Chapter 5

Logistic Regression

In the last chapter we provided an introduction to linear regression, the first predictive modeling tool created, and one that is still extremely powerful. Unfortunately, the method does have its limitations. One important one is that categorical target variables (variables with a nominal or ordinal scale) violate the assumptions underlying linear regression. However, in a pinch, a reasonable predictive model for a binary target variable (typically of the "Yes/No" variety) can be created using linear regression (assuming the values of "Yes" and "No" have been converted to values of 1 and 0). The same is true for an ordinal target variable, assuming the categories of the variable have been given numerical values that reflect their ordinal positions. Even though it is possible to create reasonable predictive models for binary and ordinal target variables, more appropriate methods exist.

One problem with using linear regression for a binary target variable is that what we are interested in predicting is the *probability* that a customer will respond in a favorable way (e.g., that his or her response will be "yes" to our offer). The predicted probability of a favorable response should fall between zero and one. However, using linear regression, we will often predict probabilities that fall outside the zero to one range.

Logistic regression is the preferred method for developing a regression-like predictive model when the target variable is binary. The method falls in the same broad class of methods (known as generalized linear models) as linear regression, and provides a similar set of outputs, so if you are comfortable reading a linear regression output, a logistic regression output is also easy to read. In this chapter we again begin by providing a graphical visualization of a one predictor problem along with the logistic regression approach, to build some intuition for the method, and then move on to explaining some of the technical details behind the method. The final section is devoted to a tutorial that covers visualization methods for examining the relationship between a binary target variable and several potential predictor variables, estimating logistic regression models, and assessing the model.

5.1 A Graphical Illustration of the Problem

To motivate the illustration, consider the case of a charity that has developed a frequent giving program in which donors make a small, pre-determined donation that is automatically debited against a donor's credit card each month. Since the amounts are small, this has a limited effect on the donor's monthly cash flow, allowing that donor to comfortably give more in total than is possible with a single annual donation. A trial program done by the charity reveals that people who do enroll in the frequent giving program do in fact give substantially more than they did before entering the program. The question now is whom, in the charity's database, to target with an offer to join this monthly giving program in order to increase the percentage of the donor base that are enrolled in the program?

A preliminary analysis done by the charity reveals that one important factor in determining whether a donor joins the frequent giving program is that donor's average annual donation amount prior to receiving an offer to join the frequent giving program. Figures 5.1 and 5.2 provide a scatterplot of the relationship between whether a donor joined the program (given a value of 1)

Figure 5.1: Joining the Frequent Donor Program and Average Annual Donation Amount, Database 1

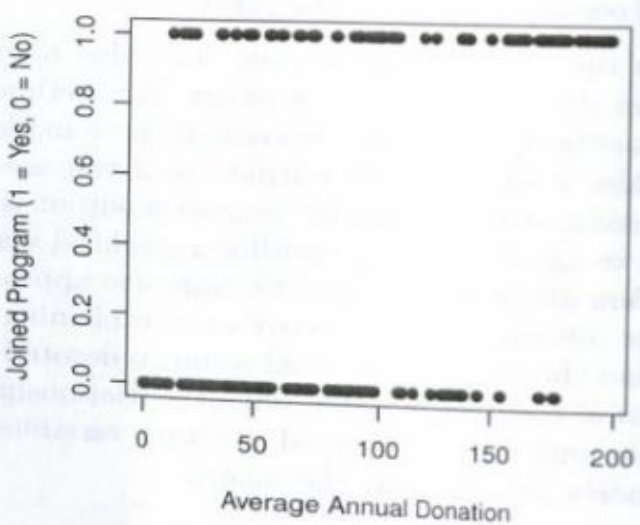
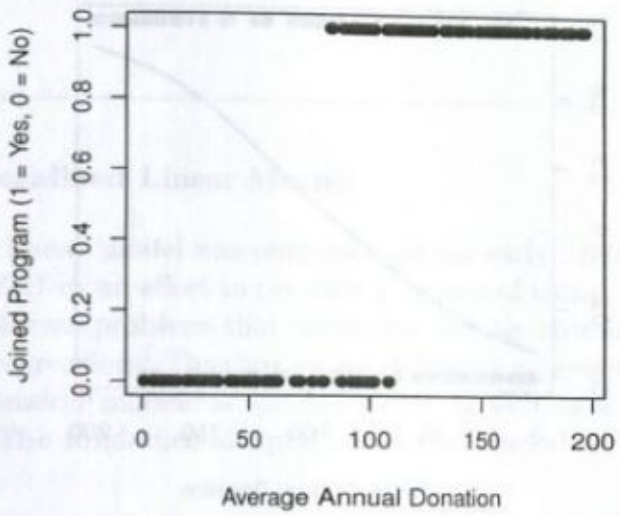


Figure 5.2: Joining the Frequent Donor Program and Average Annual Donation Amount, Database 2



or not (given a value of 0) and each donor's average donation amount for two different hypothetical databases.

An examination of the two figures indicates that the relationship between joining the program and past average donation amounts is greater in the second example database than it is in the first. However, in both cases it is apparent that as a donor's past average donation amount increases, the more likely that donor is to join the monthly giving program.

Ideally, we want to have an equation that maps the level of the past average annual donation amount to the probability of joining the frequent giving program for donors in the charity's database in order to determine which donors should be targeted with an offer to join the program. Figures 5.3 and 5.4 illustrate what this relationship would look like graphically for the two different hypothetical databases. In the figures the vertical (y) axis of the plots give the probability of joining the frequent giving program for a given past average annual donation amount. Consistent with what one would expect, the relationship between the average annual donation amount and the probability of joining the frequent giving program is sharper in the second example than it is in the first.

Logistic regression provides a means of estimating these probability functions. While our example has been based on examining a single predictor relation-

Figure 5.3: Probability Plot for Database 1

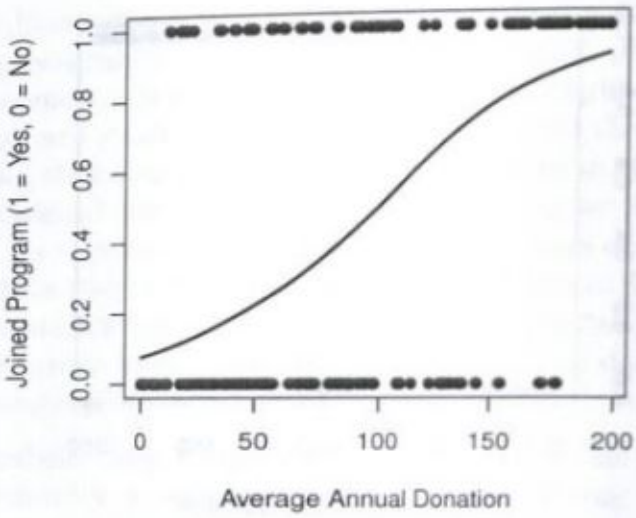
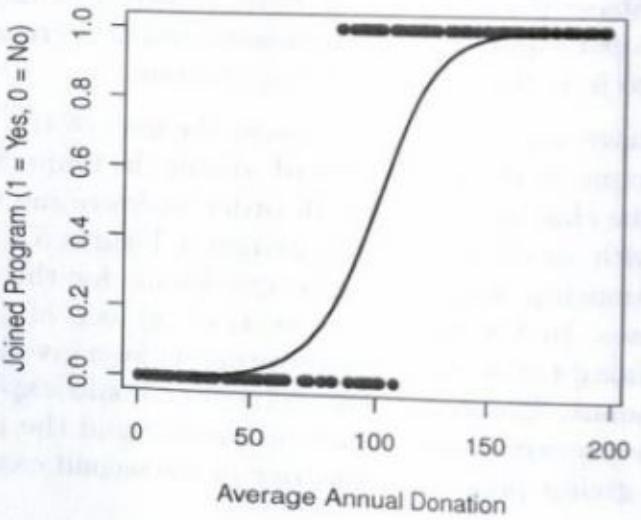


Figure 5.4: Probability Plot for Database 2



ship, logistic regression (like linear regression) can use more than one predictor variable, and some or all of those predicator variables can be indicator variables. As indicated earlier, there are a number of similarities between linear and logistic regression, the topic to which we now turn.

5.2 The Generalized Linear Model

The generalized linear model was proposed in the early 1970s by Nelder and Wedderburn (1972) in an effort to provide a means of using linear regression-like models to address problems that were not directly amenable to the application of linear regression.[1] The class of models they proposed included linear and logistic regression models as special cases, as well as a number of other related models. The fundamental equation of the generalized linear model is

$$g(\mathrm{E}(y)) = \alpha + \beta x_1 + \gamma x_2,$$

where $\mathrm{E}(y)$ is the expectation of the target variable y, $\alpha+\beta x_1+\gamma x_2$ is the linear predictor (the coefficients of which, α, β, and γ, are to be estimated), and $g()$ is the link function that maps the expectation of y to the linear predictor. While we present only the use of two predictor variables here, you should be aware that the method generalizes to an arbitrary number of predictors.

For our purposes, it is easier to think of things in terms of the inverse link function (which we label $f()$), rather than the link function itself. Consequently, we are interested in the equation

$$\mathrm{E}(y) = f(\alpha + \beta x_1 + \gamma x_2).$$

In the case of linear regression, the inverse link function is simply the identity function, so this equation becomes

$$\mathrm{E}(y) = \alpha + \beta x_1 + \gamma x_2.$$

Things become more complex for logistic regression since the expectation of the target variable y is actually a probability (which we label $\Pr(y = 1)$, but it could be $y =$ "Yes," or any other value we labeled as the desired outcome for the target variable), so the inverse link function needs to map the linear

[1] McCullagh and Nelder (1989) is an excellent, and updated source, on the generalized linear model.

predictor (which can take on any real value) to a value that falls in the zero to one interval of a probability. Moreover, that transformation must be an increasing monotone transformation of the linear predictor (which means as the value of the linear predictor increases, so does the probability). While these two conditions greatly limit the number of potential link functions, there is still a number of functions that meet these requirements. The most commonly used inverse link function is

$$\mathrm{E}(y) = \Pr(y = 1) = \frac{\exp(\alpha + \beta x_1 + \delta x_2)}{\exp(\alpha + \beta x_1 + \delta x_2) + 1}.$$

This inverse link function corresponds to the cumulative density function of the logistic distribution, which gives the method its name. Frequently, a logistic regression model using this inverse link function is called a *logit* model. Examine this function more carefully (something that might seem a bit daunting at first, but may be less difficult than it seems). In particular, notice that as the linear predictor takes on extremely large *negative* values that the term $\exp(\alpha + \beta x_1 + \delta x_2)$ approaches zero, so the ratio (the probability) also approaches zero. Conversely, when the linear predictor takes on extremely large positive values, the term $\exp(\alpha + \beta x_1 + \delta x_2)$ approaches positive infinity, so the ratio approaches one. The probability curves shown in Figures 5.3 and 5.4 are based on this inverse link function.

The other commonly used inverse link function is the cumulative density function of the standard Normal distribution (typically written as $\Phi(\alpha + \beta x_1 + \delta x_2)$), and a logistic regression model that uses this inverse link function is commonly referred to as a *probit* model. The drawback to the probit model is that the cumulative density function of the standard Normal distribution is not "closed form," which means (unlike for the cumulative logistic density function) values for it must be found via numerical integration (making this model computationally more demanding than the logit model). As illustrated in Figures 5.5 and 5.6 for the two examples of the frequent giver program used earlier, the value of the probit inverse link function for a particular value of the linear predictor (which is set to an average annual donation amount of $125 in the figures) is equal to the area under the standard Normal density curve from minus infinity to $125. As can be seen in the figures, the density is much more concentrated for the second example database, so the cumulative density (the probability of joining the monthly giving program) when the past annual donation amount is $125 is greater in this database than it is in the first database.

Figure 5.5: The Probit Inverse Link Function for Database 1

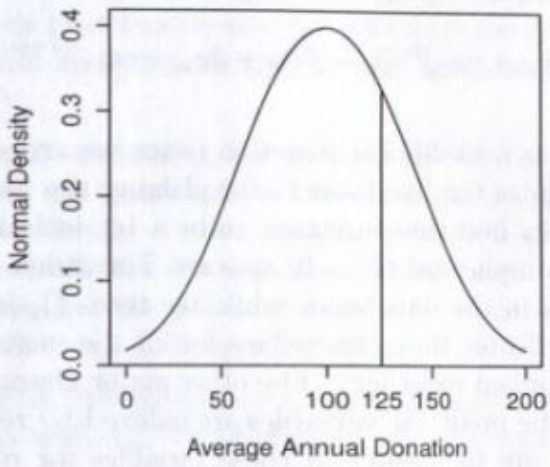
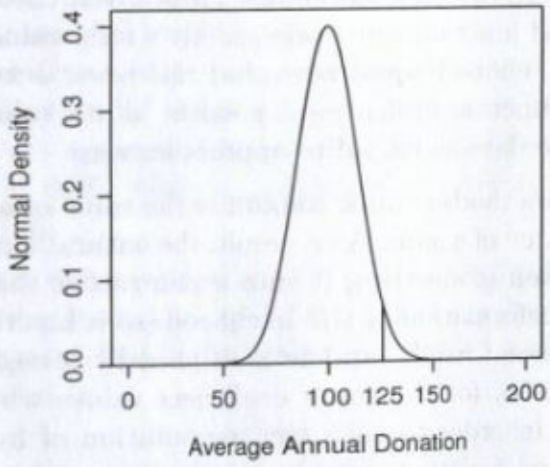


Figure 5.6: The Probit Inverse Link Function for Database 2



5.3 Logistic Regression Details

The logistic regression algorithm works by finding the set of coefficient values (α, β, and γ for our example linear predictor) that maximizes the function

$$\prod_i [f(\alpha + \beta x_{1i} + \gamma x_{2i}]^{y_i} [1 - f(\alpha + \beta x_{1i} + \gamma x_{2i}]^{(1-y_i)}.$$

This function is known as a likelihood function (since we are selecting coefficient values that maximize the likelihood of explaining the observed data). We admit that you might find this function to be a bit intimidating. However, it is actually less complicated than it appears. The index i is a counter for all customers/donors in the database, while the term $\prod_i$ is the product operator (i.e., it just indicates that the values for all the customers/donors in the database are multiplied together). The other major change is that now the target variable and the predictor variables are indexed by record (i.e., the value of y_i, x_{1i}, *and* x_{2i} are the values of these variables for record i of the database).

As you may have already guessed, $f(\alpha + \beta x_{1i} + \gamma x_{2i})$ is the probability that customer/donor i will respond favorably, while y_i indicates whether customer/donor i did in fact respond favorably. If the model is exactly correct for donor i (who responded favorably), then both y_i and $f(\alpha + \beta x_{1i} + \gamma x_{2i})$ will equal one, as will the value of the likelihood for that consumer/donor. If, on the other hand, the model is exactly wrong, then y_i will equal one, while $f(\alpha + \beta x_{1i} + \gamma x_{2i})$ will equal zero, and the value of the likelihood for that customer/donor will equal zero (which would cause havoc given the multiplicative structure of the likelihood function since one exactly wrong value would make the entire value of the likelihood equal zero, but this never actually occurs). Overall, the likelihood function indicates a good fit as its value approaches one, and a poor fit of the data as its value approaches zero.

Numerical optimization methods cannot maximize the value of a product, but they can minimize the value of a sum. As a result, the natural logarithm of the likelihood function is taken (converting it into a sum rather than a product) which is a monotone transformation of the likelihood (so it has the same maximum point as the likelihood itself), and is multiplied by a negative number (which reaches its minimum for the same coefficient values which maximize the original likelihood). In order to ease the computation of hypothesis test statistics for the estimated logistic regression model (but without influencing the coefficient estimates), the natural logarithm of the likelihood function is multiplied by -2 (rather than -1 as one might expect), resulting in a quantity

that is called the *deviance* (which we admit is an odd name). Thus, logistic regression ultimately attempts to find coefficient values that minimize the deviance of a model given the data, but these same coefficient values correspond to those that maximize the original likelihood function. A model that fits the data perfectly (something that never occurs) would have a deviance of zero, while the deviance for a less-than-perfect fit is greater than zero. There are a number of methods that can be used to minimize the deviance of the model given the data, and R uses a method known as Fisher Scoring (McCullagh and Nelder, 1989).

While the important statistics generated in the estimation of a logistic regression model are not the same as they are for a linear regression model, they are analogous. Specifically, a z-test (based on the standard Normal distribution) is used to test whether the individual coefficients are different from zero rather than a t-test, but the p-values of these tests are interpreted in exactly the same way as they are in linear regression.

A true R^2 cannot be calculated, but a McFadden R^2 can be calculated (McFadden, 1974). As with R^2, the McFadden R^2 is bounded between zero and one, and the closer this statistic is to one, the better the model fits. However, as with R^2, the value of the McFadden R^2 never decreases (and typically increases) as additional variables are added to the model, leading to potential overfitting problems. The formula used to calculate the McFadden R^2 value is

$$1 - \frac{deviance}{null\ deviance},$$

where the *null deviance* is the deviance of a logistic regression model that contains only a constant. If the model with additional variables does no better than a constant-only model, then the deviance and null deviance are equal to each other, and the McFadden R^2 equals zero. On the other hand, if the model with additional variables fits perfectly, then the deviance of that model equals zero, and the McFadden R^2 equals one. To give some sense of what different McFadden R^2 values mean, the probability function shown in Figure 5.3 has a McFadden R^2 of 0.25, while the probability function in Figure 5.6 has a McFadden R^2 of 0.6.

In linear regression the use of adjusted R^2 is intended to mitigate potential overfitting problems associated with R^2. The analogous statistic for logistic regression is known as the Akaike Information Criterion (or AIC; Amemiya, 1985), which is given by

$$deviance + 2c,$$

where c is the number of estimated coefficients in the model. Based on this formula, it is clear that the AIC is the value of the deviance for a model that has been penalized for the number of model coefficients. Given this structure,

we are often interested in finding the model that has the *minimum* AIC value. At this point it should be noted that the AIC can also be used for linear regression models as well since the criterion used in linear regression (minimizing the sum of the squared model errors) corresponds to the minimization of the deviance of a model if we assume that the error terms are normally distributed. Since the deviance of a linear regression model can be calculated, so can its AIC. However, by historical convention, adjusted R^2 is the statistic usually reported.

5.4 Logistic Regression Tutorial

The purpose of this tutorial is to make you familiar with some of the logistic regression tools in R. In addition a number of the visualization tools that were examined in the linear regression tutorial (Chapter 4) will also be used in this tutorial. It turns out that visualizing the relationship between potential predictor variables and the target variable requires a bit more work when the target variable is a categorical (a *factor*, in R) rather than a continuous variable. Finally, you will see an example of a process that can be used to quickly develop a reasonably good predictive logistic regression model. It is likely that a better model could be found, but the improvement would be small, and the effort large.

5.4.1 Highly Targeted Database Marketing

The data set used to illustrate the methods presented in this lab involves a variant on an "up-selling" application. Specifically, the data come from a project conducted for a Canadian Charitable Society, British Columbia and Yukon Region (or CCS). The objective of this project was to develop a model to determine which of the current CCS donors should be selected for a targeted mailing to encourage them to join the "monthly giving program." Under the monthly giving program a donor to the CCS elects a monthly amount to give and provides a credit card number so that automatic monthly donations are made. The CCS had offered this program as one of several payment options under its general annual appeal campaign over the prior three years. Donations from individuals who had moved to this program from the annual campaign increased their annual giving by a factor of three on average (e.g., moving from giving the CCS $20 a year to $60 a year), and had a very low attrition rate (about a 3 percent loss per year compared to the 33 percent loss for the annual campaign). However, only about 1 percent of donors elected to join

the monthly giving program when it was offered as one of several payment options and was given no special prominence in the CCS's appeal. Based on this experience, the CCS was interested in developing a special campaign to increase the number of donors enrolled in the monthly giving program. In the absence of any modeling, all that the CCS could do would be a costly mass mailing with an attendant low response rate. A highly targeted campaign would be preferable, requiring identifying those current donors most likely to upgrade to the monthly giving program in response to CCS's direct mail.

To this end, a database was constructed that included donors' behaviors prior to being offered the monthly giving program, and a variable indicating whether or not they had joined the program after the offer. A simple analysis would be to compare the joiners and non-joiners on the variables available and look for differences. A much better approach would be to build a model of the probability of joining the program as function of available variables, and then to use this model to predict the probability of other donors accepting the offer to join the monthly giving program.

In the following subsections, we will first introduce two important concepts in database marketing, over-sampling and controlling over-fitting.

5.4.2 Oversampling

Because the percentage of individuals who belonged to the monthly giving program is a very small percentage of the total donor database, the large majority of non-members will overwhelm any predictive model estimated on the total database. Determining what influences an individual to become a member of the program is therefore difficult. Instead, following common data mining practice, samples were drawn (without replacement) to create a new database that consists of a 50–50 percent split of monthly givers and non-monthly givers. This is known as *over-sampling* the target population of monthly program members. A model estimated on this more balanced database has a better chance of recognizing differences between the two types of donors.[2] When using the final model to calculate return on investment, the important trick to remember is that we have over-sampled the members of the program to create the model. We will need to correct from the one-in-two proportion in the oversampled database to the 1-in-100 proportion in the original CCS database. That is a simple ratio correction, and will be discussed later.

[2]The needed level of over-sampling varies with the predictive modeling tool used. If only logistic regression is used, over sampling to 20–80 split seems to work fairly well. However, other predictive modeling tools (tree models in particular) work best with a 50–50 split sample.

5.4.3 Overfitting and Model Validation

In the previous discussion, we really should say "samples" (as opposed to "sample"). The best way to control overfitting using any of the models we will encounter from here on is to split the database into two (sometimes more) samples. Each contains the same variables and oversampling rate, but included completely different individuals. An *estimation sample* is used to calibrate several potential models. The *validation sample* is not used in the process. It is held back, and used to test, or validate, the models, and to select the best among them. This is done by using the predictor variables from the validation data (not used to create the model) to calculate *predicted* target values. Since we also have the *observed* target values, we can compare, and determine how well the model predicts on data that were *not* used in creating the model. We then select the model (estimated using the estimation sample) that best predicts the target value in the validation sample. Some modeling procedures take this one step further, creating a third sample, a *holdout sample*. In the remainder of this book we will only use two samples.[3]

This procedure is used to avoid the possibility of developing a model that happens to fit the estimation sample data well but is fitting variations that are not repeatable from sample to sample, that is, fitting unrepeatable noise in the data. When applied to the new validation sample with different "noise" it will fit worse. We have already met this problem, and one way of helping to control for it linear regression. The adjusted R^2 controls for merely capturing noise when adding variables by reducing the R^2 by an adjustment that depends on the number of variables used. The adjusted R^2, the AIC, and validation and holdout methodologies, guard against *overfitting* the estimation sample.

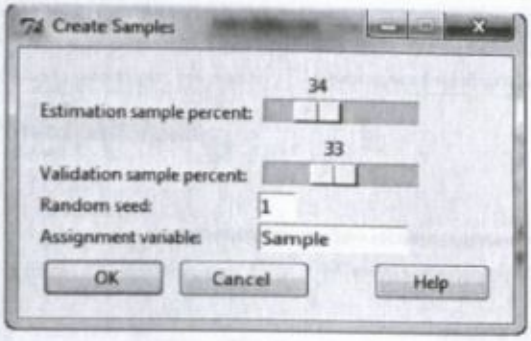
In this tutorial, and those in the next three chapters, we are interested in calibrating and selecting models, so we will only be working with the estimation and validation samples, which we will create using tools in R Commander. The data set we will be using in this tutorial (and for the next several) is entitled CCS, and can be found in the BCA data library.

1.

Use the pull-down menu command **Data → Get From →R package → Read data from an attached package. . .** to read the **CCS** data set from the BCA package into R.

[3]Jargon Alert: The estimation sample is often called the *training* sample, and the validation sample is sometimes called a *test* sample. As well, when only two samples are used, the terms "holdout" and "validation" can both be applied to the second sample. Always take care to clarify exactly what is meant when you see these terms.

Figure 5.7: The Create Samples Dialog



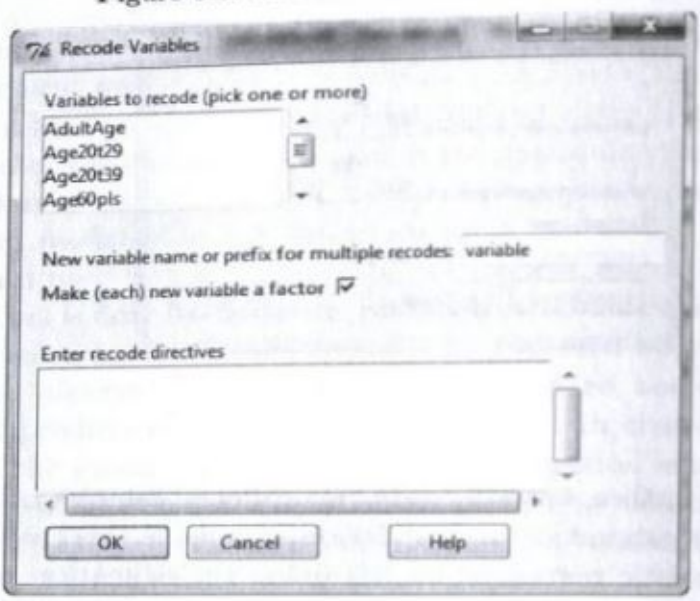
2.

