuted beta.sd, dim ngene x nperm
#genename: names of genes
#d0: threshold

  #create di and di.perm
  obsdata_c(beta.obs, beta.sd.obs)
  di.obs_mkdi(obsdata, n=length(beta.obs))
#  di.perm_beta.perm
#  for(i in 1:ncol(beta.perm)) di.perm[,i]_mkdi(beta.perm[,i],beta.sd.perm[,i])
  permdata_rbind(beta.perm, beta.sd.perm) #put the data together for mkdi
  di.perm_apply(permdata, 2, mkdi, n=length(beta.obs))

#browser()
  save(di.obs, file="/project/arrays/Rdata/di.obs.Rdata")
  save(di.perm, file="/project/arrays/Rdata/di.perm.Rdata")

  #sort di.obs from min to max, save the order
  order.di.obs_order(di.obs)

  #sort di.perm by each permutation, row1=min, row_ngene=max
  di.perm.sort_apply(di.perm, 2, sort)
  #get expectation using means and medians by row
  di.perm.mean_apply(di.perm.sort, 1, mean) #ngenex1
  di.perm.median_apply(di.perm.sort, 1, median) #ngenex1

  #plot di.obs.sort versus di.perm.mean/di.perm.median
  #save the plot as ps file in Rplot, use ghostview to read
  postscript("/project/arrays/Rplot/betas.samps", horizontal=T)
  plot(di.perm.mean, di.obs[order.di.obs], xlab='expected mean score',
       ylab='observed score', type='p', cex=0.5)
  abline(0,1)

#  plot(di.perm.median, di.obs[order.di.obs], xlab='expected median score',
#      ylab='observed score', type='p', cex=0.7)

  #lookat difference between di.obs_ordered and di.perm.mean and di.perm.median
  diff.mean_di.obs[order.di.obs]-di.perm.mean
  diff.median_di.obs[order.di.obs]-di.perm.median

  #pick up significant genes and calcuate FDR based on each d0
  fdr_matrix(NA, nrow=3, ncol=length(d0)) #mean, 50%, 95%
  out_as.list(rep(NA, length(d0)))