Before proceeding further, we will create two different samples of individuals within the data, an estimation and validation sample. In this tutorial we will estimate several logistic regression models using the estimation sample, and in the tutorial of the next chapter we will use the validation sample to assess whether our models can accurately predict new data. To create the samples use the pull-down menu command **Data → Organize → Create samples in active data set...**, which brings up the dialog box shown in Figure 5.7. As indicated above, for this tutorial, and the next several, we will only be using an estimation and validation sample, but not a holdout sample. With this dialog, the size of the holdout sample is not specified, but is determined by the percentage of the total data set not included in either the estimation or validation sample. In Figure 5.7, 77% of the sample is for estimation and validation, leaving 33% for holdout. Consequently, if the percentage of the total data set allocated to the estimation and validation samples equals 100 percent, no holdout sample is created. At this point **use the slider bars to set both the estimation and validation samples to 50 percent of the data set.** The two samples are drawn at random (without replacement). Which specific records are allocated to each data set is determined the "Random seed:" field, which controls how R generates random numbers. The use of a random seed allows the same samples to be created by different users on different occasions. Use the **View data set button** to see that a new variable named "Sample" has been added to the data set, with values "Estimation" and "Validation." This variable will be used to specify which individuals we use for the following procedures.

3.

As in the linear regression tutorial, we will start with an examination of data visualization tools. The variable of central interest is `MonthGive`, which is

Figure 5.8: Recode Variables Dialog

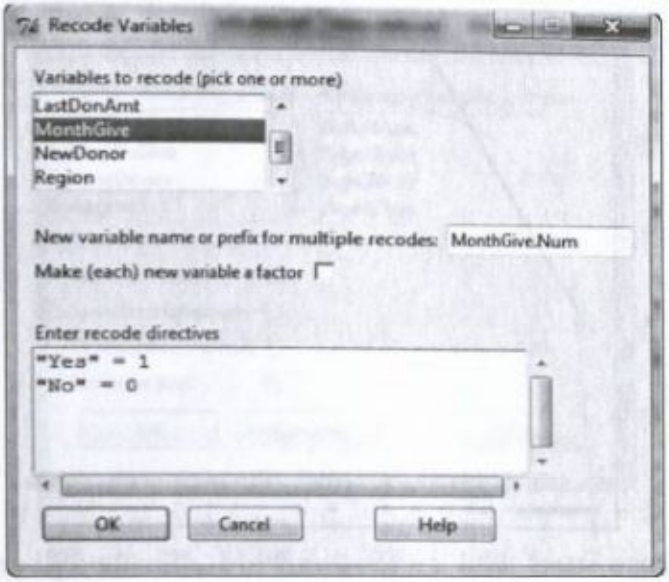


a binary factor that takes on the level "Yes" if a donor signed up for the monthly giver program and the level "No" if a donor did not. **View data set** to confirm these values. We will create a new numeric variable that takes on the value 1 if the level of `MonthGive` is "Yes" and the value 0 when the level of `MonthGive` is "No." The tricky and useful bit is that the average of this variable that consists of 0s and 1s will be between 0 and 1, and is the proportion of donors who joined the monthly giver program. (To convince yourself, calculate by hand the average for ten donors, three of whom are monthly givers). This allows simple and easy-to-understand visualizations. To create the new numeric `MonthGive.Num` variable use the pull-down menu option **Data → Manipulate variables → Recode variable...**, which will bring up the dialog box shown in Figure 5.8.

4.

In the **Recode Variable** dialog box, **select `MonthGive`** as the "Variable to recode (pick one or more)"; **enter `MonthGive.Num`** in the "New variable name or prefix" field. We want the new variable to be numeric so we can take averages, so **UNcheck the "Make new variable a factor" box**. In the expression box (the entry box located on the lower right-hand side of the Recode Variable dialog box), **enter `"Yes" = 1`, press <Return>, and on the second line enter `"No" = 0`**. When you are done, your **Recode Variable** dialog box should look like the one shown in Figure 5.9. Remember that R is case sensitive, so the entries need to be exactly as shown.

Figure 5.9: The Completed Recode Variables Dialog



One important point that needs to be made is that the lift chart tools we present in the next chapter only work if the target variable is a two-level (binary) *factor*. While we will use `MonthGive.Num` for visualization and exploration, it is not a factor, therefore you will want to use the factor `MonthGive` as the target variable when you move on to constructing your logistic regression models. You have to remember this since the logistic regression tools are perfectly happy using either a numeric variable with zeros and ones, or a factor variable with "Yes" and "No" as the target variable.

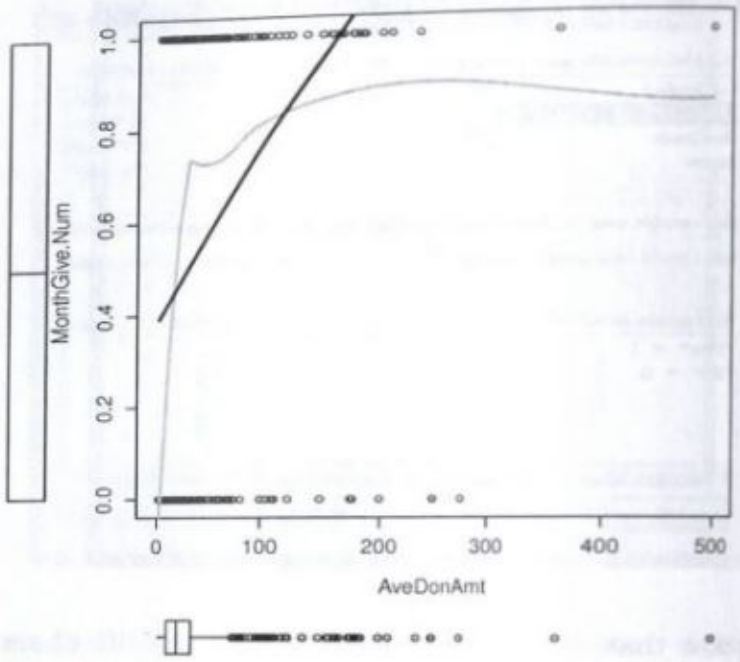
5.

Press **OK** in the Recode Variable dialog box to create `MonthGive.Num`, and then create a scatterplot (through the pull-down menu option **Explore and Test** → **Visualize**→ **Scatterplot...**) using `MonthGive.Num` as the y-variable and `AveDonAmt` as the x-variable. You should have a plot that looks like the one shown in Figure 5.10.

6.

The circles represent the data points. The estimated "smooth line" (solid line) attempts to fit the data points in Figure 5.10, and indicates there is a strong nonlinear (in this case, concave) relationship between `AveDonAmt` and `MonthGive`. The concave relationship suggests that a logarithm transformation of `AveDonAmt` may be in order for this variable. While the smooth line in

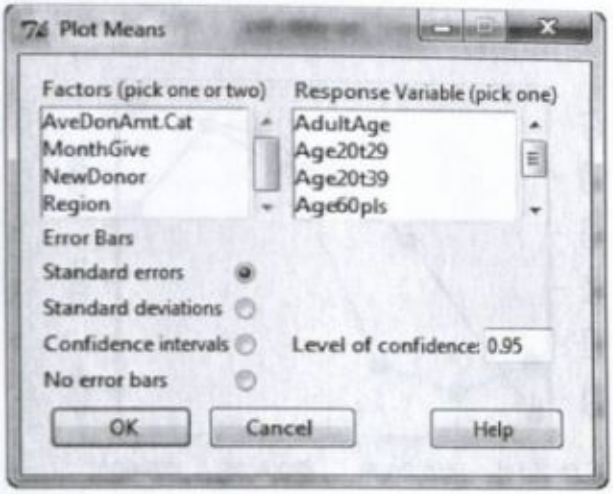
Figure 5.10: Monthly Giver vs. Average Donation Amount



the scatterplot is one of the easiest ways to look for the nonlinear relationships between two variables, it tends to run into problems in binary (0/1) data when there are very unequal numbers of the two binary responses. Fortunately, we have oversampled the Yes responses (the 1s) so that there is a 50–50 split between Yes and No values, which allows the smooth line to detect the nonlinear effect of `AveDonAmt`. However, if we had not oversampled the Yes values, almost all of the variables would have a y-value of 0, and the smooth line would have been a straight line at 0 for `MonthGive.Num`.

Fortunately, there is a second method that allows us to look for nonlinear relationships in cases where the oversampling is not as extensive (i.e., when the sample contains fewer than 50% positive responses to the target variable). The second approach involves binning (which we have done before) the continuous predictor variable `AveDonAmt` into categories, and then using a plot of means to visualize the relationship between the target and predictor variables. This method works best if there is a fairly even distribution of the predictor variable across its range of values. Things are a bit less ideal when there is a large skew (e.g., when a large number of customers have a low value, while a small number have a high value) in the predictor of interest, but even in these cases useful information can be obtained. Use the pull-down menu option **Data → Manipulate variables → Bin a numeric variable...** to create the new variable

Figure 5.11: Plot of Means Dialog



`AveDonAmt.Cat`. Specifically, **bin `AveDonAmt` into four equal-count bins and select "ranges" as the level names**. As usual, check the result with **View data set.** `AveDonAmt` turns out to be highly skewed, which is why we are using equal count bins. If it was less skewed, we would have selected equal interval bins, which allows for a plot of means that is a bit easier to interpret. However, equal interval bins with highly skewed variables can end up with bins that have no members, which causes problems. Next, use the pull-down menu option **Explore and Test →Visualize → Plot of means...** to bring up the dialog box shown in Figure 5.11.

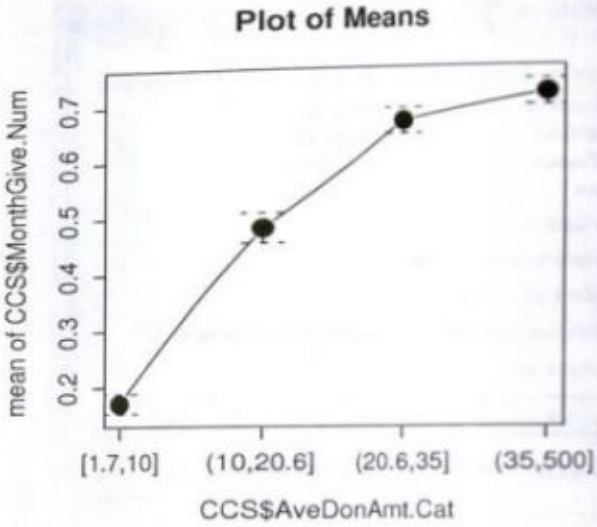
7.

In the Plot Means dialog box **select `AveDonAmt.Cat` as the "Factors (pick one or two)" and `MonthGive.Num` as the "Response Variable (pick one)."** While not required, you may want to select "No error bars" as the "Error Bars" option since if there are two or fewer observations in a particular bin, error bars cannot be created, and the plot of means tool will fail. Click **OK** and the plot in Figure 5.12 will be created.

8.

You can see the ranges represented by each level of `AveDonAmt.Cat` by looking at the labels along the horizontal axis of the figure. For example, the second label, (10,20.6], indicates the range of donations in the second level is from \$10 (the round left bracket indicates "not including \$10") to \$20.60 (the square right bracket indicates "including \$20.60"). Recall that the mean of `MonthGive.Num` is the proportion of "1" or "Yes" values in each category.

Figure 5.12: Plot of Means of Monthly Giver vs. Average Donation Amount



Slightly less than half of the individuals in the second donation category are members of the program. Donors with low average donation amounts have fewer members in the monthly giver program, and we can see that the increase with larger donations is of the concave (or decreasing-returns) structure, rather than linear.

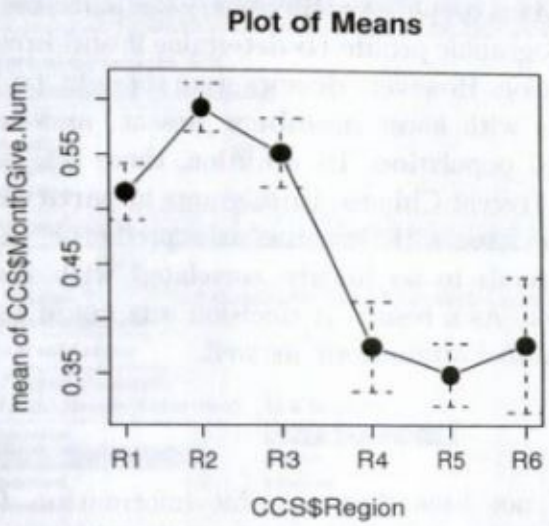
9.

The plot of means is a reasonable way of visualizing the effect of a continuous predictor variable (after binning) on a binary target variable, and it is the first choice in viewing the effect of a factor (nominal) predictor variable on the target variable. Another way would be to use a boxplot of the continuous variable, grouped by the two levels of the binary target variable, although this would not give as much detailed insight.

10.

Create a plot of means using `Region` as the factor and `MonthGive.Num` as the response variable. Once you have done this, you should have a plot like the one shown in Figure 5.13. Figure 5.13 suggests that later on we may want to consider reducing the number of levels of Region to two or three. Specifically, we may want to create a new region variable that combines regions R1 (Vancouver Island), R2 (Greater Vancouver), and R3 (the Fraser Valley) into a single region, and R4 (the North Coast of BC), R5 (South and Central Interior of BC), and R6 (the Northern Interior of BC and the Yukon) into a

Figure 5.13: Monthly Giver vs. Region Plot of Means



second region. In the interest of speculating (or hypothesizing, or theorizing, or storytelling), we note that a possible causal explanation of what is driving these apparent region effects is related to how urbanized a region is, with donors living in more urban regions being more likely to join the monthly giving program.

11.

Exercise: Ensure you have mastered the above material before continuing on by examining the relationship between `MonthGive.Num` and other potential predictor variables. Use the help file on the CCS data set to see the variable definitions and get some idea of what might be interesting. At a minimum, look at `LastDonAmt`, `DonPerYear`, and `YearsGive` as predictors of membership in the Monthly Giving Program, and identify any that appear to be nonlinear.

12.

We are now ready to begin building a logistic regression model. However, before doing this we need to consider which variables are likely candidates for predictors. This stage makes use of managerial knowledge, and is essentially creating some theory to guide our modeling. This will hopefully speed up the development of a *reasonably good* — if not the absolutely best — model. Based on discussions with CCS personnel, a number of donor characteristics were identified as generally being related to donation behavior on the part of donors. These characteristics include measures related to past donation

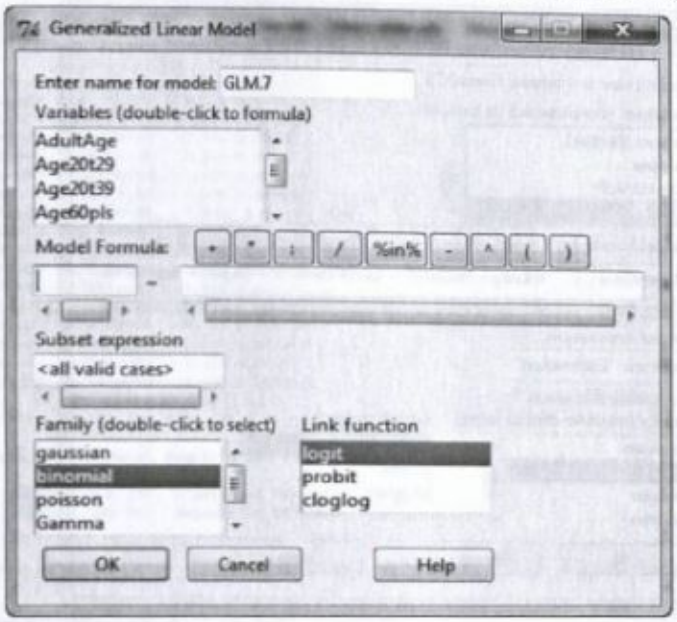
behavior (captured by the variables `AveDonAmt`, `LastDonAmt`, `DonPerYear`, and `YearsGive`) of the donor, the region in which the donor resided, and (anecdotally) his or her demographic profile. The CCS had never surveyed its donors to develop a demographic profile to determine if and how they differed from the general population. However, donors were thought to be on average older, live in households with fewer members present, and to have higher incomes than the general population. In addition, there was some evidence that some ethnic groups (recent Chinese immigrants in particular) were more likely to be donors. One issue with income as a predictor variable is that an individual's income tends to be highly correlated with his or her level of educational attainment. As a result, a decision was made to examine the potential effect of educational attainment as well.

13.

Although the CCS did not have demographic information for individual donors, it did have the postal code of each donor, and access to neighborhood-level census data. The postal code was used to assign the neighborhood-level (typically 300 to 400 households) average demographic characteristics in which the individual resided, based on the 1996 Census of Population, to the individual. These geo-demographic variables are therefore not true measures of the demographic characteristics of the individual, but are proxy measures based on population averages of the small geographic area in which a donor resides. For instance, we do not know the donor's actual income, but we can determine the average income level among households that reside in the small geographic area in which the donor resides, and use this as a proxy measure for that individual's income level. We do need to remember that these variables are only proxies of the neighborhood when *interpreting* our estimated model.[4] A number of variables related to the age distribution of the donors' neighborhoods were included in the data set to capture the possible effects of donor age, but we will only consider two of these: `Age20t29` (the percentage of people residing in the area who are between 20 and 29 years of age) and `Age70pls` (the percentage of people residing in the area who are 70 years of age or older). Two variables related to household size are included in the data set, but we will only examine the effect of `hh1mem` (the percentage of households with only a single member present). The effect of ethnicity and recent immigration is measured using the variable `EngPrmLang` (the percentage of households in the area in which the donor resides where English is the primary language spoken at home). Income is measured using the variable `AveIncEA` (the average annual income level of households residing in the same

[4]Technically, assuming that the area average represents the value for a household is known as an ecological fallacy.

Figure 5.14: The Generalized Linear Model Dialog



area as the donor). Finally, `FinUnivP` (the percentage of individuals in the area who have obtained a university degree) is used to capture the potential effect of educational attainment.

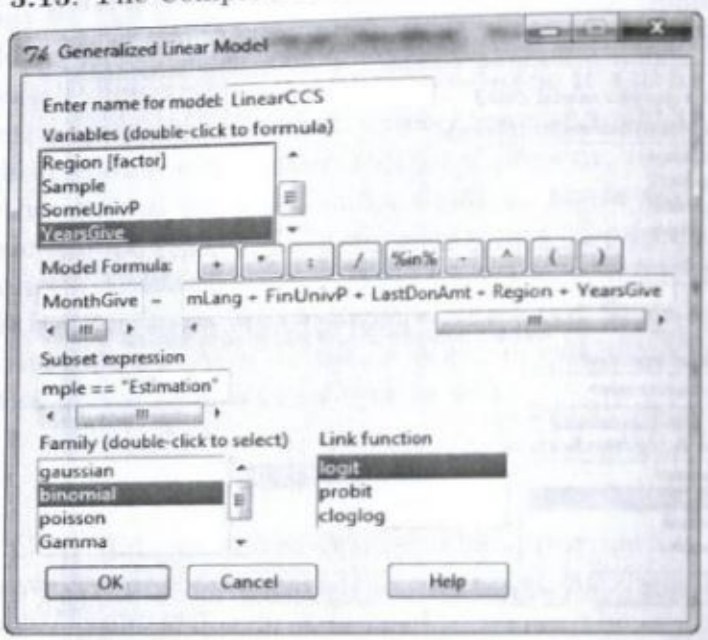
14.

To actually estimate the logistic regression model use the pull-down menu option **Models** → **Statistical models** → **Generalized linear model...**, which brings up the dialog box shown in Figure 5.14.

15.

The **Generalized Linear Model** dialog box is very similar to the **Linear Model** dialog box we saw in the linear regression tutorial. What is different are the "Family (double-click to select)" and "**Link function**" fields at the bottom of the **Generalized Linear Model** dialog box. Luckily, the default selections in these two fields correspond to the logistic regression model, so we do not need to alter them. The remaining fields are identical (and behave identically) to those in the **Linear Model** dialog box. To indicate that we have not transformed any variables yet, **enter `LinearCCS` in the "Enter name for model:" field**. The target variable is **`MonthGive`**, the categorical factor variable. Our continuous MonthGive.Num would work equally well, but later on we will be using lift charts for model assessment, which require binary factor variables as targets.

Figure 5.15: The Completed Generalized Linear Model Dialog



MonthGive should be on the left-hand side of the "Model Formula:" field, while the variables Age20t29, Age70pls, AveDonAmt, AveIncEA, DonPerYear, EngPrmLang, FinUnivP, LastDonAmt, Region and YearsGive (as with a linear model formula, the predictor variable names will be separated by a "+"). The one new field we will be working with is "Subset expression," which is the way in which we implement the use of different samples. To estimate the model using only records from the Estimation sample, **enter into this field the expression Sample == "Estimation."** The use of the two equal signs is intentional, and indicates a *test* for equality. If the statement is true, that is, if the variable `Sample` has the value `Estimation`, the record is selected for analysis. The single equal sign that we have used in previous situations is known as an assignment operator, and assigns a new value to a variable. Once you have done all of this, your dialog box should look like the one shown in Figure 5.15. When it does, press **OK** to estimate the model, the summary of which will appear in the R Commander output window, and should look "approximately" (e.g., the results may have a different variable order) like those shown in Figure 5.16.

16.

Figure 5.16 indicates that variables related to past donation behavior (`DonPerYear` and `LastDonYear`) have the greatest impact statistically, while the only demographic variables that seem to have any impact is `AveIncEA`

Figure 5.16: LinearCCS Model Results

(but still isn't quite statistically significant). The McFadden R^2 is a statistic that ranges between 0 and 1, with 1 being a perfect fit, and can be interpreted similarly to R^2 in linear regression. However, McFadden R^2 values are typically less than R^2 values, so we tolerate lower values for McFadden R^2. For the model the McFadden R^2 is a low, but acceptable, 0.114. The AIC value, 1012.6, is used like adjusted R^2 for choosing a set of predictors for a model by comparing its value across different models, since it includes both the goodness of fit and a penalty for using more variables. A reasonable critierion to select a model is to choose the one that minimizes the AIC.

Finally, it is difficult to determine if `Region` has any effect. One of the reasons is that R1 is the "base case" region, which Figure 5.13 indicates is in the high group, but has the lowest average in this group. Recall that the statistics for nominal variables are measuring whether each region has a significantly different impact from R1 (whereas for the continuous variables, the measure is whether the coefficient is significantly different from zero). The difference between the highest and lowest values in Figure 5.13 has the best chance of actually being significant. As with a linear regression model, a better way to examine if Region has an influence on its own is to use the pull-down menu option **Assess → Hypothesis tests → ANOVA** table, which produces the results in the R Commander output window shown in Figure 5.17. The results in this figure indicates that, on the whole, Region does have a statistically

Figure 5.17: LinearCCS ANOVA Results

```
> Anova(LinearCCS, type="II", test="LR")
Analysis of Deviance Table (Type II tests)

Response: MonthGive.Num
             LR Chisq Df Pr(>Chisq)
Age20t29       0.1189  1  0.7301842
Age70pls       0.0246  1  0.8752618
AveDonAmt      0.7661  1  0.3814137
AveIncEA       2.5569  1  0.1098129
DonPerYear    30.0458  1   4.22e-08 ***
EngPrmLang     0.0314  1  0.8593249
FinUnivP       0.1747  1  0.6759754
LastDonAmt    12.5129  1  0.0004042 ***
Region        12.4327  5  0.0293165 *
YearsGive      2.2941  1  0.1298686
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

significant effect, although a fairly weak one. Based on this, and Figure 5.13, we probably should reduce the number of levels in this variable and recalibrate the model.

17.

One thing that is surprising is that there appears to be no relationship between `AveDonAmt` and being a member of the monthly giving program. However, Figure 5.10 indicates that the influence of this variable is strongly non-linear (concave) in nature. Thus, a logarithm transformation seems to be in order. It turns out that the logarithm of an average (which is what we have for all our demographic measures in the CCS database) is problematic in terms of its interpretation (i.e., the log of the mean does not equal the mean of the log transformed original variable). As a result, we will only transform those variables that relate directly to an individual donor (i.e., `AveDonAmt`, `LastDonAmt`, `DonPerYear`, and `YearsGive`). At this point we will encounter a common technical concern with log transformations: the natural logarithm of zero (as well as negative numbers) is undefined, so if the variable we wish to transform has zeros, the computation will fail. Before doing log transformations, we must always determine which variables have zero values, and modify those variables. This is not an issue for `AveDonAmt` and `LastDonAmt` since to be a donor, an individual had to have given a non-zero amount in the past. `DonPerYear` and `YearsGive` could have zero values (years of giving to the CCS is measured as integer full years, so recent donors could still be assigned 0). You can easily determine if a variable has zero or negative values by using the pull-down menu option **Explore and Test → Summarize → Numerical summaries...**. The dialog box that then appears allows you to select a variable for which summary statistics will be printed to the R Commander output window. Select

Figure 5.18: Numerical Summaries for DonPerYear and YearsGive

```
> numSummary(CCS[,c("DonPerYear", "YearsGive")], statistics=c("mean
               mean       sd 0%        25% 50% 75% 100%    n
DonPerYear 0.6157976 0.5008012  0 0.2727273 0.5   1     5 1600
YearsGive  4.8593750 3.1716931  0 2.0000000 4.0   8    14 1600
```

`YearsGive`, and **`DonPerYear`**. When you are done, the summary statistics for these two variables should appear as they do in Figure 5.18. In this instance, the statistic of interest is the 0% quantile (which gives the minimum value of the variable). As can be seen in Figure 5.18, the minimum value of both `DonPerYear` and **`YearsGive`** is zero.

18.

For `DonPerYear` and **`YearsGive`** a modification of the log transformation is needed. Specifically, for variables with zero (but no negative) values, the transformation $\log(X + 1)$ will eliminate the problem with zeros, where X is the original variable to be transformed. As in the linear regression tutorial, use the pull-down menu option **Data → Manipulate variables → Compute new variable...** and this transformation to create new log transformed variables `Log.AveDonAmt`, **`Log.LastDonAmt`**, **`Log.DonPerYear`**, and **`Log.YearsGive`**. For `AveDonAmt` and **`LastDonAmt`** a simple log transformation, $\log(X)$, should be used. Check that the variables have been created and that they have no missing values (**View Data set**).

19.

Run a second logistic regression model, this time **entering `LogCCS` in the "Enter name for model:" field**. The target variable is again `MonthGive` and should be on the left-hand side of the "Model Formula:" field, while the variables **`Log.AveDonAmt`**, **`Log.DonPerYear`**, **`Log.LastDonAmt`**, `Log.YearsGive`, **`Region`**, **`Age20t29`**, **`Age70pls`**, **`EngPrmLang`**, **`AveIncEA`**, `FinUnivP`, and should be the right-hand side variables. Remember to **set the Sample=="Estimation."** Your results for this second model (with the exception of possible variable order differences) should be identical to those shown in Figure 5.19. The overall fit of the model has improved considerably, with the McFadden R^2 rising to a much more reasonable 0.177 and the AIC falling (improving) to 942. The log transformed version of **`AveDonAmt`** is now barely statistically significant (only at the 10 percent level, not 5 percent). However, the log version of **`YearsGive`** is not statistically significant, nor are any of the demographic variables. As before, determining whether or not `Region` is significant is difficult from this output, but can easily be de-

Figure 5.19: LogCCS Model Results

Output Window

```
Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)    -3.691e+00  9.487e-01  -3.890  0.00010 ***
Log.AveDonAmt   4.695e-01  2.600e-01   1.806  0.07097 .
Log.DonPerYear  1.111e+00  3.420e-01   3.248  0.00116 **
Log.LastDonAmt  6.022e-01  2.371e-01   2.540  0.01109 *
Log.YearGive   -1.656e-01  1.390e-01  -1.192  0.23345
Region[T.R2]    5.062e-01  2.698e-01   1.876  0.06065 .
Region[T.R3]    4.360e-01  2.658e-01   1.640  0.10098
Region[T.R4]   -1.884e-01  3.593e-01  -0.524  0.59998
Region[T.R5]   -1.708e-01  2.620e-01  -0.652  0.51448
Region[T.R6]   -4.604e-01  4.617e-01  -0.997  0.31868
Age20t29       -5.209e-01  1.994e+00  -0.261  0.79394
Age70pls        3.589e-01  1.021e+00   0.351  0.72531
EngPrmLang      6.230e-01  7.220e-01   0.863  0.38823
AveIncEA       -6.310e-06  6.359e-06  -0.992  0.32105
FinUnivP       -4.381e-01  1.071e+00  -0.409  0.68258
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1108.99  on 799  degrees of freedom
Residual deviance:  912.33  on 785  degrees of freedom
AIC: 942.33

Number of Fisher Scoring iterations: 4


> 1 - (LogCCS$deviance/LogCCS$null.deviance) # McFadden R2
[1] 0.1773371
```

termined using the pull-down menu option **Assess** → **Hypothesis tests** → **ANOVA table** (Figure 5.20), which indicates it is, but only at 0.10 percent level.

20.

Comparing the results of the two logistic regression models indicates that a log transformation of some of the variables noticeably improves the overall fit of the model (lower AIC and higher McFadden R^2) but, based on changes in variable significance levels, it may well be the case that not all variables should be log transformed. In addition, some of the variables we included in the original model seem to have little or no explanatory power, and could be removed, while (based on Figures 5.13 and 5.20) the Region variable would likely benefit from a reduction in the number of levels. To help make an assessment of which variables should be log transformed and which variables should be removed, Table 5.1 provides a comparison of the absolute values of $\Pr(>|z|)$ of all predictor variables except `Region` for both the `LinearCCS` and `LogCCS` models.

Figure 5.20: LogCCS ANOVA Results

```
> Anova(LogCCS, type="II", test="LR")
Analysis of Deviance Table (Type II tests)

Response: MonthGive.Num
               LR Chisq Df Pr(>Chisq)
Log.AveDonAmt    3.2125  1  0.0730756 .
Log.DonPerYear  10.8590  1  0.0009832 ***
Log.LastDonAmt   6.7393  1  0.0094312 **
Log.YearGive     1.4196  1  0.2334625
Region           9.9016  5  0.0780728 .
Age20t29         0.0681  1  0.7940791
Age70pls         0.1236  1  0.7252020
EngPrmLang       0.7393  1  0.3898763
AveIncEA         0.9986  1  0.3176492
FinUnivP         0.1670  1  0.6827739
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Table 5.1: LinearCCS and LogCCS Variable p-Values

Variable	LinearCSS	LogCCS
AveDonAmt*	0.380	0.070
LastDonAmt*	0.000	0.011
DonPerYear*	0.000	0.001
YearsGive*	0.131	0.233
Age20t29	0.730	0.794
Age70Pls	0.875	0.725
AveIncEA	0.115	0.321
EngPrmLang	0.859	0.388
FinUnivP	0.676	0.683

21.

An examination of Table 5.1 reveals that the only variable which the log transformation improved was `AveDonAmt`, while the demographic variables `Age20t29`, `Age70pls`, `EngPrmLang`, and `FinUnivP` have low z-statistics in both models. Based on the table, all demographic variables could be removed. Although, `AveIncEA` is nearly significant, so we will keep it in the set of variables for some additional testing. We will also keep `YearsGive` for the same reason, and only use a logarithmic transformation on `AveDonAmt`. In addition, as argued earlier, `Region` should be recoded to have only two levels (one containing original regions R2, and R3, and the other containing original regions R1, R4, R5, and R6). The reason for this choice is that regions R2 and R3 were found to be statistically different from R1 in both estimated models, but this is not true for regions R4, R5, and R6. Use the pull-down menu option **Data → Manipulate variables → Relabel factor levels...** to **create the variable `New.Region` in which regions R2, and R3 are assigned to the level "VanFraser" (since R2 is Greater Vancouver, while R3 is the Fraser River Valley), and regions R1, R4, R5, and R6 are assigned to the level "Other." View your data** to check you have the new variable with correct values. Following this, **estimate a third logistic regression model (which you should name `MixedCCS`), which has `AveIncEA`, `DonPerYear`, `LastDonAmt`, `Log.AveDonAmt`, `New.Region` and `YearsGive`** as predictor variables. Your results for this model should correspond to those contained in Figure 5.21.

22.

Figure 5.21 indicates that this model (based on both the McFadden R^2 and AIC) fits much better than did the original `LinearCCS` model. The McFadden R^2 is lower (indicating a loss of fit) for this model than for the `LogCCS` model (although the AIC favors the new model over the `LogCCS` model due to the reduction in the number of estimated parameters). The individual z-statistics indicate that all variables except `YearsGive` are statistically significant at the 10 percent level or better. At this point it seems appropriate to give up on `YearsGive` as a predictor variable. One of the most important changes is that `LastDonAmt` is now only significant at the 10 percent level, which is surprising. However, both a scatterplot and a plot of means for `LastDonAmt` indicate that the relationship between this variable and being a member of the Monthly Giving Program is concave in nature. As a result, **estimate one final logistic regression model, which you should label `MixedCCS2`, which uses `AveIncEA`, `DonPerYear`, `Log.AveDonAmt`, `Log.LastDonAmt`, and `New.Region` as predictor variables.** Your results for this model should be identical to those of Figure 5.22.

Figure 5.21: MixedCCS Model Results

Output Window

```
Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -3.852e+00  4.701e-01  -8.195 2.50e-16 ***
AveIncEA                -7.694e-06  4.078e-06  -1.887  0.05921 .
DonPerYear               8.761e-01  2.088e-01   4.197 2.71e-05 ***
LastDonAmt              -5.158e-03  2.724e-03  -1.893  0.05833 .
Log.AveDonAmt            1.193e+00  1.407e-01   8.474  < 2e-16 ***
New.Region[T.VanFraser]  4.797e-01  1.677e-01   2.860  0.00424 **
YearsGive                2.164e-02  2.762e-02   0.784  0.43332
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1108.99  on 799  degrees of freedom
Residual deviance:  915.42  on 793  degrees of freedom
AIC: 929.42

Number of Fisher Scoring iterations: 4


> 1 - (MixedCCS$deviance/MixedCCS$null.deviance) # McFadden R2
[1] 0.1745503
```

Figure 5.22: MixedCCS2 Model Results

Output Window

```
Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -3.384e+00  4.024e-01  -8.408  < 2e-16 ***
AveIncEA                -7.442e-06  4.132e-06  -1.801  0.07169 .
DonPerYear               8.089e-01  1.864e-01   4.340 1.43e-05 ***
Log.AveDonAmt            4.166e-01  2.581e-01   1.614  0.10653
New.Region[T.VanFraser]  4.792e-01  1.676e-01   2.859  0.00426 **
Log.LastDonAmt           6.214e-01  2.375e-01   2.617  0.00887 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1108.99  on 799  degrees of freedom
Residual deviance:  911.96  on 794  degrees of freedom
AIC: 923.96

Number of Fisher Scoring iterations: 4


> 1 - (MixedCCS2$deviance/MixedCCS2$null.deviance) # McFadden R2
[1] 0.1776685
```

Note: A feature of the software is that the dialog box will start with the parameters (with the exception of the model name) of whichever model is selected in the blue Model button. This allows you to generate new models quickly by starting with an existing model that is close to the new one.

23.

Figure 5.22 indicates that this model fits the data better than any of the other three models we have estimated (two of which had many more estimated parameters) based on both the McFadden R^2 and AIC statistics, and that all the predictor variables except `AveDonAmt` are statistically significant (it turns out that removing `AveDonAmt` significantly worsens both the AIC and McFadden R^2). The problems we experienced in obtaining significant results for both `AveDonAmt` and `LastDonAmt` are related to the fact that these variables are highly correlated with one another. In the context of linear regression, this is known as collinearity, and the effect in logistic regression (and other models we will encounter later) is similar. It appears that both variables help in predicting monthly giving program membership. If our purpose is simply to apply the model for prediction this is not a major problem. If we wish to determine which variables matter on the basis of significance levels, however, correlated variables are problematic. Correlation causes the reported estimates of the standard errors on the coefficients of the correlated variables to be inflated above their true values, which can make variables that truly matter appear to be statistically insignificant.

24.

Exercise: In two or three sentences, describe how the members of the monthly giving program differs from nonmembers.

25.

In the tutorial for the next chapter, we will be comparing the four models you estimated in this tutorial using lift charts, a technique that only considers the predictive abilities of the models. In subsequent tutorials, we will also want to have the last model (`MixedCCS2`) available. To make doing these tutorials easier, save the workspace files from this tutorial so that the four models are readily available to you when you start the next tutorial. Use **File → Save R Workspace as...** to bring up the dialog box, and **save the file as CCSLogistic.RData**. Remember that the extension ".RData" is case sensitive.

Chapter 6

Lift Charts

Lift charts (Barry and Linoff, 1997) are a graphical tool that both indicate the relative predictive power of different possible candidate models, and allow managers to quickly apply an estimated predictive model in order to address specific managerial questions.

This chapter describes how lift charts are created and how they are used to profitably target customers in direct marketing applications. To make the description of how lift charts are created concrete, this note makes use of an example based on a random sample of 50 donors from the Canadian Charitable Society data set (CCS in the BCA data library) and the `MixedCCS2` logistic regression model that is developed in the logistic regression tutorial in Chapter 5. The managerial use of lift charts is illustrated using a second example involving a regional telephone service provider's sales campaign for DSL service.

While the example approach makes the concepts behind the creation of lift charts more concrete, the disadvantage is that lift chart methods rely on there being many customers in the database, breaking those customers into a set of groups (traditionally ten groups), and then presenting averages for each of these groups. The use of only 50 observations allows us to reasonably look at all the data, but makes dividing the data into ten separate groups impractical for reasons of random sampling error. Consequently, instead of using ten groups in this note, the data will be divided into five groups. However, the concepts presented are applicable to an arbitrary number of groups.

6.1 Constructing Lift Charts

6.1.1 Predict, Sort, and Compare to Actual Behavior

Table 6.1 presents the data for the 50 donors randomly chosen from the CCS data set. The first column indicates whether the donor actually joined the monthly giver program, and the second column is the estimated probability (calculated using the MixedCCS2 logistic regression model) that they would

join the monthly giver program based only on their past donation behavior, the region of BC and the Yukon in which the donor resides, and the demographic characteristics of people living in the small geographic area in which the donor resides. The 50 observations have then been sorted from the highest to the lowest predicted probability. In this case, the predicted probability is a score that is given to each customer. The process of scoring a customer database, by this or any of a number of other methods, is a common method of quantifying our assessment of better and worse customers. The score becomes a targeting variable that allows us to approach more desirable customers.

Our ultimate goal is to apply a model to customers who have never joined the monthly giver program—that is, customers for whom we have not yet observed the "yes" and "no" information. Before doing that, we really should assess how well the model predicts. Here, "good" means how much better we can target the individuals in the database by using the model than by a purely random selection approach. The best way to assess the model is to compare the model predictions with already known behavior of individuals. Specifically, the individuals for whom we have "yes" and "no" data.

Table 6.1 has been divided into five groups of ten. In the first group eight out of ten donors (80 percent) joined the monthly giver program, while five out of ten (50 percent) joined the monthly giver program in the second group, only three out of ten (30 percent) joined the monthly giver program in the third group, and only two of ten (20%) in the final two groups. The group membership, based only on predicted behavior, is strongly related to actual behavior, exactly what we would hope. The prediction is not perfect, but much better than we could do without the model. Without the model, any random selection of 100 individuals from this database would give, on average, 40 members of the program, so any procedure that will allow us to find donors with a probability of joining greater than 40% is an improvement.

In this chapter, we will discuss how to construct two of the most commonly used (and managerially relevant) lift chart types: the "incremental response rate" chart and the "total cumulative response" chart. The incremental chart is a plot of the information described in the preceding paragraph, and a plot of this information can be found in Figure 6.1.

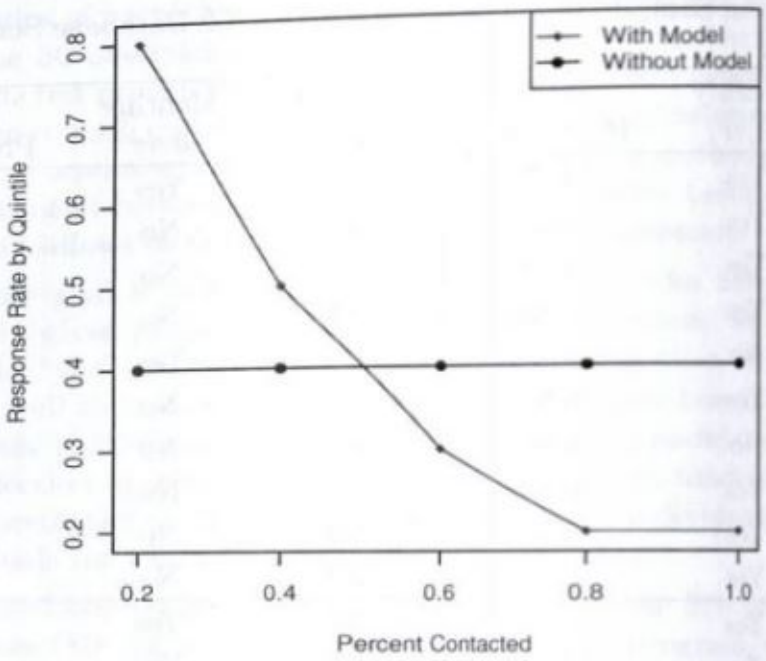
In Figure 6.1, the horizontal axis in the plot displays the cumulative percentage of the sample each group represents. For managerial purposes, we read this axis by saying, "Suppose we contacted the top 20% (second 20%, third 20%, etc.) of our database...". The data points for each group are the diamonds in the plot, and are connected together by solid lines. They allow our manager to finish the above statement as "... then we would have an incremental response rate in that quintile of 80% (50%, 30%, etc.)."

The horizontal line at the incremental response rate of 40% is the baseline

Table 6.1: A Random Sample of 50 Donors from the CCS Database Sorted by Fitted Probability

Sorted Rank	Monthly Giver?	Fitted Probability	Sorted Rank	Monthly Giver?	Fitted Probability
1	Yes	0.976	31	Yes	0.410
2	Yes	0.934	32	No	0.408
3	Yes	0.897	33	No	0.407
4	Yes	0.894	34	No	0.386
5	No	0.890	35	Yes	0.352
6	Yes	0.877	36	No	0.345
7	No	0.865	37	No	0.339
8	Yes	0.832	38	No	0.327
9	Yes	0.805	39	No	0.324
10	Yes	0.778	40	No	0.288
11	Yes	0.749	41	Yes	0.265
12	No	0.744	42	Yes	0.245
13	No	0.710	43	No	0.233
14	No	0.650	44	No	0.205
15	Yes	0.648	45	No	0.197
16	Yes	0.646	46	No	0.190
17	Yes	0.645	47	No	0.181
18	No	0.644	48	No	0.172
19	No	0.626	49	No	0.156
20	Yes	0.611	50	No	0.097
21	No	0.592			
22	No	0.565			
23	Yes	0.543			
24	Yes	0.537			
25	No	0.519			
26	Yes	0.514			
27	No	0.492			
28	No	0.471			
29	No	0.440			
30	No	0.435			

Figure 6.1: The Incremental Reponse Rate Chart for the Sample

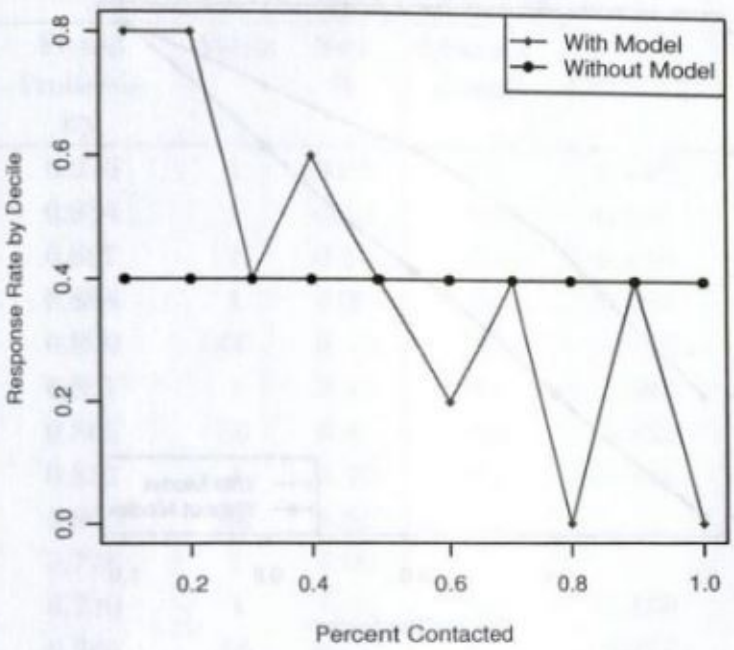


from which we assess model lift. It corresponds to the expected response rate we would achieve if we randomly selected donors from the sample (i.e., a 40 percent response rate).

In a standard incremental response chart the sample would have been divided into ten groups (or deciles) and not quintiles as done here. The problem with using deciles for this example is that there would be only five donors in each group, and such a small number can result in very jagged lift charts. To illustrate this point, an incremental response rate chart based on decile groups for the sorted sample of 50 donors is presented in Figure 6.2. In general, the more observations in each group, the smoother the resulting lift chart is likely to be.

The other useful lift chart, the total cumulative response chart, is constructed by taking the sum of total positive responses (donors who joined the monthly giver program) up to and including a particular group, and dividing that sum by the total number of positive responses for the sample. For the random sample of 50 donors, the total number of "positive responders" (i.e., monthly givers) is 20, and the number of these that are in the first group (the "best" fifth of the sample) is eight. Thus, if we had contacted the top 20% of the group as identified only by the model, we would have captured 8 of the 20 true positives, or 40%. If we had contacted the top 40%, we would have captured 13 of the 20 givers, or 65%. The total cumulative captured response over

Figure 6.2: The Incremental Response Rate Chart Using Deciles



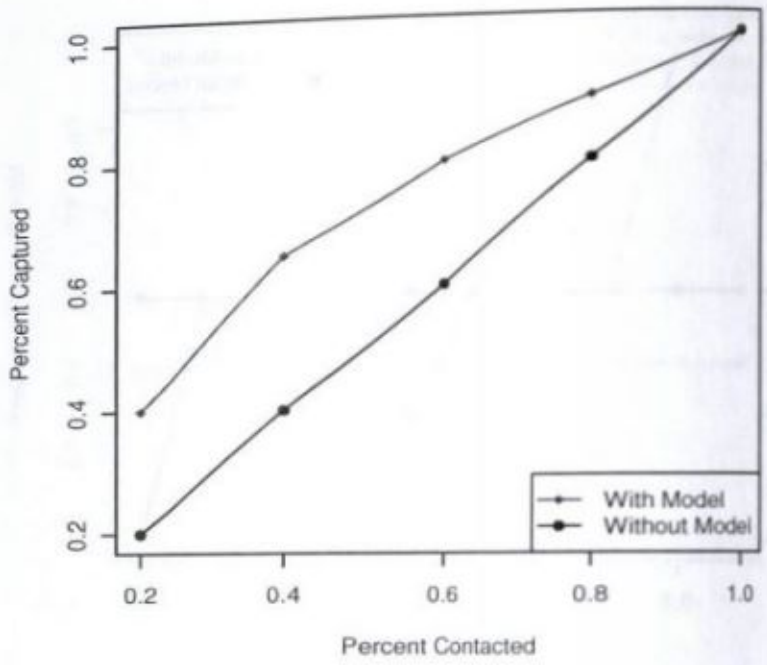
all five groups is, of course, 100%—if we contact everyone, we capture all responders. The cumulative response chart always ends at 100%. Figure 6.3 provides a plot of the total cumulative response chart. In the plot the 45-degree line is the baseline from which to assess model lift. It corresponds to the expected total cumulative response when donors are selected at random from the database. Generally, the more rapidly the total cumulative response approaches 1, the greater is the potential benefit from using the predictive model to target customers.

6.1.2 Correcting Lift Charts for Oversampling

So far we have been implicitly assuming that the proportion of members of the monthly giver program in the sample used for model estimation is also true of the entire Canadian Charitable Society donor base. As we know, however, the members have been oversampled here to allow the model to detect predictors of membership. Since our actual campaign will use the complete database, we will need to correct for this oversampling to determine the lift and true response rates we can expect when we apply our model to the original database. In the original complete database members represent only 1% of donors, not 40% as in the previous calculations.

The easiest way to think about this correction is to imagine a new propor-

Figure 6.3: The Total Cumulative Response Rate Chart for the Sample



tionate sampling of the original database that continued to draw donors until we had 20 donors who opted into the Monthly Giver Program. To get 20 donors that are in the program at a 1% response rate, we would expect to have to draw 2000 donors from the original database—20 members, and 1980 nonmembers. Comparing this to our oversampled set of 20 members and 30 nonmembers, we see that each one of the 30 nonmembers has to stand in for 66 nonmembers in the complete database (1980 / 30 = 66). We now simply reconstruct Table 6.1.1 replacing each nonmember with 66 nonmembers.

Table 6.2 presents the results giving each nonmember a weight of 66. An additional column is included to track the new percentages so that we can redefine our quintiles. The lines show the cutoff points at each quintile (these are not exactly at the quintile points but as close as we can get—with more records one can get very close). Table 6.2 can now be used to construct the new lift charts, which will tell us what to expect in the way of lift and response when we apply our model to the entire database.

As a quick check on the new lift chart, note that a 40% overall rate in the sample must become a 1% overall rate in the database. This means that the incremental response rates will be roughly 40 times smaller. It won't be exactly 40 times, because the cutoff points for the five bins will change. Since there are fewer "no's" in the earlier part of the database than the later, our correction adds fewer new individuals to the earlier part of the database than the later.

Table 6.2: The Weighted Sorted Sample

Monthly Giver?	Fitted Probability	Weight	New %	Monthly Giver?	Fitted Probability	Weight	New %
Yes	0.976	1	0.05	No	0.440	66	43.70
Yes	0.934	1	0.10	No	0.435	66	47.00
Yes	0.897	1	0.15	Yes	0.410	1	47.05
Yes	0.894	1	0.20	No	0.408	66	50.35
No	0.890	66	3.50	No	0.407	66	53.65
Yes	0.877	1	3.55	No	0.386	66	56.95
No	0.865	66	6.85	Yes	0.352	1	57.00
Yes	0.832	1	6.90	No	0.345	66	**60.30**
No	0.805	66	6.95				
Yes	0.778	1	7.00				
Yes	0.749	1	7.05	No	0.339	66	63.60
No	0.744	66	10.35	No	0.327	66	66.90
No	0.710	66	13.65	No	0.324	66	70.20
No	0.650	66	16.95	No	0.288	66	73.50
Yes	0.648	1	17.00	Yes	0.265	1	73.55
Yes	0.646	1	17.05	Yes	0.245	1	73.60
Yes	0.645	1	17.10	No	0.233	66	76.90
No	0.644	66	**20.40**	No	0.205	66	**80.20**
No	0.626	66	23.70	No	0.197	66	83.50
Yes	0.611	1	23.75	No	0.190	66	86.80
No	0.592	66	27.05	No	0.181	66	90.10
No	0.565	66	30.35	No	0.172	66	93.40
Yes	0.543	1	30.40	No	0.156	66	96.70
Yes	0.537	1	30.45	No	0.097	66	**100.00**
No	0.519	66	33.75				
Yes	0.514	1	33.80				
No	0.492	66	31.10				
No	0.471	66	**40.40**				

Table 6.3: Lift Chart Calculations Corrected for Oversampling

Quintile	Incremental Response	Cummulative Captured Response
1	12/408 = 2.94%	12/20 = 60%
2	4/400 = 1%	16/20 = 80%
3	2/398 = 0.5%	18/20 = 90%
4	2/398 = 0.5%	20/20 = 100%
5	0/396 = 0%	20/20 = 100%

That in turn moves the quintile cut points down in the table, and the first bin will now include more of the "yes's" than the first quintile did in the sample. This in turn raises the lift for the earlier part of the lift charts so that the reduction in the incremental response will be less than 40 times; the later parts of the chart will be reduced more than 40 times. The base rate of 1%, however, is exactly 40 times less.