  for(i in 1:length(d0)) {
    sig.pos_diff.mean >= d0[i]
    sig.neg_diff.mean < -d0[i]
```


observed score
4
2
0
−2
−4
−6
−2
−1
0
1
2
expected mean score

Appendix B Invoking R

B.1 Invoking R under UNIX

The command 'R' can be used both for starting the main R program with a variety of options in the form

R [*options*] [<*infile*] [>*outfile*],

or, via the R CMD interface, as a wrapper to various R tools (e.g., for processing files in R documentation format or manipulating add-on packages) which are not intended to be called "directly".

Most options control what happens at the beginning and at the end of an R session. The startup mechanism is as follows (see also the on-line help for topic 'Startup' for more information).

- Unless '--no-environ' was given, R searches for the file '.Renviron' in the current directory, or failing that the file pointed to by the environment variable R_ENVIRON, or if that is not set, the file '.Renviron' in the user's home directory. The first of these files found is processed to set environment variables. It should contain lines of the form '*name=value*'. (See help(Startup) for a precise description.) Variables you might want to set include R_PAPERSIZE (the default paper size), R_PRINTCMD (the default print command) and R_LIBS (specifies the list of R library trees searched for add-on packages).
- Then R searches for the site-wide startup profile unless the command line option '--no-site-file' was given. The name of this file is taken from the value of the R_PROFILE environment variable. If that variable is unset, the default is '$R_HOME/etc/Rprofile'.
- Then, unless '--no-init-file' was given, R searches for a file called '.Rprofile' in the current directory or in the user's home directory (in that order) and sources it.
- It also loads a saved image from '.RData' if there is one (unless '--no-restore' or '--no-restore-data' was specified).
- Finally, if a function .First exists, it is executed. This function (as well as .Last which is executed at the end of the R session) can be defined in the appropriate startup profiles, or reside in '.RData'.

In addition, there are options for controlling the memory available to the R process (see the on-line help for topic 'Memory' for more information). Users will not normally need to use these unless they are trying to limit the amount of memory used by R.

R accepts the following command-line options.

'--help'
'-h' Print short help message to standard output and exit successfully.

'--version'
Print version information to standard output and exit successfully.

'RHOME' Print the path to the R "home directory" to standard output and exit successfully. Apart from the front-end shell script and the man page, R installation puts everything (executables, packages, etc.) into this directory.

'--save'
'--no-save'
: Control whether data sets should be saved or not at the end of the R session. If neither is given in an interactive session, the user is asked for the desired behavior when ending the session with `q()`; in batch mode, one of these must be specified.

'--no-environ'
: Do not read any user file to set environment variables.

'--no-site-file'
: Do not read the site-wide profile at startup.

'--no-init-file'
: Do not read the user's profile at startup.

'--restore'
'--no-restore'
'--no-restore-data'
: Control whether saved images (file '.RData' in the directory where R was started) should be restored at startup or not. The default is to restore. ('--no-restore' implies all the specific '--no-restore-*' options.)

'--no-restore-history'
: Control whether the history file (normally file '.Rhistory' in the directory where R was started, but can be set by the environment variable R_HISTFILE) should be restored at startup or not. The default is to restore.

'--vanilla'
: Combine '--no-save', '--no-environ' '--no-site-file', '--no-init-file' and '--no-restore'.