Table 6.3 takes the results directly from Table 6.2 to calculate the incremental response rates and the cumulative captured response. For example, in the best 20% of the database, we can expect to have 12 members from the 408 individuals, for a response rate of 2.94%. Since there are a total of 20 members, we have captured 60% in the best 20% predicted by the model.

Figures 6.4 and 6.5 show the incremental response rate lift chart and the cumulative captured response lift chart corrected for oversampling. These lift charts indicate what we should expect when the predictive model is applied to the complete database.

6.2 Using Lift Charts

One real benefit of lift charts is that they make data mining models directly applicable to managerial decision making. The incremental response rate chart quickly allows a manager to determine how many existing customers should be targeted with a particular sales offer, and the predicted probabilities indicate which specific customers should be targeted. The total cumulative response chart allows the manager to forecast sales, revenues, and profits associated with a particular sales offer. In addition, the total cumulative response chart is a valuable tool for the marketing analyst for selecting a specific predictive model out of a set of potential predictive models. The tutorial for this chapter describes how to use the total cumulative response chart as a tool to select

Figure 6.4: The Weighted Sample Incremental Response Rate Chart

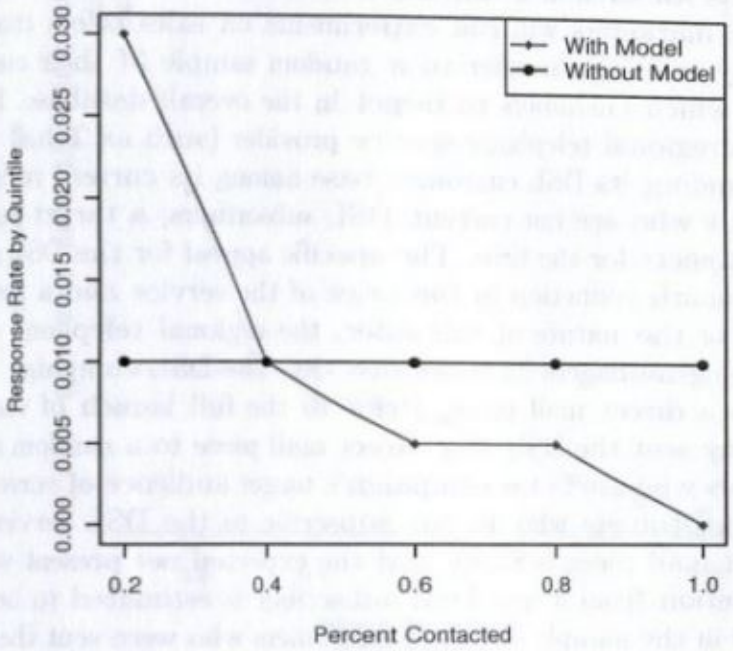
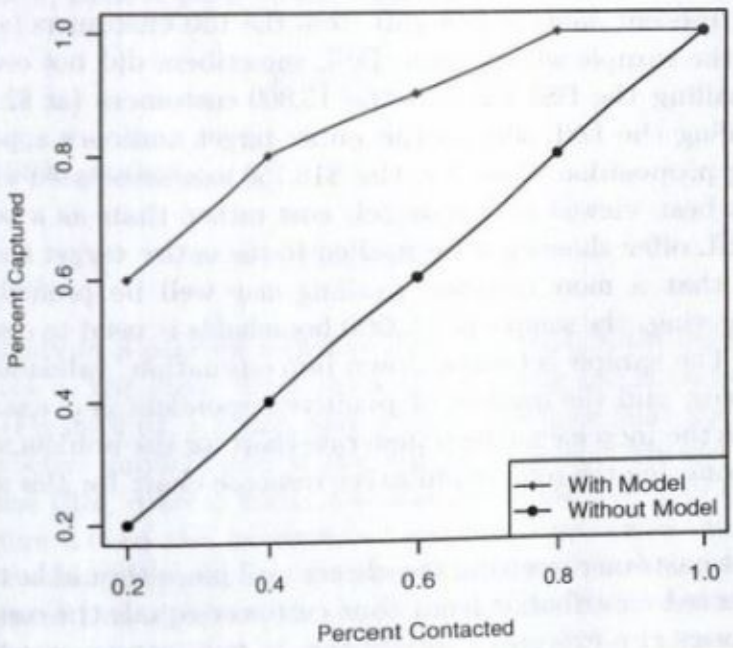


Figure 6.5: The Weighted Sample Total Cummulative Response Rate Chart



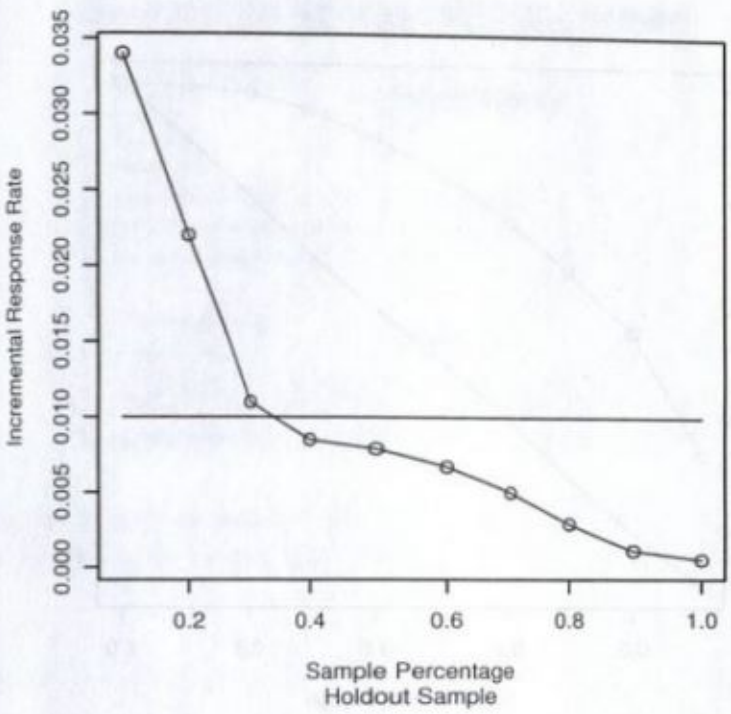
among potential predictive models, while here the focus is on the use of the chart to forecast sales, revenues, and profits.

Direct marketing is the area in which these tools have had their greatest impact. Most direct marketers will run experiments on sales offers that they are considering by sending the offer to a random sample of their customer list to determine which customers to target in the overall database. For example, consider a regional telephone service provider (such as Telus) that is interested in expanding its DSL customer base among its current residential land-line customers who are not current DSL subscribers, a target audience of 1.5 million customers for the firm. The specific appeal for the DSL service includes a three-month reduction in the price of the service and a free DSL modem. Because of the nature of this offer, the regional telephone service provider's marketing managers have decided that the DSL campaign should make the offer in a direct mail piece. Prior to the full launch of the campaign, the company sent the DSL offer direct mail piece to a random sample of 15,000 customers who are in the campaign's target audience of current residential land-line customers who do not subscribe to the DSL service. The cost of each direct mail piece is \$2.50, and the expected net present value of the profit contribution from a new DSL subscriber is estimated to be \$125. Out of the 15,000 in the sample of target customers who were sent the direct mail DSL offer 150 (or one percent) became DSL subscribers. Consequently, if all 1.5 million target customers were sent the direct mail piece, then (based on the positive response rate among sample customers) the company should expect that one percent of them (or 150,000) would become DSL subscribers.

The expected net present value of \$18,750 from the 150 customers (at \$125 per customer) in the sample who became DSL subscribers did not cover the \$37,500 cost of mailing the DSL offer to the 15,000 customers (at \$2.50 per customer), so mailing the DSL offer to the entire target audience appears to be a money losing proposition. However, the \$18,750 loss associated with the sample mailing is best viewed as a research cost rather than as a business loss. While the DSL offer should not be mailed to the entire target audience, it seems possible that a more targeted mailing may well be profitable. To undertake the targeting, the sample of 15,000 households is used to develop a predictive model. The sample is broken down into estimation, validation, and holdout sub-samples; and the number of positive responders is oversampled. Figure 6.6 contains the incremental response rate chart for the holdout sample, while Figure 6.7 contains the total cumulative response chart for this sample.

In general, the last customer receiving the direct mail piece should be the one for which the expected contribution from that customer equals the cost of the mailing. What causes the expected contribution to vary across customers is

Figure 6.6: The Incremental Response Rate for the DSL Subscriber Campaign



the probability they will respond favorably to the direct mail DSL promotional offer. Let the probability of becoming a DSL subscriber for the "break-even" customer be π, then it must be the case that

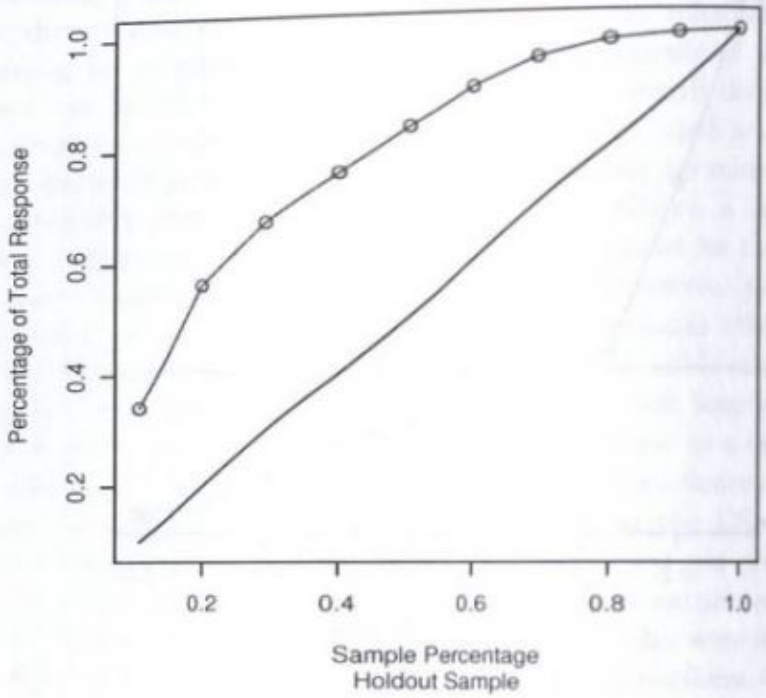
$$\$125(\pi) = \$2.50,$$

or, after a little algebra,

$$\pi = \$2.50/\$125 = 2\%.$$

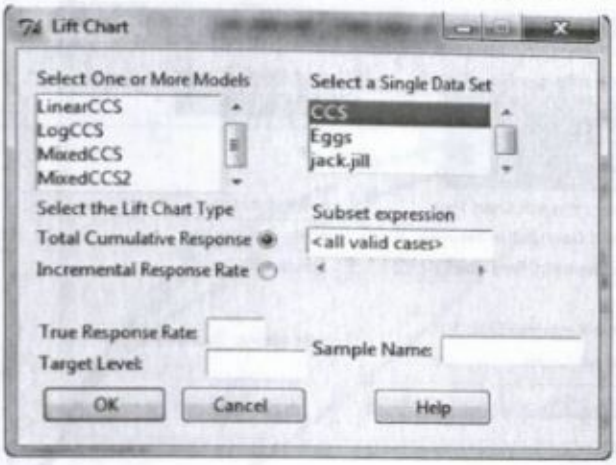
The probability of a positive response and the incremental response rate turn out to be one in the same thing. As a result, using the incremental response rate chart (Figure 6.6), a pencil, and a ruler, one finds that the "best" (based on the predictive model) 22% of the target customers should be contacted. To determine this, draw a horizontal line from 0.02 (or 2%) on the vertical axis of Figure 6.6 to the incremental response rate curve, and then at the point where your horizontal line intersects the incremental response rate curve, draw a vertical line down to the horizontal axis. Your vertical line should intersect the horizontal axis at roughly 0.22 (or 22%). Based on the total

Figure 6.7: The Cummulative Total Response Rate for the DSL Subscriber Campaign



cumulative response chart in Figure 6.7 (along with the ruler and the pencil), the "best" 22% of the target customers are expected to produce roughly 58% of the total new DSL subscribers that would be obtained if all the target customers were contacted. To see this, locate 0.22 along the horizontal axis of Figure 6.7 (where 0.22 corresponds to 22% of the total database), and then draw a vertical line from the horizontal axis at that point to the weighted total cumulative response curve. Next, draw a horizontal line from the point where your vertical line intersects the cumulative response curve out to the vertical axis. You should find that your horizontal line crosses the vertical axis at around 0.58 (or 58%). The 22% of customers to contact translates into $22\%(1.5\text{Mil} - 15{,}000) = 326{,}700$ actual customers that should be sent the direct mail piece (the 15,000 corresponds to the customers in the experimental sample who have already received the direct mail piece), while this mailing is expected to generate $58\%(15{,}000 - 150) = 8{,}613$ new subscribers, where the 15,000 corresponds to the 1% expected response rate multiplied by the 1.5 million target customers, and the 150 corresponds to the new subscribers already generated from the experimental mailing. The cost of mailing the offer to the 326,700 targeted customers (at \$2.50 each) is \$816,750, while the total

Figure 6.8: The Lift Chart Dialog Box



contribution from the expected 8,613 new DSL subscribers (at $125 each) is $1,076,625, resulting in an expected profit from the campaign of $259,875.

6.3 Lift Chart Tutorial

In this tutorial you will learn how to create lift charts using R Commander's lift chart tools, and how to use the validation sample total cumulative response chart to select among possible candidate models.

1.

Double-click on the workspace in which you saved your work from the logistic regression tutorial. This should launch R.

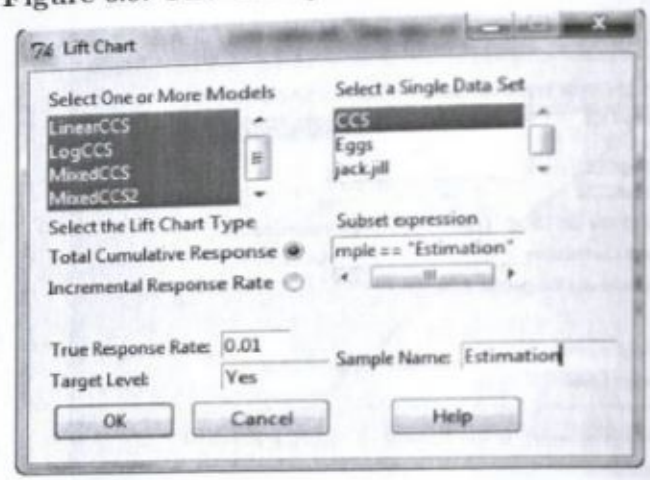
2.

Initially we will be creating a total cumulative response lift chart using the estimation sample. To do this select the pull-down menu option **Assess** → **Graphs** → **Lift charts**..., which will bring up the Lift Chart dialog box shown in Figure 6.8.

3.

In the Lift Chart dialog box **select the four CCS models from the "Select One or More Models" option**. For the **"Select a Single Data Set"**

Figure 6.9: The Completed Lift Chart Dialog

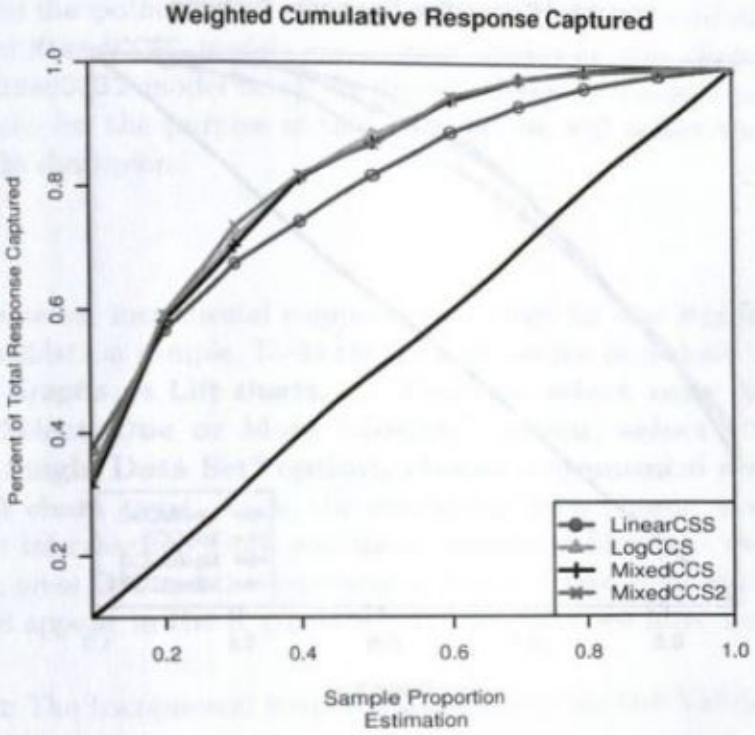


option select CCS. In the "Subset expression" field enter `Sample == "Estimation."` The default **Total Cumulative Response** for "Select the Lift Chart Type" is correct. The monthly givers in the CCS data set are oversampled so that they represent 50% of the sample. Recall, though, that they represent only 1% of all CCS donors. Consequently, **enter 0.01 in the "True Response Rate:" field.** The target level (i.e., the desired value of the target variable `MonthGive`) is "Yes," so **enter Yes (without quotes) in the "Target Level:" field.** The first lift chart we will be making is for the estimation sample. While the final model will be selected based on the *validation* sample, we will first briefly look at a cumulative total response chart for the *estimation* sample. This will give some indication of how the four different models estimated in the logistic regression tutorial perform. Give this chart a title by **entering "Estimation" in the optional field "Sample Name."** This name field does nothing more than add a title, but that is very useful to help you keep track of your charts. Once you have done all of this your Lift Chart dialog box should look like the one shown in Figure 6.9. When it does, press **OK**, and the total cumulative response chart shown in Figure 6.10 should appear in the R graphics window.

4.

An examination of Figure 6.10 indicates that the total cumulative response curve for the `MixedCCS` model is noticeably below the total cumulative response curve for the other three models over the range of the sample percentage from 10% to 30%, indicating that its performance over this range is inferior to the other models over this range. Similarly, for values of the sample percentage over 50% the `LinearCCS` model underperforms the other three

Figure 6.10: The Total Cumulative Response Rate Chart for the Estimation Sample

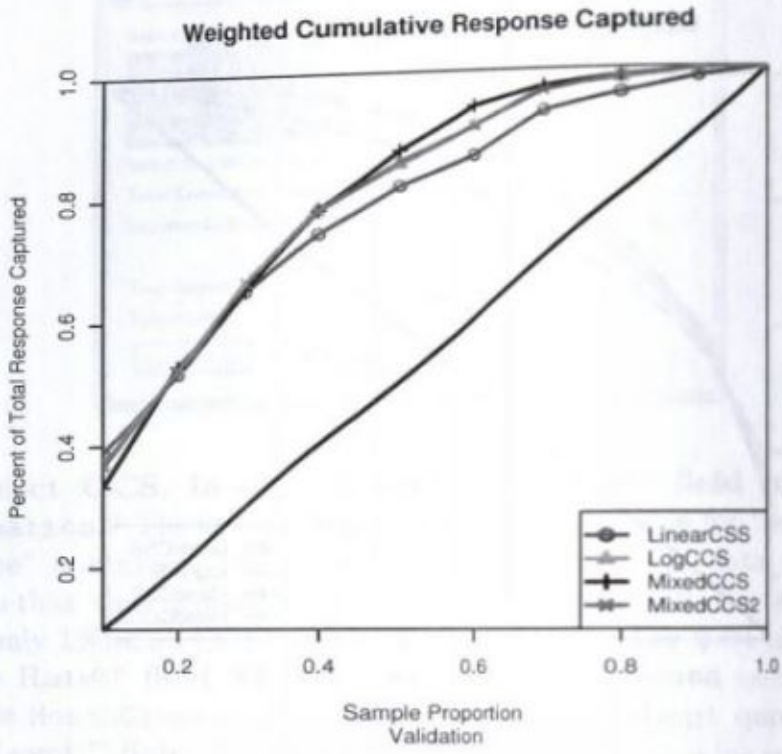


models. The `LogCCS` model and the `MixedCCS2` models perform comparably well over the entire range of the sample.

5.

To select among possible models, we will use the validation data to avoid overfitting. To do this we use the estimated coefficients just calculated based only on the *estimation* data set, and the predictor variables from the *validation* data set to calculate a predicted target probability (of `MonthGive == "Yes"`) for each individual in the validation data set. This predicted probability of the target in the validation data, and the the true observed target value in the validation data, is used to construct the lift chart and *validate* the model. To do this, use the pull-down menu option **Assess** → **Graphs** → **Lift charts...** again. **Select the four models and CCS data** set. The only difference between this cumulative response chart and the first one you created is that you need to **change the "Subset expression" to Sample == "Validation," enter Validation in the "Sample Name:" field.** Once you have done this, and

Figure 6.11: The Total Cummulative Response Rate Chart for the Validation Sample



pressed **OK**, you should have a lift chart identical to the one shown in Figure 6.11.

6.

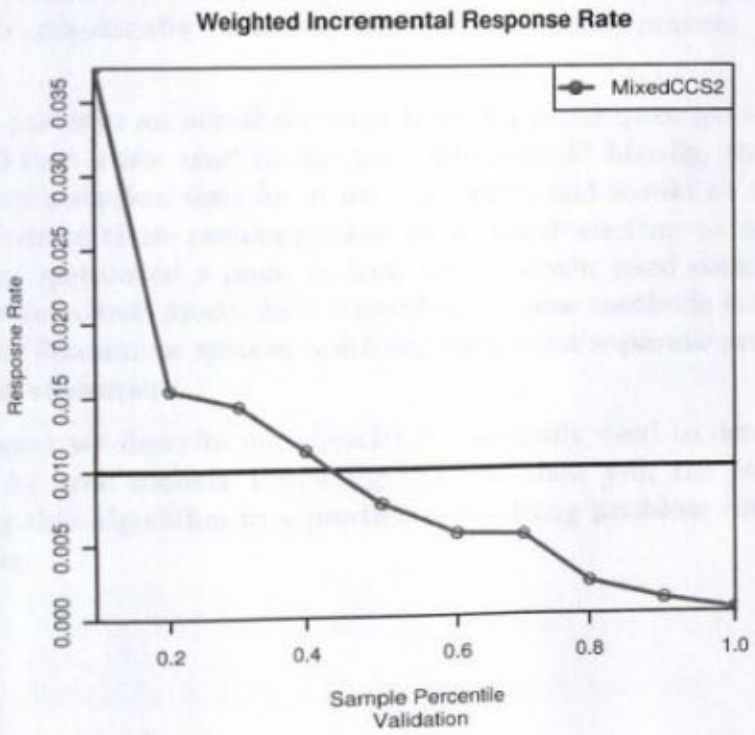
The resulting story for the validation sample is very similar to the one for the estimation sample. The `LinearCCS` model does best at first (but only slightly better than the `LogCCS` and `MixedCCS2` models), and then underperforms the other three models. The major difference between the two samples is for the `MixedCCS` model, which now performs worse the other three models for the first 10% of the database, but outperforms the other models from 50% to 70% of the sample. Finally, the performance of the `LogCCS` and `MixedCCS2` models is nearly identical. However, the `MixedCCS2` model is preferred to the `LogCCS` model since it has fewer estimated coefficients (a property known as "parsimony"), and more parsimonious models tend to have better prediction capabilities across multiple new samples of data. Overall, making a decision of which model to select as the "champion" is not easy in this circumstance. Generally, in evaluating models, we place more weight on the ability to predict

the "best" prospects (suggesting the selection of the `LinearCCS` model as the champion). However, elsewhere, this model performs quite a bit worse than the other three candidates. Almost the reverse situation holds for the `MixedCCS` model. From the point of view of consistent prediction capabilities, both the `LogCCS` and `MixedCCS2` models are a good choice as the champion model, with the `MixedCCS2` model being the better choice for reasons of parsimony. Consequently, for the purpose of this tutorial, we will select the `MixedCCS2` model as the champion.

7.

Finally, create an incremental response rate chart for the `MixedCCS2` model using the validation sample. To do this, again use the pull-down menu option **Assess** → **Graphs** → **Lift charts....** This time **select only MixedCCS2 for the "Select One or More Models" option, select CCS for the "Select a Single Data Set" option, choose incremental response rate as the lift chart type**. Again, the remaining fields should already contain the correct information. Once you have completed filling in the Lift Chart dialog box, press **OK** and the incremental response rate chart shown in Figure 6.12 should appear in the R graphics window. Since we have now sorted the

Figure 6.12: The Incremental Response Rate Chart for the Validation Sample



donors by the likelihood of joining the Monthly Giving Program, we can see that targeting the most likely donors first will give good incremental response rates (3.7% for the first tenth of donors targeted), and the rates will fall off as we target progressively less likely donors. If we know the costs of contact and the average dollar amount of donations, we can use these incremental response rates to calculate the costs and expected donations associated with contacting more and more donors; then we can choose a profit maximizing level of contact. This is good, but we may be able to do better by using different tools and fine-tuning our procedures.

Chapter 7

Tree Models

One method of predictive modeling, which we call tree models, but are also commonly referred to as decision trees or simply trees, generates a set of if-then rules that allow the use of predictor variables to predict target variables or make a decision. As a simple example of these types of decision rules, consider a rule used by a credit card company deciding whether to issue a Platinum Card to an applicant:

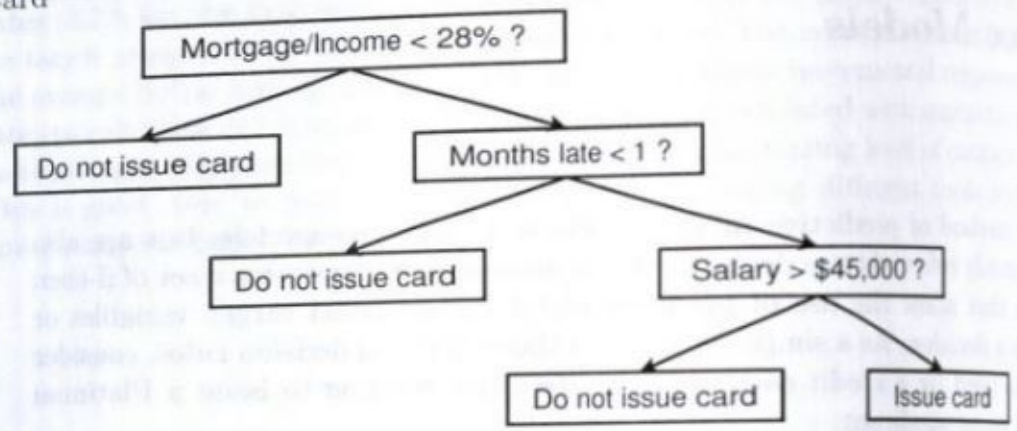
- "If monthly mortgage to income ratio is less than 28% and
- months posted late is less than 1, and
- salary is greater than $45,000,
- then issue a platinum card."

The set of if-then rules for making the decision to issue an applicant a credit card can be graphically portrayed using a tree representation, as shown in Figure 7.1.

Figure 7.1 portrays an actual decision tree. A natural question to ask is how were the if-then rules used in the tree determined? Ideally, they would be derived from historical data for other customers, and would be based on the characteristics of those customers that determined whether or not a particular customer generated a profit or loss for the credit card company. This is precisely what a tree model does. Specifically, these methods infer the set of if-then rules (known as *splits* or *splitting rules*) that separate profitable from unprofitable customers.

In this chapter we describe one algorithm commonly used to determine splitting rules for tree models. Following this, we show you the steps involved in applying this algorithm to a predictive modeling problem using R and R Commander.

Figure 7.1: A Tree Representation of the Decision to Issue a Platinum Credit Card



7.1 The Tree Algorithm

To create a tree the algorithm uses an estimation data set with historical information on both predictor and target variables. The target variable is typically binary, which in this case might take the value of "Bad" (to indicate that "this customer has proven to be a costly cardholder") or "Good" (to indicate that "this customer has proven to be a profitable cardholder"). The algorithm searches for decision rules based on the predictor variables (mortgage to income ratio, months late, salary) to progressively divide the cases in the data into smaller groups. The criterion used at each split is that each smaller group has a higher concentration of either "Bad" or "Good" than the preceding larger group. We say that each subset is more homogeneous, or pure, than its parent set. The final groups, called leaves, will have either mostly good or bad cardholders. The decision rules can then be used to sort new applicants who currently do not have a platinum card. The decision rules determine whether the applicant is likely to be a good or bad customer based on the leaf to which they are assigned, and provide the corresponding decision as to whether or not to issue a card.

It may be helpful to compare tree models with logistic regression:

1. Both logistic regression and tree models require historical data containing both predictor and target variables.

2. Both regression and tree algorithms use the historical data to create (i.e.,

calibrate) a model that approximates the relation between the predictor and target variables.

3. The regression model is an algebraic formula, while the tree model is a sequence of if-then rules.

4. Either model, once generated, can then use predictor variables from a new case and "predict" the likely value of the target variable.

5. The logistic regression model calculates the probability of a target variable taking the value "Good" for a particular case (e.g., customer). This probability can then be used to split the data into two groups using only the predictor variables. For example, we can separate the data into cases that are more likely to have a target value of "Good" from those that are more likely to have a target value of "Bad" by splitting at the probability of 0.5. The tree model can directly assign a new case (e.g., a customer) to a group (leaf) that is more likely to have either a "Bad" value or a "Good" value for the target variable. The probability of a new case actually being a "Good" (or "Bad") is given by the proportion of "Good" (or "Bad") customers in the assigned group (leaf) in the estimation data set.

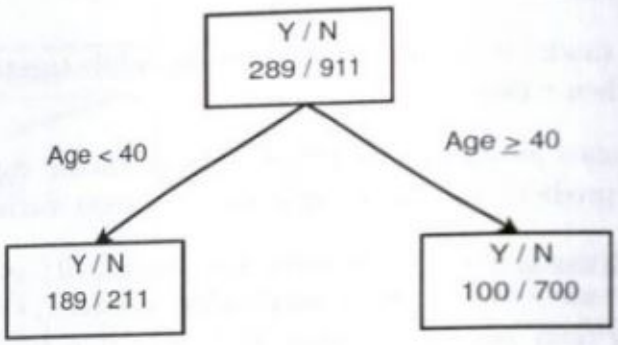
Tree models have both advantages and disadvantages over other types of predictive models. Interpretation of a calibrated tree model is very simple and intuitive, which makes it particularly useful for presentations to non-technical audiences. It is also more flexible than regression models in its assumptions about model structure, easily handles large numbers of variables, and is less affected by missing values. These features make it a good tool for initial exploration of a database. However, in our experience, a regression model will pick up weaker (but still meaningful) effects; and with enough data, the right variable transformations, and careful management of missing values, will fit and predict as well (and typically) better than a tree model.

Tree model algorithms vary in a number of characteristics, including the type of target variable they handle and the branching complexity. We will use the *rpart* (which stands for *recursive partitioning*) algorithm (Venables and Ripley, 2002), and confine ourselves to binary target variables and binary branches. The *rpart* model is also known as the Classification and Regression Tree (CART or C&RT) model (Breiman et al., 1984).

7.1.1 Calibrating the Tree on an Estimation Sample

At each stage of tree "growth," the rpart algorithm searches all possible split points of all possible predictor variables to find the one that splits the cases

Figure 7.2: A Three-Node Tree of Potential Bicycle Purchases



into the most homogeneous (or "pure") subgroups. This requires a measure of purity so that all possible splits can be compared. The measure we will use in rpart is what is known as the Gini index. The following example illustrates the calibration process.

Suppose a bicycle shop has collected information on adult customers entering the store. This information includes the customer's age, the number of children he or she has, and whether or not he or she purchased a bicycle. The owner of the shop wishes to quantify the relation between a bicycle purchase — the target — and age and children — the predictors. The data set has 289 buyers and 911 non-buyers, or 24% buyers out of the total. This information is contained in the variable "Buy" which takes the values "Yes" or "No." Using the decision tree strategy of searching for binary splits, we would start generating trial splits. For example, a trial split at customer age 40 produces 400 customers who are under 40, and 800 who are over 40, with the following proportions of buyers and non-buyers:

Buy	Age < 40	Age ≥ 40
Yes/No:	189/211	100/700
Total in branch:	400	800
	(47% Yes)	(12.5% Yes)

We can write this as a tree with three *nodes* (Figure 7.2). Leaves are the final nodes in a calibrated model.

This looks like an improvement—the concentration of buyers in the under-40 group, 47%, has doubled over the original concentration across all customers, while the 40-and-over group has become much more "pure" than the original group (at 12.5% buyers compared to 24% for the group overall). In order to automate the decision as to the best split, we need a single number that we

can use to compare the improvement across all possible splits, and use that number to pick the best split. As a guide for finding such a measure, a 50–50 split is the most impure a node can be, and a 0–100 (or 100–0) split is least impure. The Gini index provides such a measure of the change in impurity resulting from a split of the data. The Gini index is calculated as follows for the bicycle shop example:

Define P_L and P_R as the proportions of the original node in the left and right nodes, resulting in

$$P_L = 400\ /1200 = 0.333$$

$$P_R = 800/1200 = 0.666.$$

Define the proportion of one type (Yes) within the left and right subnodes as p_L and p_R, which gives

$$p_L = 189/400 = 0.4725.$$

$$p_R = 100/800 = 0.125$$

Define the Gini index as

$$P_L(p_L)(1 - p_L) + P_R(p_R)(1 - p_R)$$

$$= 0.333 \times 0.4725 \times 0.5275 + 0.666 \times 0.125 \times 0.875$$

$$= 0.154.$$

We can calculate this single number for all possible splits on all possible predictor variables. To build some intuition for this index, note that when p_L or p_R are 0.5 (the most impure, 50–50, split) the terms $(p_L)(1 - p_L)$ and $(p_R)(1 - p_R)$ are largest, having a value of 0.5 times 0.5 = 0.25. When p_L or p_R is one or zero, the most pure cases, then these terms are smallest, having a value of zero. These terms are then combined into a weighted average, with P_L and P_R as the weights. The most pure subgroups have the smallest Gini index.

For the bicycle shop, there are many possible splits on the age variable, and one possible split on the kids variable. Table 7.1 provides the Gini index for six possible splits, five for the age variable and the one possible split for the kids variable.

The first split in the tree would be on age, at a value 40 (since this has the lowest associated Gini index). All of those customers under 40 go into one group, and all of those 40 or older into the other group. In this instance, the split is driven by the combination of the size of the over-40 group (two-thirds of potential customers entering the store) and the high degree of purity (12.5%) associated with this group after splitting. The under-40 group is actually less pure, but it is half the size of the 40-and-over group. The same procedure would then be followed to split each of these subgroups into two further groups, most likely splitting the under-40 group given its high impurity level at the next split. The process continues until a stopping rule is met at each node.

7.1.2 Stopping Rules and Controlling Overfitting

A node is not split again, and becomes a leaf, when a stopping rule is met. Some stopping rules are:

- All cases in a node have identical values for predictors.
- The depth of the tree has reached its pre-specified maximum value.
- The size of the node is less than a pre-specified minimum node size.
- The split at a node results in producing a child node whose size is less than a pre-specified minimum node size.
- The node becomes pure, i.e., all cases have the same value of the target variable.
- The maximum decrease in impurity is less than a pre-specified value.
- The tree exceeds a pre-specified level of complexity as measured by a complexity parameter, cp.

Table 7.1: Possible Splits for the Bicycle Shop Customer Data

Possible Split	Gini Index Value
Kids="no," Kids="yes"	0.213
Age<30, Age≥30	0.190
Age<40, Age≥40	**0.154**
Age<50, Age≥50	0.168
Age<60, Age≥60	0.208
Age<70, Age≥70	0.219

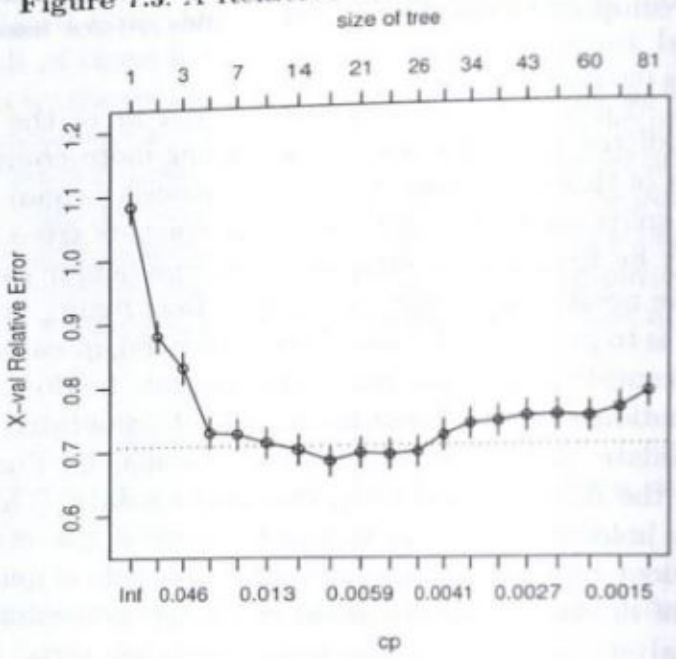
In our R Commander implementation of *rpart*, the stopping rule under the analyst's control is the complexity parameter. Other rules are not used, or left at their default values.

In regression modeling we can get an increasingly better fit of the data by incorporating more predictor variables, and by including more complex and flexible transformations of those variables. With trees models we can progress with increasingly finer splits to grow larger and more complex trees. In both cases we will eventually be fitting noise unique to that particular estimation data set. As a result, we need criteria that prevent us from fitting noise. The approach used by *rpart* is to grow a large (over-fitted) tree and, at each stage of growth and increasing complexity, to automatically generate holdout samples as subsets of the estimation sample. These automatically generated holdout samples are used to validate each of the sequence of tree models. That is, the model is calibrated on the data not including the holdout data. This model is then used to classify holdout data. The validation error is the misclassification rates in the holdout data. This *cross-validation* procedure generates a sequence of measures of fit, the cross-validation error, for increasingly more complex trees. The analyst can then use the cross-validation error to *prune* those branches of the tree that appear to be fitting noise.

As indicated above, the parameter that the analyst actually sets in rpart to determine which branches to keep and which to prune is called the complexity parameter, designated cp. In one way this is an unfortunate label, since the complexity (or depth) of the tree actually increases as the cp decreases — it might more logically have been called a simplicity parameter. The algorithm outputs a table and a plot that gives the number of splits, the cross-validation error, and the complexity parameter at different levels of the tree. As with validation error generally, it is typical to observe the error decrease with increasing complexity, stabilize, and then start to increase again as the model becomes overfitted. The table or plot can then be used to choose the value of cp that gives the desired degree of fit. The algorithm is then rerun with this cp value entered to produce the final model.

Figure 7.3 shows a cross-validation error plot. The vertical axis gives the relative cross-validation error, where "relative" means relative to the root node (a trunk with no branches). This results in a natural fit measure for a particular tree size as a percentage of the error that would occur if no splits were used. This normalization is accomplished by dividing the error for each tree by the error of the root node. The horizontal axis is labeled by both the complexity parameter (along the bottom), and the number of nodes (along the top). As the number of nodes, and hence complexity, increases, the cross-validation error decreases, levels out, and then increases as the model becomes overfitted. From this plot the appropriate value of cp can be chosen to generate the final

Figure 7.3: A Relative Cross-Validation Error Plot

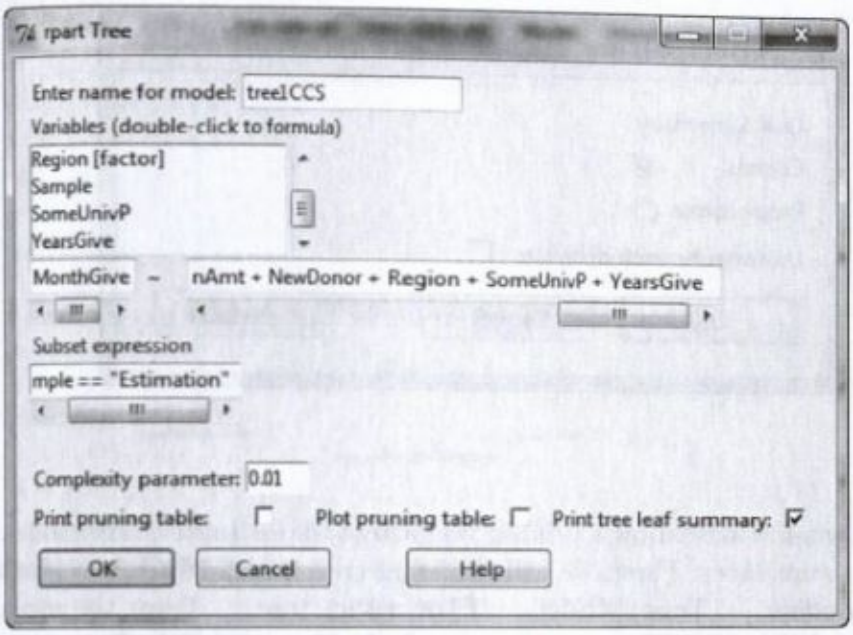


model. One choice is at the minimum error, corresponding to 17 nodes and a cp value of about 0.008. However, there are several other values nearby that are very similar, and each value is an estimate, so that the notion of being statistically equivalent applies. In an analogy to a confidence interval, the algorithm also calculates the standard deviation of the relative error at the minimum value, and plots a horizontal line that is one standard deviation away from the estimated value. Any point below that line is said to be statistically equivalent to the minimum point. If we value parsimony, as most modelers do, we would choose the simplest tree that has a cross-validation error statistically equivalent to the minimum error, which in this case corresponds to the tree with 14 nodes.

7.2 Trees Models Tutorial

This tutorial will demonstrate the use of the *rpart* (*r*ecursive *part*itioning) algorithm to generate a tree model for the Canadian Charitable Society data, and will introduce you to some of strengths and weaknesses of tree models in comparison to logistic regression models.

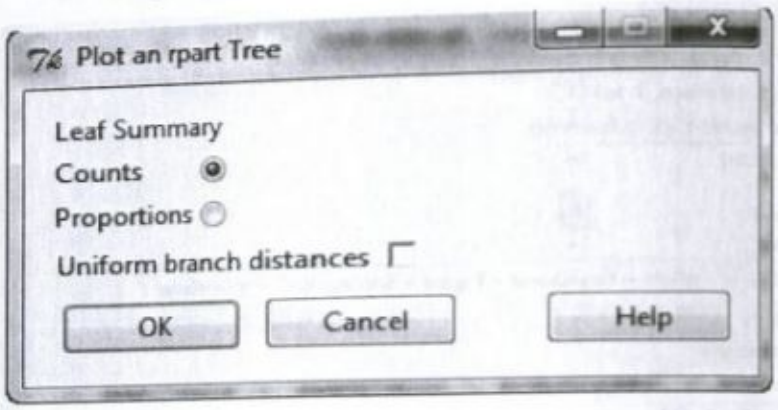
Figure 7.4: The rpart Tree Dialog



1.

Start R and R Commander, load the BCA package and the CSS data set, and **create 50/50 estimation and validation samples** as you did in the logistic regression tutorial. **Click on Model → Machine Learning Models → Tree Model → Train rpart tree...** to bring up the dialog box shown in Figure 7.4. **Enter `tree1CCS`** as the name for the model. The target and input variables are selected in the same way as in the regression model: **Choose `MonthGive`** (the factor) **as the target, and all of the remaining original** variables (not the new computed variables, if you have retained them from the last tutorial, nor Sample) as predictors. As usual, we will wish to validate our estimated model later, so **set the sample to Estimation**. The *complexity parameter* determines a cutoff level for how weak of predictors you wish to include in the tree, and thus how large a tree you wish to grow. To include weaker effects in the final model, the parameter is set closer to zero. As usual with data mining, including progressively weaker effects in the model means that at some point weak effects will be unique to this particular data set (noise), and will be useless when applied to a new data set. A reasonable value requires some effort (and a bit of judgment) to determine, which we will explore later. For now, **leave the complexity parameter at the default value of 0.01, uncheck the Print pruning table: and Plot pruning table:** options. Click **OK**.

Figure 7.5: The rpart Tree Plot Dialog



2.

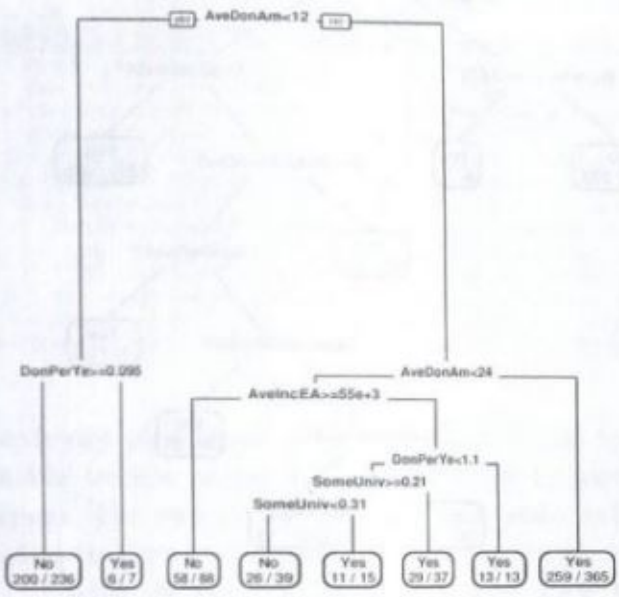
The output window will show a printed version of the estimated tree model. We will return to this later. First, we will plot the tree. Select **Model** → **Machine Learning Models** → **Tree Model** → **Plot rpart tree...** from the menu, to bring up the dialog in Figure 7.5. **Uncheck Uniform branch distances** and click **OK**, and the plot in Figure 7.6 should appear.

3.

The plot of the tree in Figure 7.6 shows the approximate split values that the model finds. Precise values can be seen in the text output. The best first split of the data to concentrate the target `MonthGive` is at the `AveDonAmt` of about $12 (the text output shows it more preciesly as $11.83). The convention in the plot is that those individuals where the statement in blue is *true* go to the left branch. In this case, those with **AveDonAmt** $<$ **12** go to the left branch, and individuals with an average donation amount greater than $12 go to the right branch. The vertical length of the branches roughly indicates the relative strengths of the relationships between the splitting variables and the target variable. In this case, the first split is by far the strongest.

The final nodes are called *leaves*. The right branch splits again on AveDonAmt, with those indivduals donating more than about $24 moving to a final leaf. This leaf is labeled "Yes" which indicates that the majority of individuals joined the monthly giving program. The specific counts, 259/365, are then given: 259 individuals joined the program out of 365 who had both preceding conditions satisfied. This makes good intuitive sense: if the average donation amount is larger, donors are more likely to join the monthly giving program. The far left branch is split again on Donations per year, with its left branch taking those individuals with the true condition, specifically with donations

Figure 7.6: The CCS Tree Diagram Where Branch Length Indicates Importance



per year greater than 0.095. The far left-hand leaf labeled "No" indicates that the majority of individuals in that leaf did not join the monthly giving program. The numbers provide the counts: 200 out of 236 in this group did not join the program. These 236 individuals satisfy the previous two conditions, that is, they give more than 0.095 times per year (i.e., about once every 10 years) *and* had an average donation amount under 12.

Note that very small and very large numbers are given in scientific notation, e.g., "e+3" means "times 10 raised to the power of three," or times 1000.

Now we can use the tree model to predict the likelihood of a new individual donor joining the monthly giving program. All we need is the value of the predictor variables for that individual. Then we can follow the path through the tree, with the prediction for that individual being the label on the leaf that the individual ends.

Exercise: Using the calibrated tree model, predict whether or not the following new donors are likely to join the monthly giving program:

Donor #1: Ave DonAmt = \$19, AveIncEA = \$85,000

Donor #2: AveDonAmt = \$9, DonPerYear = 0.04

Donor #3: AveDonAmt = \$22, AveIncEA = \$52,000, DonPerYear =0.04, SomeUniv = 0.25

Figure 7.7: The CCS Tree Diagram with Uniform Branch Sizes

4.

Plot display options: Drawing the tree with the branch length indicating relative strength of the relationship is initially useful, but it usually crowds the remaining branches, making them difficult to read. The tree can be drawn with uniform spacing to make reading the labels easier. To do this, **bring up the Plot rpart tree training dialog box** again but this **time leave "Uniform branch distances"** checked. **Press OK**. Figure 7.7 shows the redrawn tree.

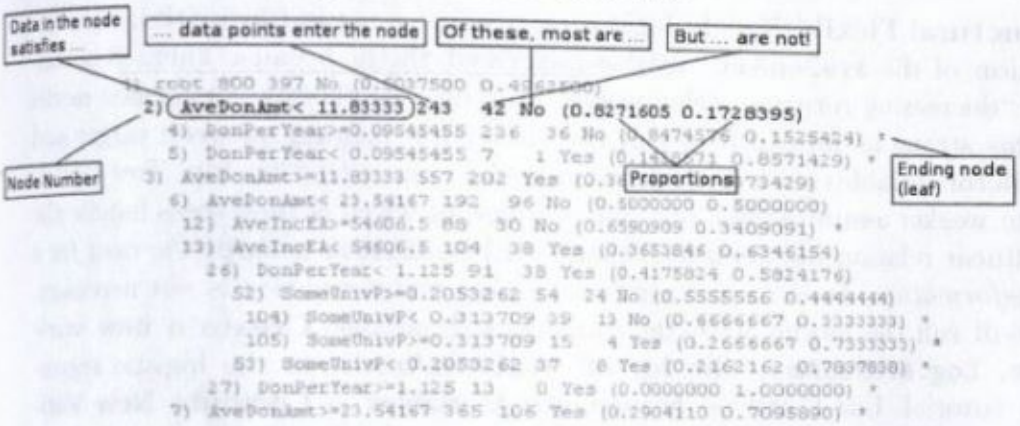
Recall that Windows users can save plots by **clicking on the History menu** at the top of the graphics window, and **selecting "Recording."** If you do not record the plots, each new plot will overwrite the previous one, losing it. You can also redo the plots with each leaf showing the proportions of the two values of the target variable, by **selecting "Proportions"** in the plot dialog. Try this now. As long as the previous plot was saved, you can **switch between the two plots using the Page Up and Page Down keys**.

Exercise: In one sentence, explain the meaning of the two proportions given in a node.