'--no-readline'
: Turn off command-line editing via readline. This is useful when running R from within Emacs using the ESS ("Emacs Speaks Statistics") package. See Appendix C [The command line editor], page 92, for more information.

'--min-vsize=*N*'
'--max-vsize=*N*'
: Specify the minimum or maximum amount of memory used for variable size objects by setting the "vector heap" size to *N* bytes. Here, *N* must either be an integer or an integer ending with 'M', 'K', or 'k', meaning 'Mega' (2^20), (computer) 'Kilo' (2^10), or regular 'kilo' (1000).

'--min-nsize=*N*'
'--max-nsize=*N*'
: Specify the amount of memory used for fixed size objects by setting the number of "cons cells" to *N*. See the previous option for details on *N*. A cons cell takes 28 bytes on a 32-bit machine, and usually 56 bytes on a 64-bit machine.

'--quiet'
'--silent'
'-q'
: Do not print out the initial copyright and welcome messages.

'--slave' Make R run as quietly as possible. This option is intended to support programs which use R to compute results for them.

'--verbose'
Print more information about progress, and in particular set R's option verbose to TRUE. R code uses this option to control the printing of diagnostic messages.

'--debugger=*name*'
'-d *name*' Run R through debugger *name*. Note that in this case, further command line options are disregarded, and should instead be given when starting the R executable from inside the debugger.

'--gui=*type*'
'-g *type*' Use *type* as graphical user interface (note that this also includes interactive graphics). Currently, possible values for *type* are 'X11' (the default), 'gnome' provided that GNOME support is available, and 'none'.

Note that input and output can be redirected in the usual way (using '<' and '>').

The command R CMD allows the invokation of various tools which are useful in conjunction with R, but not intended to be called "directly". The general form is

R CMD *command args*

where *command* is the name of the tool and *args* the arguments passed on to it.

Currently, the following tools are available.

BATCH	Run R in batch mode.
COMPILE	Compile files for use with R.
SHLIB	Build shared library for dynamic loading.
INSTALL	Install add-on packages.
REMOVE	Remove add-on packages.
build	Build add-on packages.
check	Check add-on packages.
LINK	Front-end for creating executable programs.
Rprof	Post-process R profiling files.
Rdconv	Convert Rd format to various other formats, including HTML, Nroff, LaTeX, plain text, and S documentation format.
Rd2dvi	Convert Rd format to DVI/PDF.
Rd2txt	Convert Rd format to text.
Rdindex	Extract index information from Rd files.
Sd2Rd	Convert S documentation to Rd format.

The first five tools (i.e., BATCH, COMPILE, SHLIB, INSTALL, and REMOVE) can also be invoked "directly" without the CMD option, i.e., in the form R *command args*.

Use

R CMD *command* --help

to obtain usage information for each of the tools accessible via the R CMD interface.

3 Importing from other statistical systems

In this chapter we consider the problem of reading a binary data file written by another statistical system. This is often best avoided, but may be unavoidable if the originating system is not available.

3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata

The recommended package **foreign** provides import facilities for files produced by these statistical systems, and for export to Stata. In some cases these function may require substantially less memory than would `read.table`.

Stata '`.dta`' files are a binary file format. Files from versions 5.0, 6.0 and 7.0 of Stata can be read and written by functions `read.dta` and `write.dta`. Stata variables with value labels are optionally converted to (and from) R factor.

EpiInfo versions 5 and 6 stored data in a self-describing fixed-width text format. `read.epiinfo` will read these `.REC` files into an R data frame.