5.

Tree Table: In the R Commander output window, **scroll back to the output from the tree model**. The text output gives a list of the variables used

Figure 7.8: The Printed Tree



for the given complexity parameter, a table version of the tree, and a "pruning table." The table version of the tree is shown in Figure 7.8, along with explanatory captions. The root node does not have split information, so the captions are directed to the second node. To read the table, use the captions, and fill in the "..." with the entry in the table. The node numbering starts with "1)" for the root, which in the table includes all those individuals entering the first split. To help follow the branching system, the convention is that the left branch in the tree plot has a node number that is double the preceding node, and the right branch is double plus 1. Hence the branches from node 3 are nodes 6 and 7. This makes the node number increase rapidly, but it does aid navigation of the tree table and comparison with the graphic tree. Node 2) in the table thus corresponds to the first node in the left branch of the split. Notice that the condition given in the table is for individuals ENTERING node 2), which in the tree is the label on the preceding node. Thus, node 3) has the opposite condition from node 2). The main piece of additional information beyond what is in Figure 7.7 is the proportions of the binary variable in the node. Compare the information in nodes 2), 3), 4), and 5) with Figure 7.7.

Exercise: The tree *plot* only gives the count information for leaves. The Tree *table* gives count information for all intermediate nodes as well. Using the tree table, determine what the count information would be on the node in the tree plot that is labled "AveIncEA>=55e+3" and for the node labled "SomeUniv>=0.21"

In the next two steps we will explore tree models' characteristics of structural flexibility and the effect of correlated predictors. Following that we will learn how to assess and control overfitting in tree models, before going on to model comparisons using lift charts.

6.

Structural Flexibility: In logistic regression, a natural logarithm transformation of the `AveDonAmt` variable improved the fit because there is a convex "decreasing returns" relationship, and because the untransformed model makes strong assumptions about the structural relation between target and predictor variables that does not include the convex relationship. Tree models make weaker assumptions, one consequence of which is that trees handle the nonlinear relation between `AveDonAmt` and `MonthGive` *without the need for a transformation*. To demonstrate that the log transformation is not necessary, we will run the model with the transformed variable. **Create a new variable, `Log.AveDonAmt`, the log of AveDonAmt** as in the logistic regression tutorial. Use **Data → Manipulate Variables → Compute New Variable**. **Rerun the tree with the same parameters as before** except using `Log.AveDonAmt` but **not including `AveDonAmt`.** Since the dialog box has the previous model parameters, you can quickly make the change by placing the cursor in the input variable box, using the right arrow on the keyboard to move to `AveDonAmt`, and just typing the prefix "`Log.`" to it. **Name** the model **trlogADA,** and **click OK**. **Plot** the new tree using the default plotting parameters. The result is shown in Figure 7.9. For Windows users only, you can select **History** at the top of the Graphics Window menu, and use **Previous** and **Next** to go back and forth between the two trees (or use the Page up /Page down keys). Note that all that changes in Figure 7.9 is the value of the split. It is the log of the value of the split in Figure 7.7. The same individuals end up in the same leaves.

Generally, any *monotonic* transformation (one which does not change the ordering of the values) of a variable will not change the resulting tree model. While this flexibility in handling nonlinear relations between predictor and target is a strength of tree models, particularly for exploring data, it is difficult to interpret nonlinearities beyond the first or second split, so the various plotting methods and analyses discussed in previous tutorials remain useful for exploring data.

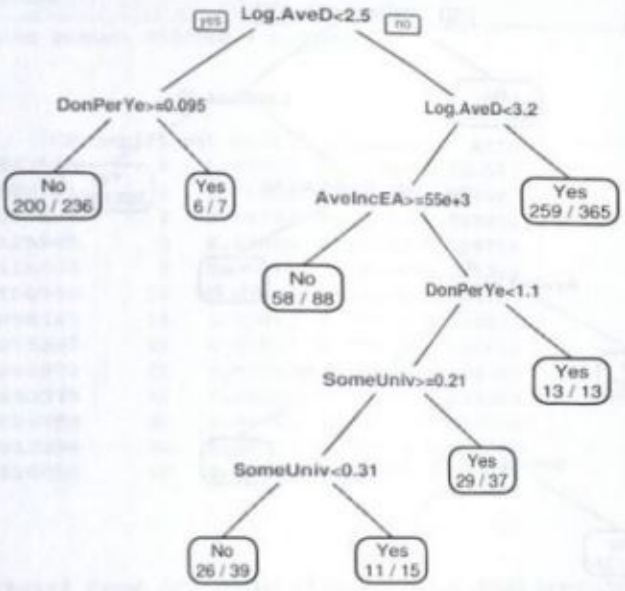
Delete the Log transformed variable (using the Data→ Manipulate Variables → Delete Variables from the Data Set menu), as we will no longer need it and do not want to accidentally include it later on.

7.

Correlated predictor variables

The dominant feature of Figure 7.7 is that the first split on `AveDonAmt` gives a much greater improvement in fit than any subsequent split. From the logistic regression tutorial, we know that this variable and `LastDonAmt` are

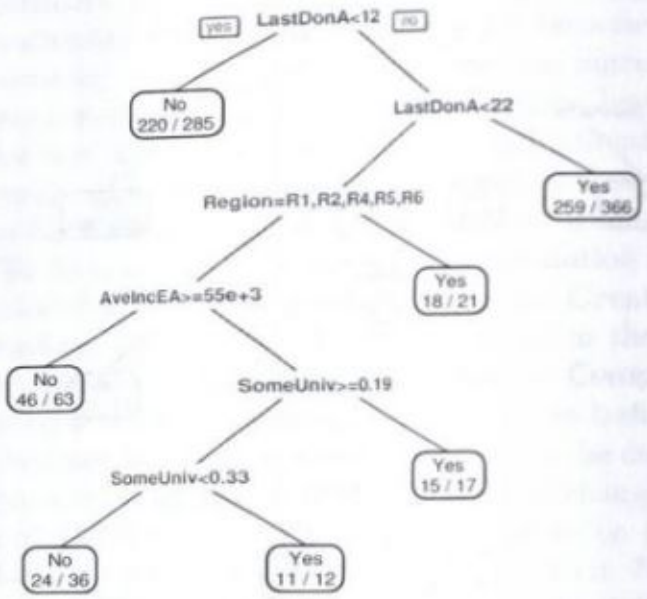
Figure 7.9: The Log Transformed Average Donation Amount Tree



correlated with each other. That means that they would separate donors very much the same way, so that once `AveDonAmt` is used, there is nothing left for `LastDonAmt` to do, and it will not be used. To highlight this, we will rerun the model without `AveDonAmt`. As before, **bring up the tree model training dialog**, which should contain the last model, and **delete** `Log.AveDonAmt` from the command, so that now there is no average donation amount information used as a predictor. **Change the name of the model to TreeNoADA. Run it and then plot it,** using the default values. Figure 7.10 shows the resulting tree. `LastDonAmt` is now able to have a large impact because `AveDonAmt`, which has nearly the same relation with `MonthGive`, is no longer available. What this generally means is that with highly correlated predictor variables one or the other will appear as effective predictors in the tree model, but not both.

In the logistic regression tutrorial we found a similar effect in the sequence of logistic regression models. In some models one of these two variables did not even appear as significant, even though they are both individually quite strongly related to the dependent variable. Even in the final model, where both appear, neither is indicated as extremely strongly significant (in fact, `AveDonAmt` appeared to be insignificant). This is a general consequence of correlated predictor variables in the same model: they will reduce the apparent impact of each other on the target. In regression models this happens because the estimate of the variables' variances is increased (a phenomenon known

Figure 7.10: Last Donation Amount Tree



as "variance inflation"), therefore lowering their significance (indicated by increasing their p-values). In tree models, whichever variable is used first (e.g., `AveDonAmt`) takes up much of any effect of the second variable, so it is not used. This mainly causes difficulty in *interpreting* models, and can complicate model selection. The good news is that including variables that are correlated with each other will not diminish the *predictive* usefulness of either tree or regression models, as long as the estimated model is applied to new predictor data that has a similar origin (and hence structure) as the original data on which the model is estimated.

8.

Overfitting

As indicated earlier, letting a tree grow too large will result in overfitting the estimation sample. To avoid this, we need to have a stopping rule that prevents us from fitting noise. For trees, the approach used by R is to grow a large (overfit) tree, and then prune it back using a *cross-validation* measure. The CCS data, because of the two strong predictors, `AveDonAmt` and `LastDonAmt`, does not lend itself to demonstrating the logic of the procedure (since the method selects a model with only a single split on one of the two variables), so we will run a tree model without these two strong predictors to demonstrate the use of cross validation and pruning to control overfitting. This time we want to print the cross-validation or "pruning" table and associated plot.

Figure 7.11: The CCS Pruning Table

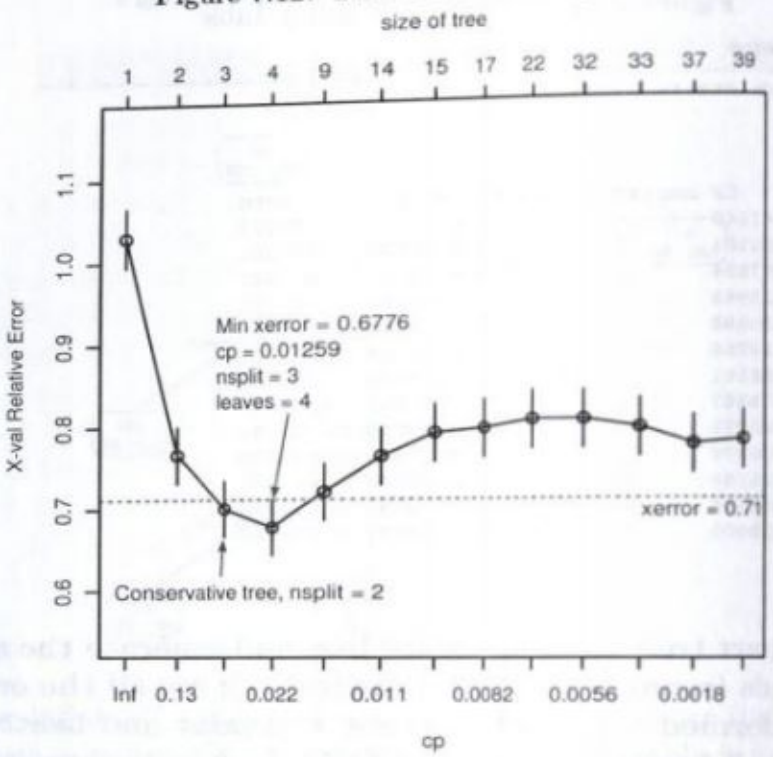
Output Window

```
Root node error: 397/800 = 0.49625

n= 800

         CP nsplit rel error  xerror     xstd
1  0.2342569      0   1.00000 1.06045 0.035573
2  0.0680101      1   0.76574 0.76574 0.034581
3  0.0377834      2   0.69773 0.70025 0.033925
4  0.0125945      3   0.65995 0.68262 0.033719
5  0.0115869      8   0.59446 0.73804 0.034324
6  0.0100756     13   0.53652 0.76826 0.034603
7  0.0088161     14   0.52645 0.77582 0.034667
8  0.0075567     16   0.50882 0.77834 0.034688
9  0.0062972     21   0.47103 0.79093 0.034789
10 0.0050378     31   0.40302 0.79345 0.034809
11 0.0025189     32   0.39798 0.78338 0.034730
12 0.0012594     36   0.38791 0.79093 0.034789
13 0.0010000     38   0.38539 0.79597 0.034828
```

Bring up the rpart tree training dialog box and generate the appropriate commands (rename the model (`noAorLDA`); use all the *original* – not log transformed – variables except `AveDonAmt` and `LastDonAmt`; set cp = 0.001; all boxes checked; and OK). Look for the pruning table in the output window — you may have to scroll up to see it. The pruning table will be similar to Figure 7.2. The last two columns, xerror and xstd, will be slightly different, and will vary somewhat with every run. The table shows how many splits would have been allowed for various values of the complexity parameter, from no splits, down to the level of cp you set in the dialog box (in this case 0.001, and 38 splits). As mentioned earlier, "complexity parameter" is a rather unfortunate name, as the complexity of the tree *increases* as cp *decreases*.

At each split the improvement in the purity of the tree is given as a fraction of the root impurity, under rel_error. The relative error of the root node (the nsplit= 0 case, in the first line) is set to one. Each subsequent split is chosen as the one that most improves the purity of the tree, and its error continually decreases. This is analogous to the typical increase in R^2 in a multiple linear regression model, or the McFadden R^2 in logistic regression, as more variables are added. Fit always increases, error decreases, and overfitting will occur at some point. To control overfitting, the tree routine also automatically and repeatedly splits the input data set into its own estimation and validation samples, re-estimates the model on the new (smaller) estimation sample, and then uses the model to predict the target in the validation sample. The error between the known and predicted target in these internal validation samples

Figure 7.12: The CCS Pruning Plot



is averaged over several iterations, and is called the *cross-validation error*, and labeled *xerror*. The table shows the common pattern of error measures that are adjusted to compensate for overfitting: xerror decreases, levels out, and then increases, with the minimum indicating approximately where overfitting starts. This is analogous to the behavior of adjusted R^2.

The table has a minimum xerror value of 0.67758 (your may have a slightly different value) at nsplit = 3 and cp = 0.0125945. Not all splits are reported; intermediate splits that could not possibly be candidates for the optimal tree are left out.

In the graphics window, R has also plotted xerror as a function of the size of the tree in terms of the number of leaves (the number of leaves is equal to the number of splits plus one, and is given along the top of the plot as "size of tree"), shown in Figure 7.12. Note that the cp scale at the bottom may not be accurate, and should not be used (free open-source software is sometimes not polished); rely on the values in the pruning table. Recall that the precise value of xerror depends on how the algorithm divides the data into estimation and validation samples during cross-validation. Since this is done randomly, each run will give slightly different results, and your plot may be slightly different.

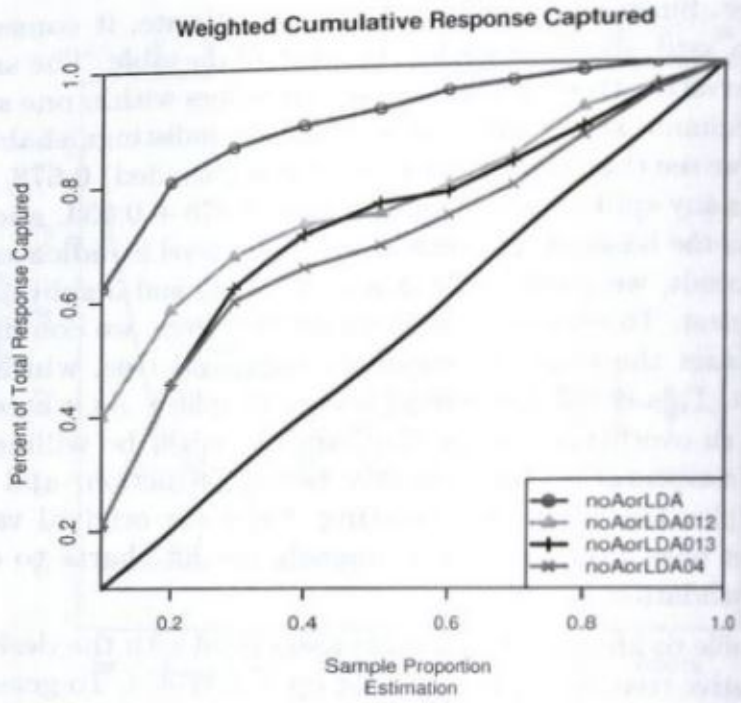
9.

Pruning the tree: Since each xerror value is an estimate, it comes with a standard deviation, xstd, shown in the last column of the table. The argument for the "best conservative tree" is that any xerror values within one standard deviation of the minimum xerror value are statistically indistinguishable. From the pruning table we see that the minimum xerror is (rounded) 0.678, and has xstd of 0.034. Thus any splits with xerror less than 0.678 + 0.034, about 0.71, are equally good on the basis of cross-validation. This level is indicated by the dotted line. As a result, we consider the 3 and 4 leaf (2 and 3 splits) trees as statistically equivalent. To choose between those two trees, we could add the criterion that we want the most parsimonious (simplest) tree, which will be easiest to interpret. This is the tree with 3 leaves (2 splits). As stated, this is conservative from an overfitting perspective, and we might be willing to give up a bit of easy interpretation for a slightly better prediction, and select a slightly more complex tree without overfitting. Since our original validation sample has not yet been used, we can ultimately use lift charts to compare these two on our validation sample.

Use the pruning table to identify the cp value associated with the desired tree. The most conservative tree, at nsplit = 2, is at cp = .0377834. To generate the conservative tree, rerun the model with any cp cutoff between that cp value and the next larger value (0.0680101) in the pruning table — let's choose 0.04. The next two more complex trees occur at cp values of 0.0125945 and 0.0115869. Setting cp values of 0.013 and 0.012 will give us those trees. **Rerun the model, changing only the complexity parameter to 0.04, and the model name to `noAorLDA04`. Plot and inspect the Rpart tree. Repeat this for complexity parameter values of 0.013 and 0.012**, renaming the models accordingly, and inspecting the plots. Note how the tree grows as cp decreases. Then select **Assess → Graphs → Lift Charts** from the menu. **Select the four noAorLDA tree models and the CCS data set** to see the improvement in the estimation sample. Enter **0.01** for the true response rate, **Yes** for the target, set the Subset expression to **`Sample == "Estimation"`**, and **`Estimation`** for the name of the sample. The result should appear as in Figure 7.13.

As expected, the more branches in the tree (the smaller its complexity parameter), the better the fit to the *estimation* sample. Next, plot the lift charts on the *validation* sample. **To do this, bring up the lift chart menu again, select CSS as the data, set the "Subset expression" to `Sample == "Validation,"` and label the plot Validation.** Set the remaining parameters as before. Figure 7.14 shows the lift chart. Unlike the last figure, this figure indicates that no tree model dominates the other models throughout the range of the data. In this particular case, the most conservative generally

Figure 7.13: CCS Estimation Weighted Cumulative Response



fairs the worst, and the advantage of simplicity in interpretation is not great. Every case will be different, but the large differences between models here and in Figure 7.13 are common, and due to overfitting the noise within the estimation sample by the more complex trees.

10.

Finding a good tree model: Since we are now most interested in good prediction, rather than clear demonstration of the method, we will bring the two strong predictor variables back into the model. **Include `AveDonAmt` and `LstDonAmt` with all of the other original variables (not log transformed) and set cp = 0.001. Name the model Full001 and run it.** The pruning chart in Figure 7.15 suggests that anything beyond three splits is overfitting.

Set the complexity parameter to 0.02 (a value between the 0.012 and 0.035, see Figure 7.15) **and rerun, naming the new model `Full02`.** This will give the minimum xerror model with three splits. **Plot the tree.** This time, we will explore whether going slightly more complex than indicated by the cross-validation plot might perform better if we make our judgment on our own validation sample instead, using lift charts. To get the next most

Figure 7.14: CCS Validation Weighted Cumulative Response

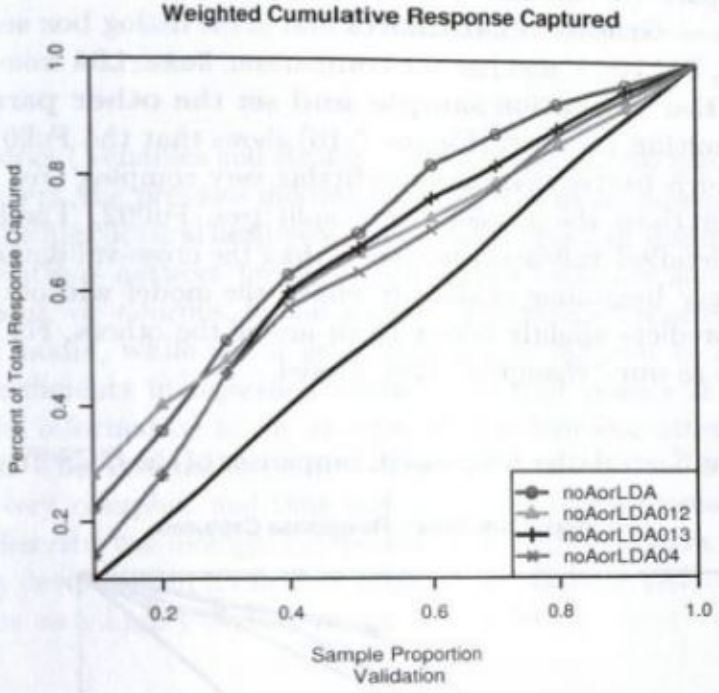


Figure 7.15: The Minimum Cross-Validation Error in the Pruning Table

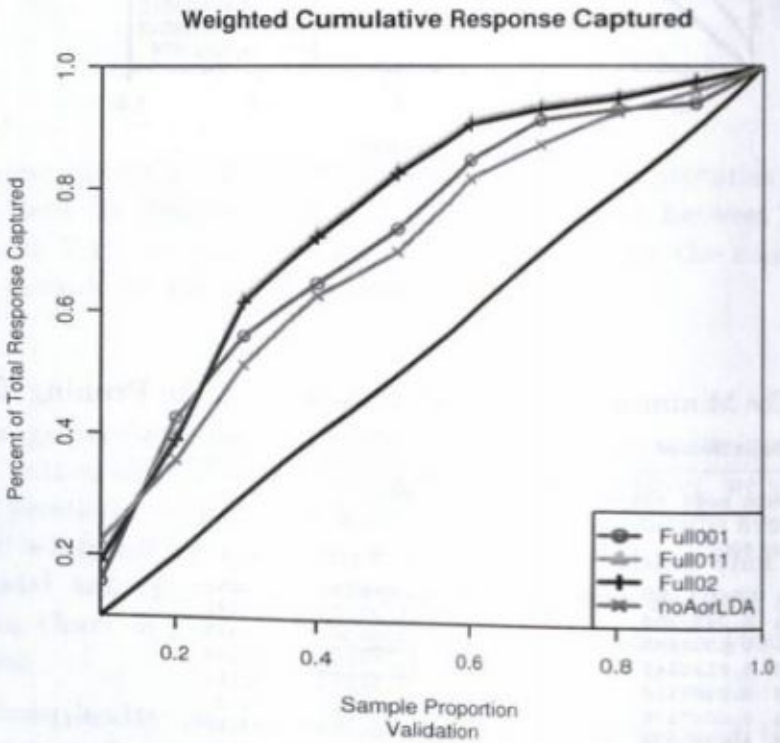
Output Window

```
Root node error: 397/800 = 0.49625

n= 800

          CP nsplit rel error  xerror     xstd
1  0.3853904      0   1.00000 1.07557 0.035541
2  0.0352645      1   0.61461 0.61965 0.032877
3  0.0125945      3   0.54408 0.57683 0.032203
4  0.0109152      4   0.53149 0.60202 0.032610
5  0.0067170      7   0.49874 0.61713 0.032839
6  0.0062972     12   0.46348 0.64484 0.033234
7  0.0050378     23   0.36272 0.64736 0.033268
8  0.0041982     25   0.35264 0.67506 0.033627
9  0.0025189     28   0.34005 0.69270 0.033838
10 0.0010000     31   0.33249 0.70025 0.033925
```

complex model with four splits, **set the complexity parameter** to 0.011, **and name the model** `Full011`. Plot the tree and note where the new branch is.

Now we will compare the lift charts of the various models on the validation set. Select **Assess → Graphs → Lift Charts** and in the dialog box **select the** `Full02`, `Full011`, `Full001`, and just for comparison, `NoAorLDA` from our earlier exercise. Use **the validation sample and set the other** parameters as before. The resulting lift chart (Figure 7.16) shows that the Full011, the 4 split tree, does much better than the overfitting very complex tree, Full001, and slightly better than the simpler three split tree, Full02. The lift chart provides a more detailed validation measure than the cross-validation. Interestingly, at the very beginning of the lift chart, the model without the two strong variables predicts slightly better than any of the others. For now, we will take `Full011` as our "champion" tree model.

Figure 7.16: Weight Cumulative Response Comparison of the CCS Tree Models



11.

Save your workspace and exit R and R Commander.

Chapter 8

Neural Network Models

Neural networks (Venables and Ripley, 2002) are a class of predictive models that complement the previous models studied. The main advantage of neural networks is their structural flexibility. They can capture an unlimited variety of functional relations between predictor and target variables simply by including more nodes, or neurons, in the model. Their main disadvantage is that a calibrated model, while useful for prediction, is difficult to interpret. The estimated coefficients in regression models and split points in decision trees provide useful information to the analyst even before any attempt at prediction is made. The internal structure of a calibrated neural network model, however, is very complex, and thus not amenable to interpretation. We will first briefly describe the biological inspiration for neural network models, which were initially developed in the field of artificial intelligence (AI), and then their interpretation as a highly flexible statistical model.

8.1 The Biological Inspiration for Artificial Neural Networks

Let's consider two variations on the general problem of aiming a projectile toward a target.

Ballistic Trajectory Problem #1: Field Artillery

An artillery gunner must set the angle of the gun barrel so that the ballistic trajectory of the shell ends on the target to be destroyed. To do this, she enters the distance to the target into a formula that gives the correct barrel angle. The formula is based on classical physics and known parameters such as the muzzle velocity of the shell V_0 and the force of gravity g. Figure 8.1 shows the formula and allows anyone to do this common calculation. More sophisticated variations would include air resistance.

Ballistic Trajectory Problem #2: The Jumpshot

A basketball player launches himself into the air, clear of an opposing player, and while airborne sends a ball on a ballistic trajectory toward a hoop 5 meters away. The balls slices cleanly through the hoop, as in Figure 8.2.