Function `read.mtp` imports a 'Minitab Portable Worksheet'. This returns the components of the worksheet as an R list.

Function `read.xport` reads a file in SAS Transport (XPORT) format and return a list of data frames. If SAS is available on your system, function `read.ssd` can be used to create and run a SAS script that saves a SAS permanent dataset (`.ssd` or `.sas7bdat`) in Transport format. It then calls `read.xport` to read the resulting file.

Function `read.spss` can read files created by the 'save' and 'export' commands in SPSS. It returns a list with one component for each variable in the saved data set. SPSS variables with value labels are optionally converted to R factors.

Function `read.S` which can read binary objects produced by S-PLUS 3.x, 4.x or 2000 on (32-bit) Unix or Windows (and can read them on a different OS). This is able to read many but not all S objects: in particular it can read vectors, matrices and data frames and lists containing those.

Function `data.restore` to read S-PLUS data dumps (created by `data.dump`) with the same restrictions (except that dumps from the Alpha platform can also be read). It should be possible to read data dumps from S-PLUS 5.x and 6.x written with `data.dump(oldStyle=T)`.

3.2 Octave

Octave is a numerical linear algebra system, and function `read.octave` in package **e1071** can read the first vector or matrix from an Octave ASCII data file created using the Octave command `save -ascii`.

Ps 10080

AN OVERVIEW OF THE CART METHODOLOGY

Introduction

CART® is a robust data-mining and data-analysis tool that automatically searches for important patterns and relationships and quickly uncovers hidden structure even in highly complex data. This discovered knowledge is then used to generate accurate and reliable predictive models for applications such as profiling customers, targeting direct mailings, detecting telecommunications and credit-card fraud, and managing credit risk.

CART uses an intuitive Windows-based interface, making it accessible to both technical and non-technical users. Underlying the "easy" interface, however, is a mature theoretical foundation that distinguishes CART from other methodologies and other decision-tree tools.

The CART methodology is technically known as binary recursive partitioning. The process is binary because parent nodes are always split into exactly two child nodes and recursive because the process can be repeated by treating each child node as a parent. The key elements of a CART analysis are a set of rules for:

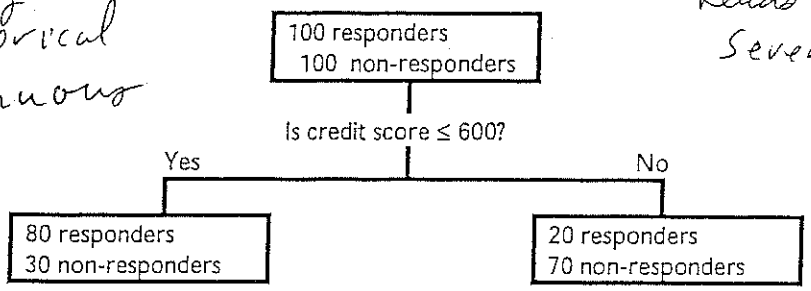
- splitting each node in a tree,
- deciding when a tree is complete, and
- assigning each terminal node to a class outcome (or predicted value for regression).

Splitting Rules

To split a node into two child nodes, CART always asks questions that have a "yes" or "no" answer. For example, the question "Is credit score ≤ 600?" splits the tree's root, or parent, node into two branches with "yes" cases going to the left child node and "no" cases to the right.

binary
categorical
continuous

Reads SAS & Several other formats



How do we come up with candidate splitting rules? CART's method is to look at all possible splits for all variables included in the analysis. For example, consider a data set with 215 cases and 19 variables. CART considers up to 215 times 19 splits for a total of 4085 possible splits. Any problem will have a finite number of candidate splits and CART will conduct a brute force search through them all.

Choosing a Split

CART's next activity is to rank order each splitting rule on the basis of a quality-of-split criterion. The default criterion used in CART is the GINI rule, essentially a measure of how well the splitting rule separates the classes contained in the parent node. Five alternative criteria are also available for classification trees and two criteria for regression trees. In addition, to deal more effectively with select data patterns, CART also offers splits on linear combination of continuous predictor variables.