Figure 8.1: The Artillery Launch Angle Calculation

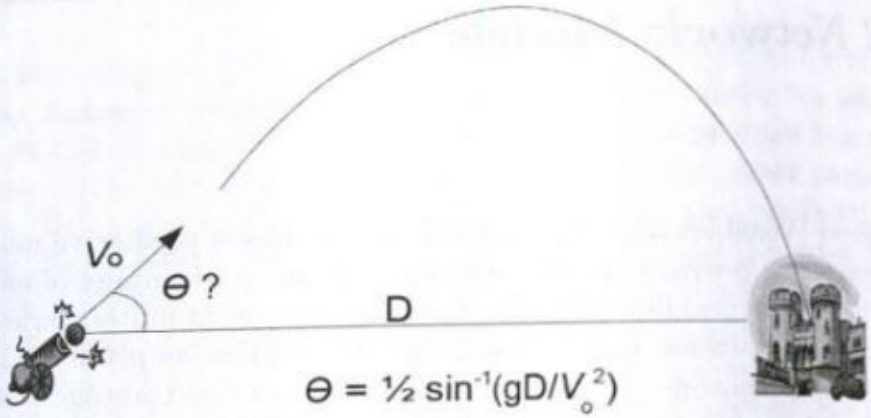
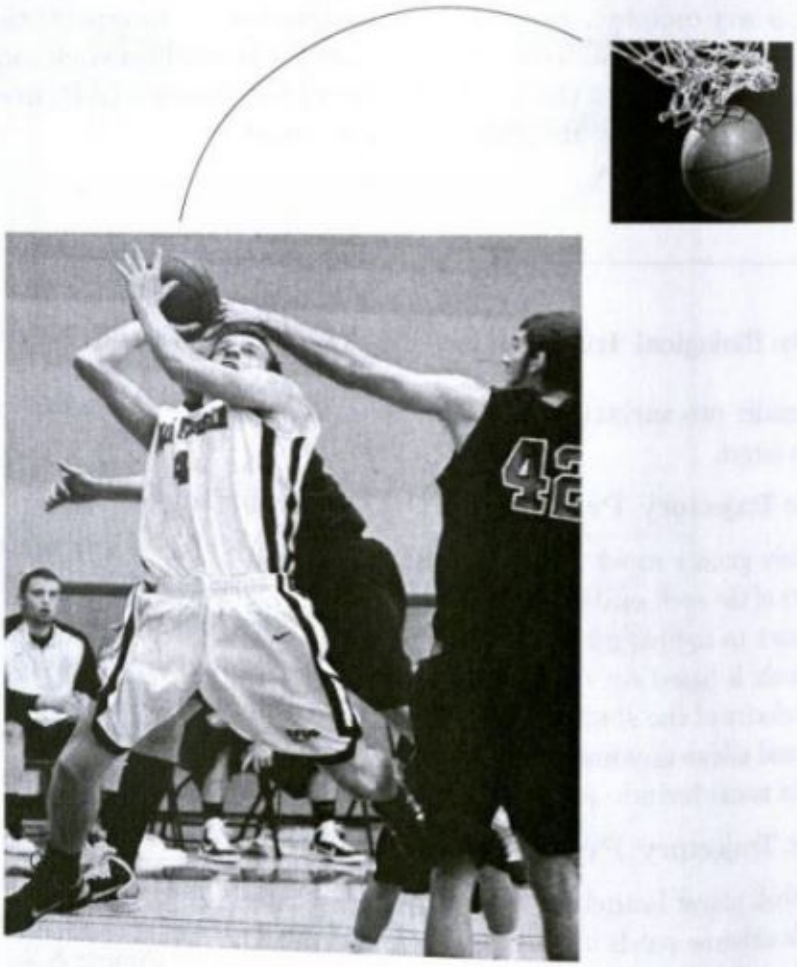


Figure 8.2: The Launch Angle "Calculation" in Basketball



The jump shot calculation is a vastly more complex problem than the artillery calculation. The equations of motion might, in principle, be written down, but there's not much point in the exercise. You can bet the player is not going to measure all of the parameters involved, let alone plug them into a formula, and then execute the necessary motions. His brain, however, must take in that information, process it to provide a predicted output, and give the necessary complex directions via his nerves to his muscles. This all occurs rapidly, automatically, and with a remarkable degree of predictive accuracy. When put in the context of the field artillery problem, which looks like it has some complicated equations, this is an absolutely amazing feat. Yet it has nothing to do with formulae from physics. Rather, it results from learning, by trial and error, over a period of time.

But just how does the brain, which is just a mass of soft, wet nerve cells in a soup of neurotransmitters, do this? That, of course, is the domain of neuroscience, the cognitive sciences, and computer science. Here, we will briefly explain a major insight on the functioning of neurons that led to intensive research in Artificial Intelligence (AI) and ultimately to the Artificial Neural Network (or ANN) we use as a predictive model today.[1]

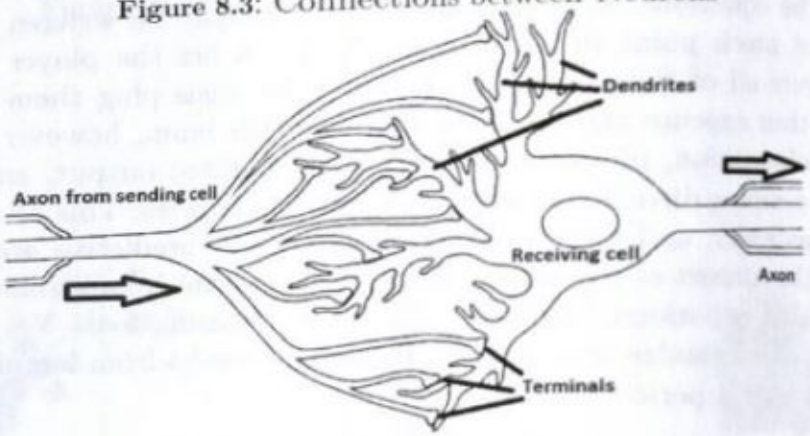
Neurons in the brain are linked together in a network. Figure 8.1 shows an abstraction of the connections between two neurons. Signals pass from the sending axon across the terminals to the dendrites of the receiving neuron. The neuron will "fire," that is, send a signal through its axon at the right, if it receives enough input from its dendrites. A simple model is that the receiving cell adds up all of the inputs it receives at its terminals from all of the other neurons it is connected to, and fires if that total exceeds a threshold level. That by itself won't do very much. In particular, it won't allow learning to occur. Learning means that behavior changes, and that means that the way the signals are transmitted through the network must change over time.

The additional piece that is necessary to make this work is that *the strength of the connections between the neurons is adjusted when the neurons fire.* More firing strengthens the connections, effectively reducing the threshold level required for the next neuron to fire. Less firing weakens the connections, effectively raising the threshold. Thus, the neuron changes as it is used. This provides a mechanism for memory, and learning becomes possible.[2]

[1]Generally attributed to Professor Donald Hebb in 1949, who was chair of the psychology department at McGill University at the time. "Hebbian Learning" (Hebb, 2002) is still the term used for various methods of adjusting the weights in artificial neural networks.

[2]From Hebb's 1949 book (Hebb, 2002): "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." This is more often paraphrased as "Neurons that fire together wire together."

Figure 8.3: Connections between Neurons



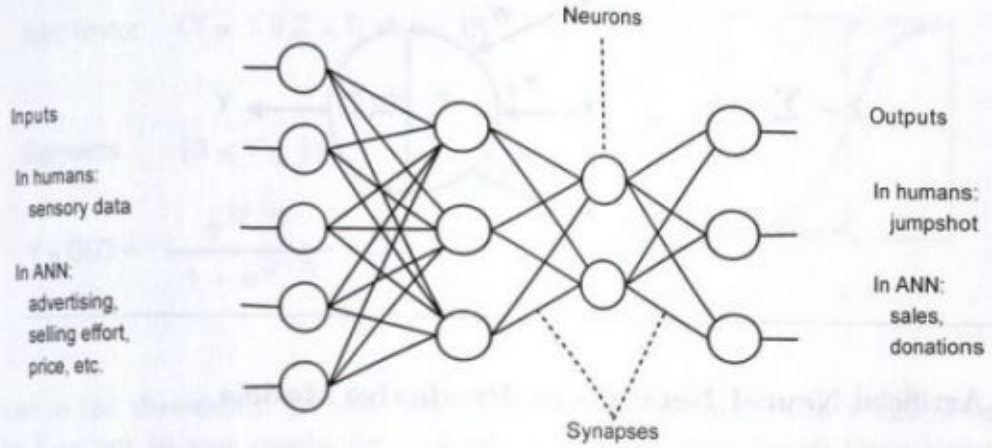
In summary:

1. A neuron
   (a) adds up its inputs,
   (b) tests the total input strength
   (c) creates an output or not depending on the input strength
   (d) adjusts the strengths of the connections
2. A neural network is a goup of connected neurons
   (a) with waves of signals moving through them
   (b) that constantly readjusts its connections,
   (c) thus changing the pattern of signals
   (d) thus allowing our basketball player to learn to score.

This mechanism provided the basis for designing artificial neural networks, and through the 1970s, it was one of the avenues that seemed to hold much promise for artificial intelligence. Unfortunately, that promise was not realized, and by the end of the 1980s research funding for AI began to dry up. Researchers involved started to focus on other applications for artificial neural networks, one of which was predictive modeling.

By treating known historical predictor variables as inputs and known historical target variables as outputs, an artificial neural network implemented as a computer algorithm could learn to provide a prediction for a target by adjusting the strength of the connections between the artificial neurons (Figure 8.4).

Figure 8.4: Comparing Actual and Artificial Neural Networks

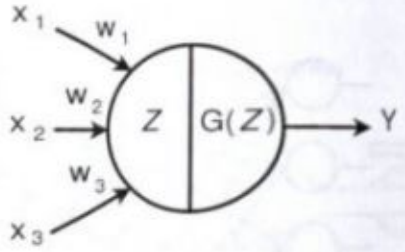


In the ANN, "nodes" take the place of biological neurons. Of course, there is quite a bit of detail that goes into adjusting the strength of the connections between the nodes during training.

In the artificial neural network, there is one *input* node for each predictor variable. This layer of nodes, on the left side of Figure 8.4, is *passive*, meaning that the nodes do nothing except pass the variables to the next layer of nodes. On the right-hand side of Figure 8.4, the *output* layer of nodes gives the predicted value of the target variables. In our examples, we only have a single target variable, but neural networks generally can have more than one output. Between the input and output nodes are *hidden* layers. The figure shows two hidden layers (the first with three nodes, the second with two nodes). In practice, the analyst selects the number hidden layers and the number of nodes in each hidden layer. For most marketing applications, a single hidden layer is adequate, and we will confine ourselves to a single hidden layer architecture in our models. The hidden and output layers contain *active* nodes, which means that this is where the calibration work is done, as described below. In a fully interconnected neural network, the type that we will use, the input nodes pass each variable to all of the hidden layer nodes.

In summary, our neural network models will be fully interconnected, single hidden layer, single output node models. We will have control over the number of nodes in the hidden layer, and of course over the number of input nodes, through the number of predictor variables used.

Figure 8.5: The Algebra of an Active Node in an Artificial Neural Network



8.2 Artificial Neural Networks as Predictive Models

As with all other types of predictive models, there are many variations on neural network models, and they have been developed and applied in many fields. Exactly how neural networks behave depend on the details of their implementation, which can vary in a number of ways:

1. **Network properties** (the number of nodes, the number of layers in the network, and the order of connections between nodes)
2. **Node properties** (threshold, activation range, transfer function)
3. **System dynamics** (initial weights, learning rule, and weight decay)

These properties are all implemented as mathematical relations between inputs and outputs, and we present a brief outline of one approach.

Each individual node is a model that relates a number of inputs to an output. Figure 8.5 shows the structure of a single node as a combination of a linear model followed by a transformation, G, of the output of the linear model. The linear model is completely analogous to the linear model in regression, with the coefficients called weights and represented by w_i (w_1, w_2, and w_3 in our example). The inputs (predictor or independent variables) are the x_i, and the output (dependent) variable is Z.

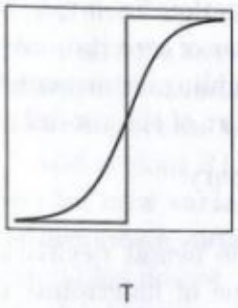
G(Z) simulates the threshold behavior of the neuron, and passes the signal along as "1" if Z exceeds a threshold level T, or blocks it by setting the output to zero if Z is less than the threshold T. As shown in Figure 8.6, the transfer function G can be a "hard" transformation, with the output Y only taking the values zero or one. It can also be a soft transformation, with intermediate values allowed via a sigmoidal, or S-shaped, function (like our old friend the logit model). This allows for some "leakage" activation if the value of Z is

Figure 8.6: Hard and Soft Transfer Functions

Hard limiter (Y = 1 if Z > T; else = 0)

Sigmoidal $(0 \leq Y \leq 1)$

$$Y = G(Z) = \frac{e^{(Z-T)}}{1 + e^{(Z-T)}}$$



close to the threshold. In this case, the relation between the x inputs and the Y output in our model for a *single node* looks very much like a logistic regression model.

The ANN connects many of these nodes together. One could take any specific architecture, and write out the full mathematical model by substituting the outputs of one set of nodes for the inputs of the following nodes. With the sigmoidal logit transfer function for all nodes, it would look like a complex linked set of logistic regression functions. You can see how this high degree of complexity allows for a lot of variety in the shape of the relationships between the original inputs and the final output of the network. You can also see that the weights, once the model is calibrated, are similar to the coefficients in the logistic regression model. Finally, it should be apparent that interpreting these weights, unlike in the comparatively simple logistic regression model, is essentially impossible. The neural network model sacrifices parsimony for greater completeness.

The calibration process has the same objective as our earlier models, which is to minimize the difference between the predicted target of the model and the observed target in the historical estimation data. The algorithm starts with trial values for the weights, and calculates a predicted output using those trial weights and the input predictor variables. It then compares that predicted output value with the observed actual target value, and calculates the magnitude and direction of the difference, or "error" in the calculation. From this, the algorithm then uses optimization techniques to adjust the weights so as to reduce this error. The process repeats, and continues reducing the error until a stopping rule is activated.

As always, overfitting needs to be controlled for, which is especially important for neural networks given their flexibility. The analyst chooses the number of nodes to put in the hidden layer. The more nodes that are used, the greater the flexibility of the neural network, and the greater the likelihood of overfitting.

Overfitting also tends to occur when some of the weights are very large. To prevent this, a weight decay factor can be used that places a penalty on the error function for large weights, thereby discouraging them. A weight decay parameter of zero indicates no penalty, with larger values of the decay parameter providing an increasing penalty for large weights. Ultimately, we will use a lift chart of the model applied to validation data to select the best model.

In summary:

1. The neural network can "learn" to represent any data set—with any type of functional relationships among the variables—to any required degree of accuracy with a sufficient number of nodes and hidden layers.
2. This allows us to capture underlying relationships without knowing the functional form (structure) of the relationship.
3. And our basketball player, or any other human, doesn't need to know Physics 101 to be able to perform exceedingly difficult tasks. He just has to train a lot.

8.3 Neural Network Models Tutorial

The major advantages of neural networks are that they are extremely flexible and can capture many sorts of relationships. Like decision trees, they easily handle nonlinearities and interactions. Unlike decision trees, they are also very sensitive to weak effects. The major disadvantage of neural network models is that the calibrated output is essentially impossible to interpret. If all we care about is prediction, this is not a problem. If we want to know which predictor variables are related to the target variable, we have to use other predictive models.

The instructions given in this tutorial assume you are starting with an essentially new workspace, requiring you to load the CCS data from the BCA data library. If you are working with the same R workspace you saved after completing previous tutorials, you will need to alter some steps in what follows, and completely skip others.

1.

Read in the CCS data set from the BCA package, and set the Estimation and Validation Samples to 50/50, as before. We will take

advantage of our previous explorations of this data set, and, for this tutorial, work with variables which we have found to be predictors of MonthGive. To compare models, we will also regenerate our preferred final logistic regression (MixedCCS2) and tree model (Full011). First, re-create the two-level New.Region variable. Use the menu option **Data → Manipulate variables → Relabel factor levels**... to **create the variable `New.Region`** in which regions R2 and R3 are assigned to the level "VanFraser," and regions R1, R4, R5, and R6 are assigned to the level "Other." Then **create the new variables `Log.LastDonAmt` and `Log.AveDonAmt` using Data → Manipulate variables → Compute new variable....** Recall that these two variables do not have zero values and hence do not require the "+1" term in the log expression. Check what you have done using the **View data set** button.

2.

Select **Model → Statistical models → Generalized Linear Model**... and in the dialog box **select MonthGive as the target, and `AveIncEA`, `DonPerYear`, `Log.AveDonAmt`, `Log.LastDonAmt`, and `New.Region` as predictor variables. Set the sample to `Sample=="Estimation"`and label this model `MixedCCS2`. Click OK.**

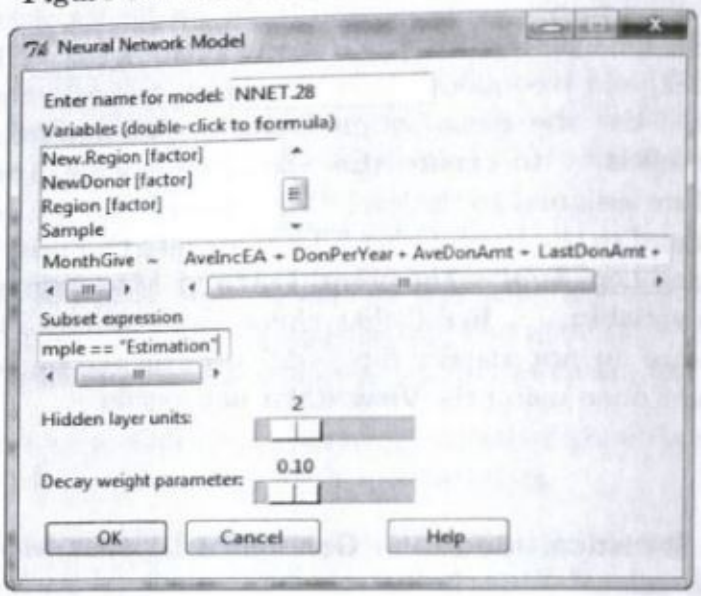
Then select **Model → Machine learning models → Tree Model → Train rpart tree...** and **select MonthGive as the target, and all of the remaining variables except the three newly created variables and `Sample` as predictors** (recall that pruning using the cp parameter will automatically eliminate many variables from the final model). **Set `Sample=="Estimation"` and `cp = 0.011`. Name the model `Full011` and click OK to run the model. Select Model → Machine learning models → Tree Model → Plot rpart tree...** and click OK.

3.

Select **Model → Machine learning models → Neural network model...** to bring up the dialog box (shown in Figure 8.7). **Set `MonthGive` as target.** The first exercise will be to observe the effect of increasing the numbers of nodes (called "units" in R) and therefore the flexibility of the model to capture detail in the data. **Select `AveIncEA`, `DonPerYear`, `AveDonAmt`, `LastDonAmt`, and `Region` as predictor variables. Set the sample to `"Estimation,"` and the decay parameter slider to `0.10`.** This level works well for our purposes and we will always use it. We will explore three different models, with 2, 3, and 4 hidden layer units. For this first model, **set the hidden layer units slider to 2, and name the model `NN.HL2`. Click on OK.**

We indicated earlier that the number of hidden layers was one of the elements

Figure 8.7: The Neural Net Model Dialog Box



an analyst can alter in a neural network's structure. However, only a single hidden layer is allowed in the underlying R function we are using (which is called `nnet`). At first this may seem like a serious limitation, but, based on our expreience, in marketing applications simpler, single hidden layer neural networks almost always predict new data better than multiple hidden layer networks. While only a single hidden layer is allowed, the number of nodes in that layer can be selected (the number corresponds to the selection of "hidden layer units"). In addition, the analyst can set the decay weight parameter, which we will set to 0.01.

The output window for the neural network model **`NN.HL2`** is shown in Figure 8.8, and it shows the calibration coefficients (or, as the jargon goes in neural networks, the "training weights") estimated between the five input nodes, the two hidden layer nodes, and the final output or target node. These weights, while inspired by synapse strength, are similar to estimated coefficients in calibrated regression and logistic regression models. The big difference is that it is much more difficult to interpret these weights in terms of a predictor variable's impact on the target variable.

4.

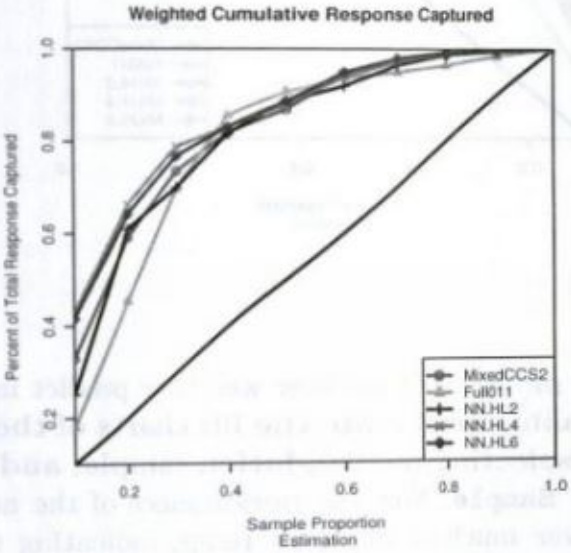
Run two more neural network models with all parameters the same, **except use four hidden layer units in one model, and six hidden layer units in the next model.** The dialog box will return with the variables previously used. Remember to set the decay parameter to 0.10. **Name these**

Figure 8.8: Neural Network Model Results

```
> summary(NNET.28)
a 9-2-1 network with 23 weights
options were - entropy fitting  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1 i9->h1
  0.22  0.00   1.23  -0.03   0.06   2.62   0.97  -0.27  -0.65  -0.58
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
  3.19  0.00  -0.96  -0.31  -0.01   3.20  -0.29   0.19  -0.85   0.00
 b->o h1->o h2->o
-1.42  2.89 -2.69
```

two models **NN.HL4 and NN.HL6.** You will likely notice that with more nodes, the model takes a bit longer to run.

Figure 8.9: Estimation Sample Cumulative Captured Response

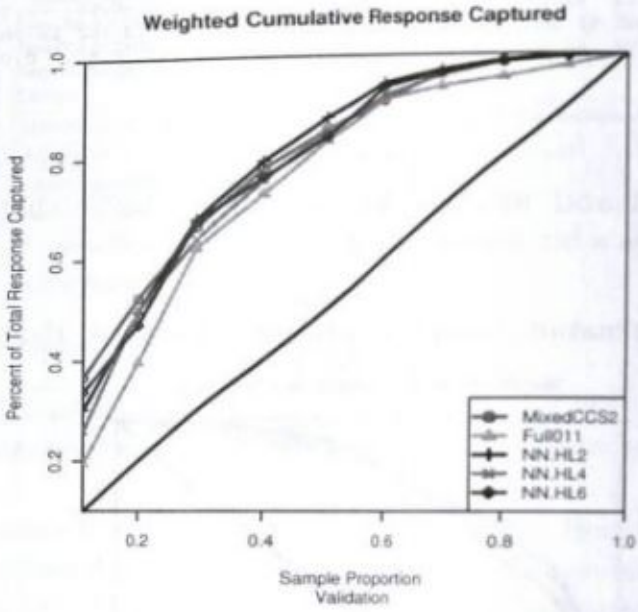


5.

Construct cumulative response lift charts of the five models using the estimation sample (**Assess** → **Graphs** → **Lift Charts**) and select the five models. Recall that the true response rate is **0.01**, the target variable level is **Yes**, and name it `CCS Estimation Sample` to identify the sample used. The result should appear as in Figure 8.9. A model with two nodes gives similar lift to the logistic model, which we put quite a bit of work into to get a good model. At different sample percentages, different models perform slightly differently, but note that increasing the number of nodes generally increases the ability of the model to follow variations in the data, especially over the first few deciles. But wait! Are these "better" models following changes that

are repeatable in different data sets, or are they just getting better at tracking unrepeatable noise?

Figure 8.10: Validation Sample Cumulative Captured Response



6.

The critical test of the models is to see how well they predict using new data that were not used in calibration. **Create the lift charts of the five models as before, this time selecting the validation sample, and naming the plot `CCS Validation Sample`.** Now the performance of the neural network models are reversed over much of the data range, indicating that four and six nodes allow too much flexibility, and overfits the estimation data (Figure 8.10). Our best logistic regression model, and the two-node neural network perform the best, especially with the best (first and second decile) prospects.

7.

The neural network may not be able to take full advantage of its flexibility here because we have restricted it to use only the variables that the previous models found to be related to the target. That is, we may be giving our best logistic regression model an unfair advantage. To explore this, **run neural network models with all of the remaining variables used as predictors**. Use the three newly computed variables (`Log.LastDonAmt`, `Log.AveDonAmt`, and `New.Region`) rather than the original variables. Make sure you don't enter the

target, MonthGive, or Sample into the predictor list. Try different numbers of nodes. Compare your new models with the `MixedCCS2` model by **generating lift charts on both the estimation and validation data sets. Briefly describe your results.**

Chapter 9

Putting It All Together

At this point you have had an opportunity to gain some experience using the data mining tools contained in R and R Commander. Given this basic background, the primary goal of this chapter is to provide you with a framework that will allow you to rapidly develop a reasonably good model. A model can usually be improved by devoting more effort to it, but there are diminishing marginal returns to this effort. The popular 80/20 rule applies, and the "reasonably good" model is sometimes referred to as the "80% model," implying that with 20% of your modeling time and effort you will get the first 80% of the return, and it will take 80% of your time to get the remaining 20% return (that results in the best possible model).