Class Assignment

Once a best split is found, CART repeats the search process for each child node, continuing recursively until further splitting is impossible or stopped. Splitting is impossible if only one case remains in a particular node or if all the cases in that node are exact copies of each other (on predictor variables). CART also allows splitting to be stopped for several other reasons, including that a node has too few cases. (The default for this lower limit is 10 cases, but may be set higher or lower to suit a particular analysis).

Once a terminal node is found we must decide how to classify all cases falling within it. One simple criterion is the plurality rule: the group with the greatest representation determines the class assignment (*e.g.*, in the sample decision tree above, the case assignment for the left child node is responders and non-responders for the right child node.). CART goes a step further: because each node has the potential for being a terminal node, a class assignment is made for every node whether it is terminal or not. The rules of class assignment can be modified from simple plurality to account for the costs of making a mistake in classification and to adjust for over- or under-sampling from certain classes.

A common technique among the first generation of tree classifiers was to continue splitting nodes (growing the tree) until some goodness-of-split criterion failed to be met. When the quality of a particular split fell below a certain threshold, the tree was not grown further along that branch. When all branches from the root reached terminal nodes, the tree was considered complete. While this technique is still embodied in several commercial programs, including CHAID™ and KnowledgeSEEKER™, it often yields erroneous results. CART uses a completely different technique.

Pruning Trees

Instead of attempting to decide whether a given node is terminal or not, CART proceeds by growing trees until it is not possible to grow them any further. Once CART has generated a "maximal tree," it examines smaller trees obtained by pruning away branches of the maximal tree. Unlike other methods, CART does not stop in the middle of the tree-growing process, because there might still be important information to be discovered by drilling down several more levels.

Testing

Once the maximal tree is grown and a set of sub-trees are derived from it, CART determines the best tree by testing for error rates or costs. With sufficient data, the simplest method is to divide the sample into learning and test sub-samples. The learning sample is used to grow an overly-large tree. The test sample is then used to estimate the rate at which cases are misclassified (possibly

CART Example 1.

Objective: Develop a prediction model that will allow physicians to predict whether a patient is having an ectopic pregnancy (ECT) or spontaneous abortion (SAB) vs. a normal pregnancy (IUP), with very high specificity (the proportion of individuals without the disorder classified as not having the disorder).

OUTCOME: ECT&SAB vs. IUP (predicting ECT&SAB)

LIMIT: minimum size below which a node will not be split is 40.

MODEL: ECT_SAB= bleed+pain+age+numsab+quantatp+parity+pid+infectio, outpatie+numvip+numliveb+chlam+gc+numectop+outpatie+hxsurg, pastusei+curr_inf+numcs+gravidit

CATEGORICAL VARIABLES: ECT_SAB,infectio,chlam,gc,hxsurg,curr_inf

Add weights in order to maximize specificity

Misclassify Cost = 1 Classify 1 as 0 (ECT&SAB as IUP)

Misclassify Cost = 10 Classify 0 as 1 (IUP as ECT&SAB)

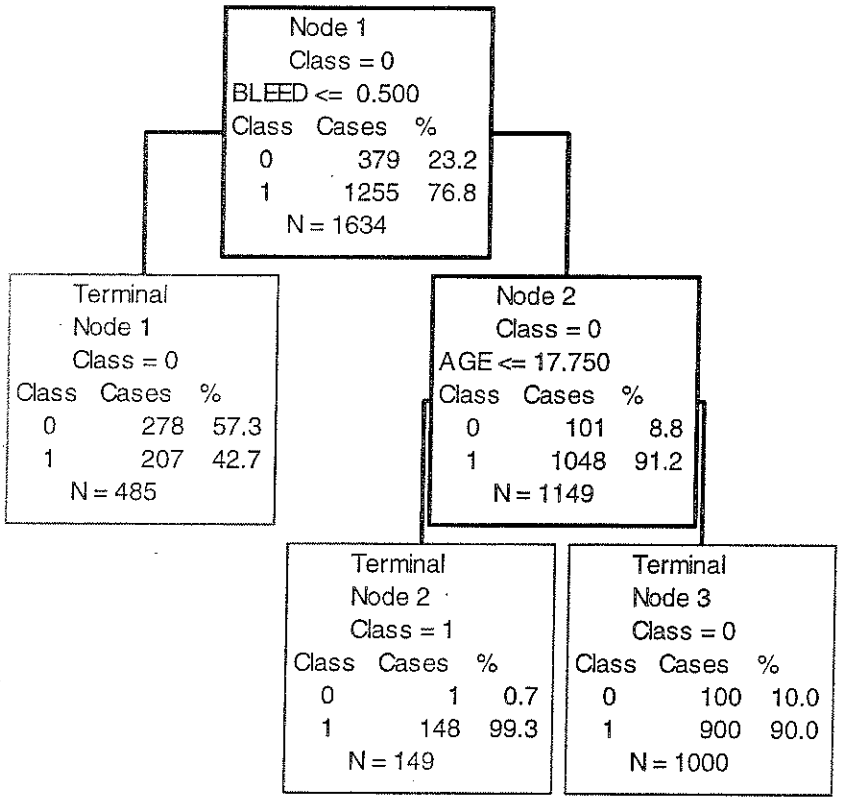
Actual CART syntax:

```
LOPTIONS MEANS = YES, NOPRINT = NO, PREDICTIONS = YES/BOTH, TIMING = YES,
PLOTS,
= YES
FORMAT = 3/UNDERFLOW

USE "C:\kbarnhart\biostats\datasets\rawdata\SAS\riskfixnm.sas7bdat"
CATEGORY ECT_SAB,infectio,chlam,gc,hxsurg,curr_inf
MODEL ECT_SAB= bleed+pain+age+numsab+quantatp+parity+pid+infectio,
outpatie+numvip+numliveb+chlam+gc+numectop+outpatie+hxsurg,
pastusei+curr_inf+numcs+gravidit
Misclass UNI
Misclassify Cost = 1 Classify 1 as 0
Misclassify Cost = 10 Classify 0 as 1
ERROR CROSS = 1
METHOD GINI POWER =  0.0000
limit atom=40, minchild=20
BUILD
```

==========

TREE SEQUENCE

==========



CART Example 2.

Objective: Develop a prediction model that will allow physicians to predict whether a patient is having an ectopic pregnancy (ECT) or spontaneous abortion (SAB) vs. a normal pregnancy (IUP), with very high sensitivity (the proportion of individuals with the disorder classified as having the disorder).

OUTCOME: ECT&SAB vs. IUP (predicting ECT&SAB)
LIMIT: minimum size below which a node will not be split is 40.
MODEL: ECT_SAB= bleed+pain+age+numsab+quantatp+parity+pid+infectio, outpatie+numvip+numliveb+chlam+gc+numectop+outpatie+hxsurg, pastusei+curr_inf+numcs+gravidit
CATEGORICAL VARIABLES: ECT_SAB,infectio,chlam,gc,hxsurg,curr_inf

Add weights in order to maximize sensitivity
Misclassify Cost = 10 Classify 1 as 0 (ECT&SAB as IUP)
Misclassify Cost = 1 Classify 0 as 1 (IUP as ECT&SAB)

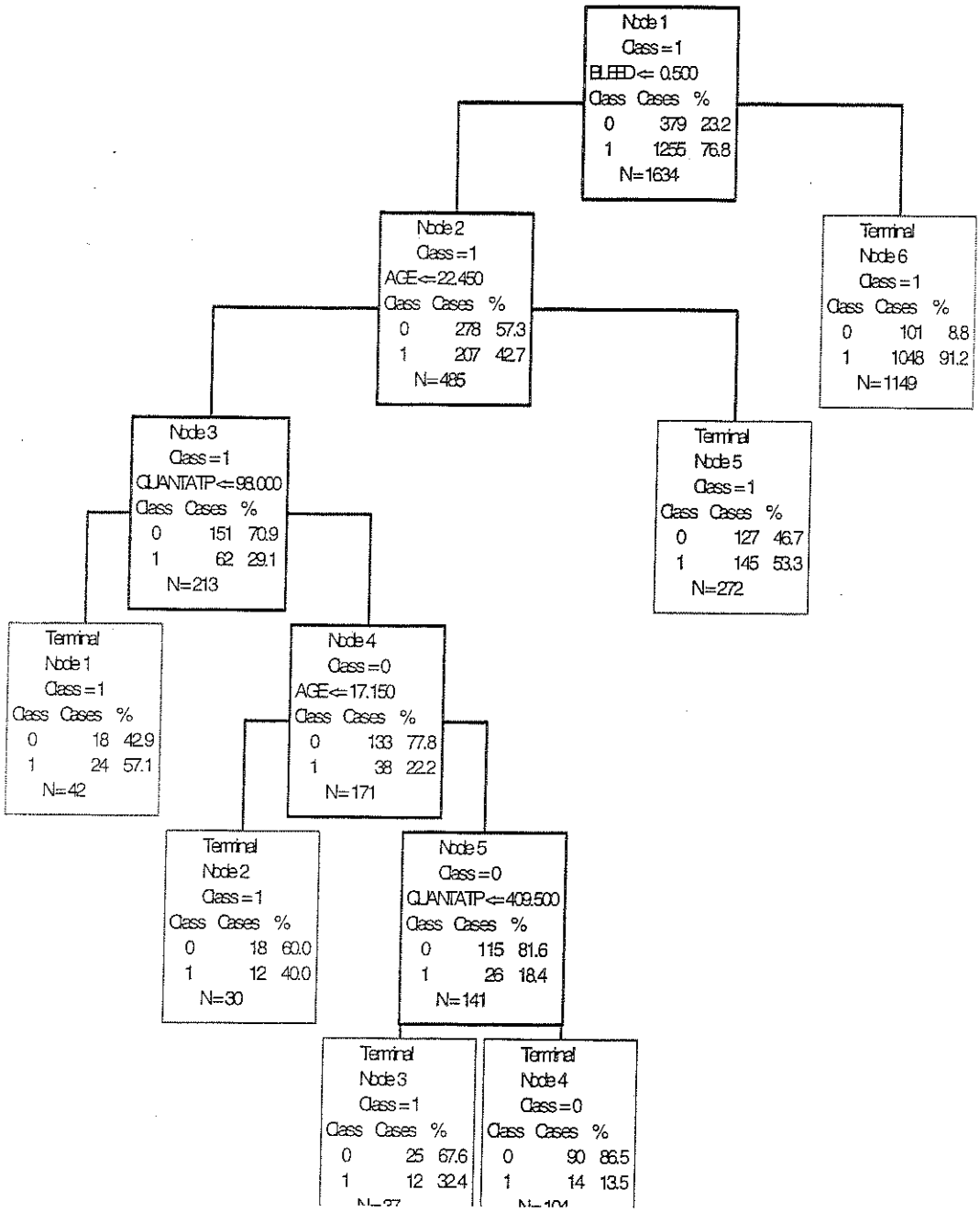
Actual CART syntax:

```
LOPTIONS MEANS = YES, NOPRINT = NO, PREDICTIONS = YES/BOTH, TIMING = YES,
PLOTS,
= YES
FORMAT = 3/UNDERFLOW

USE "C:\kbarnhart\biostats\datasets\rawdata\SAS\riskfixnm.sas7bdat"
CATEGORY ECT_SAB,infectio,chlam,gc,hxsurg,curr_inf
MODEL ECT_SAB= bleed+pain+age+numsab+quantatp+parity+pid+infectio,
outpatie+numvip+numliveb+chlam+gc+numectop+outpatie+hxsurg,
pastusei+curr_inf+numcs+gravidit
Misclass UNI
Misclassify Cost = 10 Classify 1 as 0
Misclassify Cost = 1 Classify 0 as 1
ERROR CROSS = 10
METHOD GINI POWER =  0.0000
limit atom=40, minchild=20
BUILD
```

==========

TREE SEQUENCE

==========



Based on the results from the CART analyses we created the following three-level variable:

1 = Send home if the patient is not bleeding, has a quantatp score greater than 400 and is between the age of 18 and 25(inclusively).

3 = Intervene if the patient is bleeding and is less than 18 years old.

2 = Otherwise, keep the patient under observation.

This is the actual SAS code:

```
* create the following categorical variable based on output from CART
6/24/02;
if bleed ne . and age ne . and quantatp ne . then do;
 if bleed = 0 and 18 =< age =< 25 and quantatp > 400 then cart1=1;
 else if bleed = 1 and . < age =< 18 then cart1 = 3; else cart1 = 2;
end;
```

The Steps For Developing a Parametric Prediction Model for the Ectopic Pregnancy Study

I. Crude summary of all potential predictor variables:
   a. Correlated all categorical variables with outcome using cross-tabulations.
   b. For all continuous variables we generated: means, medians, min, max, n, and histograms for each level of the outcome variable.
   c. We created categorical versions of some of the continuous variables and binary versions of many of the multi-level categorical variables and repeated step (a) with them.

II. Univariate statistics such as odd ratios and p-values for the chi-square test of significance were generated to assess the significance of each individual predictor variable on outcome. Logistic regression was used.

III. The predictors were all sorted into one of two categories:
   a. Demographic or history variables
   b. Presentation/Management/treatment variables

IV. Within each category, any variable with a p-value less than 0.2 was entered into a semi-final model and we performed an iterative backward elimination procedure in which we ran a series of logistic regressions; At each step we removed the variable with the largest p-value over 0.05. We proceeded until there were no variables in the model with $p>0.05$. Note: Any variable that was identified as important based on literature remained in the model regardless of p-value.

V. All of the predictor variables that remained in the semi-final models were all combined into one final model to be scaled down using the same backward elimination criteria specified above. The variables that remained after this final backward elimination were considered the final parametric model variables.

VI. We ran a series of logistic regression models consisting of all the final model variables, each time adding one of the original potential predictor variables to look for possible confounding and/or make sure that we didn't miss an important predictor. Any variable that caused a greater than 10% change in the odds ratio of one of the final model variables was considered a confounder. Furthermore, if the p-value for the added variable was significant ($p<0.05$) we kept it in the model.

VII. Using the parameter estimates from the final model as coefficients, we developed a continuous scored variable as a linear function of the final model variables.

VIII. Using the scored variable as the sole predictor in a logistic regression with the same outcome variable, we developed a predication rule based on the sensitivity and specificity at each level of the scored variable.

On 5/23/2002 we decided to create the following scored variable based on the final variables from the ECT&SAB vs IUP model;

```
abnscore = 1*(age_lmh=1) + 3*(age_lmh=3) + 0*(numliveb) + 1*(numepc2=1)
      + 2*(numepc2=2) - 1*(numsabc2=1) + 0*(numsabc2=2) + 0*(parityc3)
      + 4*(bleed) - 1*(hcgcat3 = 3);
```

Obs	Scored variable based on final EPSAB vs IUP model	Probability Level	Sensitivity	1 - Specificity	SPEC
1	-2	0.1758	1.00000	1.00000	0.0000
2	-1	0.2947	0.99681	0.97625	0.0237
3	0	0.4501	0.95857	0.72296	0.2770
4	1	0.6159	0.87171	0.36675	0.6332
5	2	0.7586	0.85100	0.26913	0.7309
6	3	0.8602	0.82231	0.25066	0.7493
7	4	0.9234	0.61514	0.11873	0.8813
8	5	0.9594	0.14502	0.01847	0.9815
9	6	0.9789	0.03426	0.00792	0.9921
10	7	0.9891	0.01594	0.00264	0.9974
11	8	0.9944	0.00239	0.00000	1.0000

The FREQ Procedure

abnscore	abnscorec	Frequency
-2	Send home	13
-1	Send home	144
0	Monitor	250
1	Monitor	63
2	Monitor	44
3	Monitor	322
4	Monitor	666
5	Intervene	150
6	Intervene	26
7	Intervene	18
8	Intervene	3

Frequency Missing = 404

Patient Classification based on parametric model VS ACTUAL OUTCOME

The FREQ Procedure

Table of abnscorec by ect_sab

abnscorec(Patient Classification based on Model building) ect_sab

Frequency Row Pct Col Pct	IUP	EP&SAB	Total
Send home	105 66.88 27.70	52 33.12 4.14	157
Monitor	267 20.73 70.45	1021 79.27 81.35	1288
Intervene	7 3.70 1.85	182 96.30 14.50	189
Total	379	1255	1634

Frequency Missing = 65

Statistics for Table of abnscorec by ect_sab

Statistic	DF	Value	Prob
Chi-Square	2	212.8757	<.0001
Likelihood Ratio Chi-Square	2	196.0517	<.0001
Mantel-Haenszel Chi-Square	1	180.5225	<.0001
Phi Coefficient		0.3609	
Contingency Coefficient		0.3395	
Cramer's V		0.3609	

Effective Sample Size = 1634
Frequency Missing = 65

Patient Classification based on CART
VS
ACTUAL OUTCOME

The FREQ Procedure

Table of cart1 by ect_sab

cart1(Patient Classification based on CART)

Frequency Row Pct Col Pct	ect_sab IUP	 EP&SAB	 Total
Send home	93 75.61 24.54	30 24.39 2.39	123
Monitor	283 21.07 74.67	1060 78.93 84.46	1343
Intervene	3 1.79 0.79	165 98.21 13.15	168
Total	379	1255	1634

Frequency Missing = 65

Statistics for Table of cart1 by ect_sab

Statistic	DF	Value	Prob
Chi-Square	2	236.3072	<.0001
Likelihood Ratio Chi-Square	2	220.1911	<.0001
Mantel-Haenszel Chi-Square	1	195.3027	<.0001
Phi Coefficient		0.3803	
Contingency Coefficient		0.3555	
Cramer's V		0.3803	

Effective Sample Size = 1634
Frequency Missing = 65

Patient Classification based on parametric model VS ACTUAL OUTCOME

The FREQ Procedure

Table of abnscorec by outcome

abnscorec(Patient Classification based on Model building)

Frequency Row Pct Col Pct	ACTUAL outcome EP	IUP	SAB	Total
Send home	18 11.46 6.14	105 66.88 27.70	34 21.66 3.53	157
Monitor	246 19.10 83.96	267 20.73 70.45	775 60.17 80.56	1288
Intervene	29 15.34 9.90	7 3.70 1.85	153 80.95 15.90	189
Total	293	379	962	1634

Frequency Missing = 65

Statistics for Table of abnscorec by outcome

Statistic	DF	Value	Prob
Chi-Square	4	221.8018	<.0001
Likelihood Ratio Chi-Square	4	205.8246	<.0001
Mantel-Haenszel Chi-Square	1	43.4015	<.0001
Phi Coefficient		0.3684	
Contingency Coefficient		0.3457	
Cramer's V		0.2605	

Effective Sample Size = 1634
Frequency Missing = 65

Patient Classification based on CART
VS
ACTUAL OUTCOME

The FREQ Procedure

Table of cart1 by outcome

cart1(Patient Classification based on CART)
ACTUAL outcome

Frequency Row Pct Col Pct	EP	IUP	SAB	Total
Send home	9 7.32 3.07	93 75.61 24.54	21 17.07 2.18	123
Monitor	274 20.40 93.52	283 21.07 74.67	786 58.53 81.70	1343
Intervene	10 5.95 3.41	3 1.79 0.79	155 92.26 16.11	168
Total	293	379	962	1634

Frequency Missing = 65

Statistics for Table of cart1 by outcome

Statistic	DF	Value	Prob
Chi-Square	4	275.5836	<.0001
Likelihood Ratio Chi-Square	4	260.5808	<.0001
Mantel-Haenszel Chi-Square	1	75.4088	<.0001
Phi Coefficient		0.4107	
Contingency Coefficient		0.3799	
Cramer's V		0.2904	

Effective Sample Size = 1634
Frequency Missing = 65

Reference:

Steinberg, Dan and Phillip Colla. CART: Tree-Structured Non-Parametric Data Analysis. San Diego, CA: Salford Systems 1995.