Before presenting our rapid model development framework, we first present an additional tool, stepwise variable selection (Venables and Ripley, 2002). Stepwise variable selection is an automated way of finding the set of predictor variables in a regression model (either linear or logistic) that is the best based on some criterion (often AIC).

A tutorial showing you how we applied the framework to a new database follows the presentation of the rapid development framework. At the end of the tutorial we will show how to use R Commander to "score" a database. Scoring a database amounts to creating a new variable in a database that allows the user to rank each customer in the database in terms of their attractiveness as a campaign target based on the fitted values of a particular predictive model. We are already familiar with this idea: scoring is exactly what is required to create a lift chart, but the score is not added to the database.

9.1 Stepwise Variable Selection

To illustrate what stepwise variable selection involves, we begin with a simple example. Consider the following simple three-variable linear predictor for either a linear or logistic regression model:

$$a + b_1 Var1 + b_2 Var2 + b_3 Var3,$$

where a, b_1, b_2, and b_3 are coefficients to be estimated, and Var1, Var2, Var3 are predictor variables that can be continuous numeric variables, categorical factors, or a mix of the two. In what follows, this is the linear predictor for the "maximal" model to be considered (which typically does not correspond to a model containing all the variables in a database, but does contain the set of available variables that the analyst thinks might be useful predictors).

The goal of stepwise variable selection is to determine the set of variables out of Var1, Var2, and Var3 that results in the model with the minimum AIC value.[1] There are three possible ways of doing this. The first (known as backward stepwise variable selection) begins by fitting the maximal model to determine its AIC, and then estimate the models that contain the variable sets {Var1, Var2}, {Var1, Var3}, and {Var2, Var3} to determine the AIC values for these three two-variable models. If the AIC is lowest for the maximal model, then the stepwise variable selection process terminates, returning the maximal model. On the other hand, if one or more of the other models (each containing one less variable than the maximal model) has a lower AIC value, the model containing the set of variables with the minimum AIC will be taken to the next step, thereby eliminating one variable from the maximal model. For this illustration, assume that the model with the minimum AIC value includes the variables Var1 and Var3 (dropping Var2). In the second step the AIC for this model will be compared to the two models that contain only a single variable (a model that contains only Var1 and a model that contains only Var3). If the model containing both variables has the minimum AIC, the process terminates returning this variable. However, if one or both of the single variable models has the minimum AIC, the single variable model with the minimum AIC is returned, and the process stops.[2]

The second approach (known as forward stepwise selection) is to compare all possible single variable models, and select the single variable model with the minimum AIC in the first step. For our illustration, assume that the best single variable model contains only variable Var3. In the second step the single variable model that contains Var3 will be compared to two-variable models that contain the set of variables {Var1, Var3} and {Var2, Var3}. Both alternative models contain Var3, so what the algorithm is really doing is determining the next best variable to add to the model. If the single variable model containing Var3 has the minimum AIC, the process terminates, and this model is returned. If one or both of the two-variable models has a lower AIC than the single variable model, then the two-variable model with the smallest AIC

[1] In this discussion we focus on the AIC, but criteria other than the AIC can be substituted.

[2] The process could continue comparing the best single variable to a constant-only model, but most stepwise variable selection procedures take as the "minimal" model one that contains a single variable.

is moved to the next step; assume that this is the model containing Var1 and Var3. In the next step the AIC of a model containing Var1 and Var3 is compared to the AIC for the model containing all three variables (the maximal model). If the maximal model has the smallest AIC it is returned, otherwise, the two-variable model containing Var1 and Var3 is returned.

The final approach combines both the backward and forward stepwise approaches. Specifically, this approach (like backward stepwise selection) begins with the maximal model and determines which (if any) variable should be removed to obtain the lowest AIC possible. In our example this would involve removing Var2 from the maximal model. In the second step, the model containing Var1 and Var3 is compared with models containing each of these variables by themselves (alternatively dropping Var1 and Var3) and with the model containing all three variables (adding Var2). Assume that the model with the lowest AIC contains only Var3. In the final step, this model is compared with the two models containing the set of variables {Var1, Var3} and {Var2, Var3}, and the model with the lowest AIC is returned.

In our three-variable maximal model example, the final approach results in a number of redundant tests, and would provide the same final set of variables as either the forward or backward stepwise selection approaches. However, as the number of variables increases, the final variables selected could differ between stepwise regression models. In backward variable selection, once a variable is removed, it will never be included in the set of variables in a subsequent step, while in forward variable selection, once a variable is added, it will never be removed in a subsequent step. In contrast, the combined approach allows a variable that has been added in one step to be removed in a subsequent step and vice versa. As a result, it selects the set of variables from the maximal model that has the smallest AIC possible, but at the cost of doing redundant comparisons. While this is inefficient, with modern computing power it can typically be done in a reasonable amount of time.

While stepwise regression can find the subset of variables contained in the maximal model that has the lowest AIC, it does not transform the variables in the maximal model. As a result, a variable may be rejected by stepwise variable selection methods, but the natural logarithm of that variable could be included in the final set of variables selected by these same methods by making appropriate changes to the maximal model. In a similar way, a categorical factor may be removed, while a version of this variable with a reduced number of levels may be accepted.

9.2 The Rapid Model Development Framework

The basic steps in the framework are:

1. Think about the behavior that you are trying to predict.
2. Carefully examine the variables contained in the data set.
3. Use tree and regression models to find the important predictor variables.
4. Use a neural network to examine whether nonlinear relationships are present.
5. If the neural network gives better lift (particularly for the validation sample), use visualization tools (those shown in the tutorials to Chapters 4 and 5) to look for and better understand nonlinearities. If not, stop.
6. Try to improve the lift from the regression model by transforming variables and including them as inputs (for example, create a new variable that is the square or log of a variable that appears to have a nonlinear relation with the target variable).

Having given you the steps, we next introduce the database that is the subject of the tutorial to provide a concrete example of an application, and to explore the framework in more detail before applying it.

9.2.1 Up-Selling Using the Wesbrook Database

The database that you are about to apply the rapid development framework to contains a list of donors provided by the UBC Development Office. The UBC Development Office is responsible for all fund-raising activities at the University of British Columbia. These data come from a data mining project that was conducted in an effort to find ways to increase the efficiency of UBC's fund-raising activities. With this data set, the goal is to identify current UBC donors who have a profile that is similar to a "Wesbrook" donor's profile. A donor is assigned to the Wesbrook class if that individual has a donation that exceeded $1000 in a previous year. The Development Office actively pursues donors that it identifies as potential Wesbrook-level donors in an effort to demonstrate to them the benefits of donating at the Wesbrook level. This is a classic "up-selling" exercise, and one of the most common applications of data mining techniques. The basic premise is that you identify those people

in your database who have not yet donated at the Wesbrook level, but have a profile similar to those that have donated at the Wesbrook level. Targeting those with higher likelihoods of being larger donors should result in a better response rate for UBC's marketing activities.

9.2.2 Think about the Behavior That You Are Trying to Predict

We know that the best Decision Support models are based on good theoretical models. While it is unlikely that you will want to develop watertight theory, developing a mental model of the problem will help frame your subsequent model-building efforts. In the case of a Wesbrook donor, what factors do you (based on your own beliefs) feel are likely to influence whether someone will give $1000 or more to UBC in a single year? For instance, you may believe that (1) an individual's ability to make a fairly large donation in a single year and (2) the individual's personal feelings toward UBC are the most critical factors in determining whether someone would donate at the Wesbrook level. Ultimately, you want to develop an initial mental model of how the underlying process works so you will be in a better position to critically evaluate the results provided by the data mining tools.

At some point you may find yourself dealing with a project in a domain with which you are unfamiliar, but you have a client that is very knowledgeable. In this instance, it is worthwhile spending time with that client determining what that client's mental model is in order to determine which possible predictor variables are likely to matter most, and what is the expected nature of the effect of a variable (i.e., what does the client think will happen if one variable increases in level with respect to the probability that a customer takes the desired action). Knowing this in advance can point out potential problems (likely with the data) that can undermine your relationship with the client.

9.2.3 Carefully Examine the Variables Contained in the Data Set

There are four reasons to carefully examine the variables in your data set. The first is to determine what measures are available in the data set that are related to the factors you (or your client) believe should be important in predicting the behavior of interest. Some of the measures will be readily available in the data set, while you may be able to construct other measures out of the available data.

The second reason to closely examine the data is to determine which variables are likely to exhibit missing value problems. The more variables with missing values that you use, the more data you are going to have to reject (unless each variable has its missing data for the same donors, as is the case for many

of the variables associated with donors without UBC degrees). If 10% of the values of a variable consist of missing values, you will have problems. Anything more than 3% should at least be thought about. Both of these numbers are somewhat arbitrary, and are only intended as a rough guide. If you do have variables with missing value problems, you will need to address them prior to conducting regression or neural network modeling. How you deal with missing values is context specific, and this issue will be highlighted in the application of the framework in this chapter's tutorial. However, early on you need to determine whether a missing value for a variable is due to factors related to the construction of the database itself or if it is a result of non-reporting issues (e.g., an individual simply did not provide some information). Knowing the "cause" of a missing value can help in deciding what to do about missing instances of that variable.

Similar to the last reason, the third reason to closely inspect the data is to find categorical factor variables that have levels that contain a small number of records. A number of problems can arise from this issue. Specifically, the statistical reliability for the indicator variables generated for these factor levels is very low. This situation can also cause problems with certain tools such as the plot of means visualization method. Finally, if multiple samples are created within the database (i.e., estimation, validation, and holdout samples), certain samples may contain no records that possess a particular level of a factor, which creates problems in predicting the target variable for samples other than the estimation sample, causing the lift chart tools to fail. The way to deal with these problems is to create a new factor with a smaller number of levels (and with more records in "thin" categories) from the original factor.

The fourth thing you need to think about is whether there are variables in the database that are either used to construct the target variable of interest or are "trivially related" to the target variable. As an example of a variable that is "trivially related" to the target variable, consider a variable that indicates whether an individual has purchased an extended service plan in a model that predicts whether an individual will purchase a major appliance, using department store transaction data. The variable is likely to be strongly related to appliance purchase, and hence appear as a very good "predictor" of whether someone buys an appliance. However, if we consider the timing of the two variables, it is easy to see that this would be due to a reverse causality. Specifically, purchasing an extended service plan occurs *after* purchasing a major appliance, so it cannot cause the appliance purchase — the appliance purchase causes the service plan purchase. Including such predictors in the model will often give very good fit and lift but will be of little value when it comes to identifying new customers' purchase likelihood. Therfore, variables like this should never be used as predictors, once again highlighting the critical

importance of understanding exactly what all of your variables represent in the real world.

Finally, you need to set the role of database housekeeping variables to either a case name identifier or remove them. Housekeeping variables are used to sort and merge records, but are not relevant in explaining underlying behavior. Sometimes an id number will come up among the set of predictors in a stepwise regression since the lower the id number, the longer the individual has been a part of the database, and thus the older they are likely to be. In this instance the id variable is acting as what is known as a "proxy variable" for age, but it makes more sense (and is much more interpretable) to directly include the age variable instead. At other times, the id variable is capturing random patterns in the data that will simply not be present in new data. This can be handled by methods to control overfitting. Postal and ZIP codes can also be a problem. If they are treated as categorical, and there are hundreds, the regression model will add hundreds of indicator variables, and many may not be estimable. If they are treated as interval (as U.S. five-digit ZIP codes might be) they might capture some difficult to interpret locational factors (e.g., U.S. ZIP codes increase in numeric size, say from 01111 to 99999, from east to west across the country). Appropriate geographic and geo-demographic variables should be included rather than the postal code itself.

9.2.4 Use Decision Trees and Regression to Find the Important Predictor Variables

Finding the set of important predictor variables is about 90% of the battle in developing a good predictive model. Using decision trees and regression can often help you quickly find the most promising set of predictor variables. Each of these methods has its relative strengths and weaknesses in finding the set of important predictor variables, but when both are used, you can take advantage of the strengths that each offers. Specifically, decision trees are often good at finding the most important predictor variables, and they seem to be less likely to find variables that are simply correlated with random patterns in the estimation sample. In addition, tree models are capable of detecting predictor variables that have non-linear effects on the target. As an example, consider the influence of a person's age in a model that is attempting to predict how much an individual will invest in mutual funds in a particular year. It is likely to be the case that mutual fund investments will start at a low level for the typical individual when she is young, will increase each year until this individual is in her late 50s or early 60s, and then will begin to decline (perhaps rapidly) as she enters her senior years. In this instance, two two-way splits for age would indicate relatively low levels of new mutual fund

investments for the youngest and oldest groups, and a higher level for a middle group.

The strength of regression is its ability to find those weaker, but still important, predictor variables. However, finding these weaker variables comes at the cost of finding more variables that are included only to capture random patterns in the data. Another relative weakness of regression is that it can "miss" variables that have a non-linear relationship to the target if potential nonlinearities are not explicitly taken into account. In our mutual fund example, the age variable is unlikely to be found by regression. The reason is that the relationship between age and mutual fund investing looks like an inverted U. As a result, throughout the entire range of age, the best-fitting line would have a slope of zero. Although, if you broke the age range into two variables (one that gives the age of someone aged 59 and below, and the other giving the age of someone aged 60 and above) you would likely find a strong positive relationship between age and mutual fund investing for the first variable, and a strong negative relationship for the second. Unfortunately, you would need to know that you should split up the variable this way prior to running the regression models.

Having said all this about the different tools, what we are really getting at in this step is the interplay between your initial mental model of customer behavior and the data. Specifically, your mental model will likely be incomplete, and the two modeling tools will point you to other variables that you will feel comfortable incorporating into your mental model. However, the tools may also point to other variables that simply do ***not*** make sense given your mental model. If this is the case, there is a very good chance that these variables actually do not explain the behavior, and were included as a result of a particular model's attempt to account for random patterns in the data. If you estimate a model that includes variables you do not think make sense, remove these variables and re-run the model.

9.2.5 Use a Neural Network to Examine Whether Nonlinear Relationships Are Present

Neural networks are very good at finding and mimicking nonlinear patterns in the data. Because an estimated neural network model is nearly impossible to interpret, they are not good at finding the set of predictor variables to include in the model. However, in the prior step of the framework you have hopefully uncovered the important set of predictor variables. Moreover, through the use of a decision tree, you have had a preliminary opportunity to see whether any important variables appear to have a nonlinear relationship with the target. The use of a neural network will allow you to assess the extent to which

nonlinear relationships in the data matter for purposes of prediction. The way that you accomplish this is by first estimating a fairly flexible neural network (one with three hidden nodes in the hidden layer). After you have estimated the network, compare it to your final stepwise regression model through the use of lift charts that examine the models performance on the estimation and validation sample. If your final stepwise regression model performs about as well as or better than the neural network across the samples, then you should view your final regression model as the "good enough" (or "80%") model. If the neural network outperforms the regression model in the validation sample, *and* it is important to know the details of the relation between predictors and target, then it is time to try and build a better regression model that captures the nonlinear patterns in the data.

9.2.6 If There Are Nonlinear Relationships, Use Visualization to Find and Understand Them

Investigating nonlinear relationships is only necessary for continuous *predictor* variables. If the *target* variable is continuous, then looking for nonlinear relationships can typically be accomplished by looking at a scatterplot and/or line plot of the predictor and the target variables. If the target variable is binary, and the predictor variable under investigation has an ordinal categorical scale, then a plot of means of the target (converted to 0–1 numeric) that uses the predictor variable as the grouping variable in the plot works well. The most difficult relationship to examine graphically is one that involves a binary target variable and a continuous predictor variable. The best way to look for any potential patterns is to use the methods (e.g., binning the predictor) we present in the logistic regression tutorial of Chapter 5. In general, if you cannot visualize an obvious relationship between a predictor and the target, the nonlinearities are not likely to be strong enough to matter much in the model. If you see a decreasing returns concave type of relation, a logarithmic transform of the predictor should help. Replace the predictor with its logarithm in the regression model and check the lift on the validation data. A U-shape relation can be captured by a squared predictor (remember to include both the original variable *and* the square of that variable). How far you wish to go in this search for improvement depends on how much time you have, keeping in mind that the incremental improvements you get will be smaller and smaller as you proceed.

9.3 Applying the Rapid Development Framework Tutorial

A special note for what follows. The main steps in the tutorial are in a bold typeface, with explanations given in a regular typeface.

This is a good place for a reminder that it is absolutely essential for the analyst to understand the meaning and source of all variables, whether they come from surveys, transactional data, census data, or anything else.

Database construction will typically include oversampling of the target variable, and then separating the sample into an estimation (or "calibration"), validation (or "holdout"), and (possibly) holdout samples. For this exercise, the database was oversampled so as to have a 50/50 split of Wesbrook/non-Wesbrook donors. The database was then exported to a dBase format file for easy importation into R.[3] The name of the dBase format file is Wesbrook.dbf, and it contains 2770 records. You can download a zip archive containing the data file using the link http://www.customeranalyticsbook.com/Wesbrook.zip.

Import the Wesbrook dBase file into R, (Data → Get from → dBase (dbf) data set...), naming the dataset `Wesbrook` and UNchecking the "Do not convert character variables to factors" checkbox, since we do want the character variables to be factors (categorical). When done, the bottom of the R Commander window should indicate that you have imported 2770 rows and 31 columns.

Given the objectives of this tutorial, we will only create estimation and validation samples, and no holdout sample. Moreover, given the fairly small size of this data set, we will place 70% of the records into the estimation sample, and 30% into the validation sample. To do this, **use the pull-down menu option Data → Organize → Create samples in active data set..., and allocate 70% of the observations into the estimation sample and 30% into the validation sample. Check your results with View data set.**

[3] This format belongs to a database management product called dBase that was the dominant PC database management product during the early days of personal computing (it pre-dates the introduction of MS-DOS, let alone Windows). While dBase is no longer dominant, a number of popular products use the file format. The reasons for this is that the files are fairly small (though not as small as a tab-delimited or comma-separated value text file), compress well, developers of other software packages find them easy to work with (which is not true of Excel files), and they contain enough information about the variables within the file that the variable types are correctly imported (which can be a real issue with either tab-delimited or comma-separated value text files, and even directly imported Excel files). It is the presence of the variable descriptors that make dBase files (which have the suffix of *.dbf) slightly larger than tab-delimited or comma-separated value text files.

Step 1: Mental Model

As is hinted at above, our mental model is based on the notion that there are two important determinants of whether an individual will donate $1000 or more to UBC in a single year, specifically, how favorably someone views UBC based on her/his personal experience with the university, and on her/his ability to afford a fairly large donation in a single year. We know that measures of her/his ability to afford a fairly large donation will be easier to obtain than measures of her/his personal experiences with UBC. However, even measures related to this second factor are likely to be available in the database.

Step 2: Examine and Clean Data

Are there measures related to our mental model? Start by looking at the available measures in the data (excluding the `Sample` variable we just created), which are shown below. Looking at the available data, several measures of the ability to afford a large contribution are readily available in the database (Table 9.1). "EA" refers to a census "Enumeration Area" and identifies neighborhood variables.

Specifically, the number of years an individual has been out of school, and the faculty from which he or she graduated, should be related to his or her ability to make a large contribution. In addition, the average income of households and the average value of dwellings (a measure of both income and wealth) for the enumeration area in which the individual resides should also provide good measures of an individual's ability to make a fairly large contribution. Again, remember that these demographic variables are actually geo-demographics, and, hence, give the characteristics of the donor's neighborhood, not the individual donor.

In terms of potential measures related to an individual's feelings toward UBC, there are a few. Specifically, the faculty and/or department an individual graduated from is likely to influence her/his feelings toward the university. Another potential measure of an individual's feelings may be related to whether other members of her or his family (i.e., parents, children, and/or spouse) also attended UBC. All of the above measures are readily available in the database.

Two additional variables that do not add information, but are easier to interpret at the end, will be created: `YRFDGR` (the years since the individual received his or her first UBC degree: 1999 – `FRSTYEAR`) and `YRLDGR` (the years since the individual received his or her most recent degree from UBC: 1999 – `GRADYR1`).[4] These variables will be easier to interpret than the year of grad-

[4]Note that the label of `GRADYR1` is misleading — it says "1st UBC graduation year." However, they are counting from the present backward to the past, so 1 is actually "latest."

Table 9.1: The Variables in the Wesbrook Database

Variable	Label
ATHLTCS	Participation in athletics
AVE_INC	Average 1995 household income in the EA of residence
BIGBLOCK	Participation in Big Block
CHILD	Child of UBC student or alumnus code
CNDN_PCT	% Canadian citizenship in the EA of residence
DEPT1	Department of 1st UBC degree code
DWEL_VAL	Average value of dwellings in the EA of residence
EA	Enumeration Area universal ID
ENG_PCT	% English as home lanugauge in the EA of residence
FACSTAFF	UBC faculty or staff member
FACULTY1	Faculty of 1st UBC degree code
FRSTYEAR	Earliest UBC graduation year
GRADYR1	1st UBC graduation year
HH_1PER	1 person households % in the EA of residence
HH_2PER	2 person households % in the EA of residence
HH_3PER	3 person households % in the EA of residence
HH_45PER	4-5 person households % in the EA of residence
ID	Viking ID code
INDUPDT	Date of last update of personal data
MAJOR1	Degree of 1st UBC degree code
MARITAL	Marital status code
MOV_DWEL	% of households in movable dwelling in the EA of residence
OTHERACT	Participation in other activities
OWN_PCT	% of households that own their dwelling in the EA of residence
PARENT	Parent of UBC student or alumnus code
PROV	Two character province/territory abbreviation
SD_INC	Standard deviation of household income in the EA of residence
SEX	Gender code
SPOUSE	Spouse of UBC student or alumnus code
TOTLGIVE	Total lifetime giving
WESBROOK	Ever given at the Wesbrook level

Figure 9.1: Computing the YRFDGR Variable

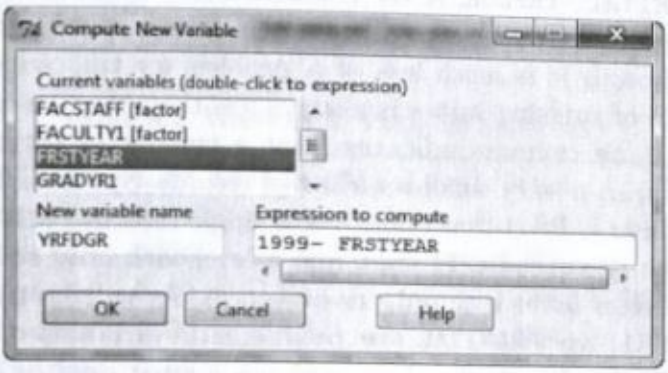
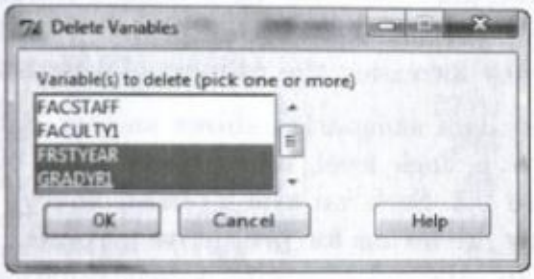


Figure 9.2: The Delete Variable Dialog



uation variables. **Select Data → Manipulate variables → Compute new variable...** to bring up the dialog box in Figure 9.1. The box is shown with the information for YRFDGR filled. **Click OK. Repeat for the new variable** `YRLDGR` **(1999 - GRADYR1).**

You can view the data set to ensure the new variables have been created.

Since we want to make sure and use the two new "years since graduation" variables instead of the original "year of graduation" variables we remove the originals from the data. **Select Data → Manage variables in active data set → Delete variables from data set...** and highlight `FRSTYEAR` and `GRADYR1` (Figure 9.2). **Click OK.**

Look for potential problems with missing values and factor levels with few records.

The next step is to check for missing values by **selecting Data → Clean → Summarize variables**. The results are shown in Table 9.3. The critical columns in the results for our purposes are `%.NA` (the percentage of missing values for each variable) and `Min.Level.Size` (the number of records in the smallest level for each factor). An examination of the `%.NA` column indicates that a number of variables have no missing values, while others have a high percentage of missing values (`MAJOR1` has nearly 71% missing values). The

particularly problematic variables in terms of missing values are FACULTY1, DEPT1, MAJOR1, MARITAL, YRFDGR, AND YRLDGR, which all have between 31% and 71% missing values. In addition, all the geo-demographic variables have missing values, although it is much less of a problem for this set of variables since the percentage of missing values is always less than 3%. An examination of the Min.Level.Size column indicates that a number of factor variables have levels that contain a very small number of records. Specifically, INDUPDT and MAJOR1 both have a level that contains a single record, PROV that has a level with only three records, DEPT1 that has a level with only seven records, and MARITAL that has a level with only two records. As will be discussed below, INDUPDT, MAJOR1, and MARITAL are problematic variables on a number of dimensions. DEPT1 is also challenging, and has a great deal of overlapping information with FACULTY1, which we also discuss below. This leaves PROV, which is the province/territory indicator of a donor's residence. The natural way to deal with this variable is to group provinces and territories into geographic groups, thereby increasing the number of records in each level.

Carefully inspecting the data summaries shows another data problem; the factor BIGBLOCK only has a single level, which you can see by examining the Levels column in Table 9.3. Such variables (called *unary*) are common in databases. However, they are useless for predictive purposes, and can foul up modeling routines if left in. Any variables with a single value should be deleted. **Select Data → Manipulate variables → Delete variables from data set...**, select BIGBLOCK, and OK.

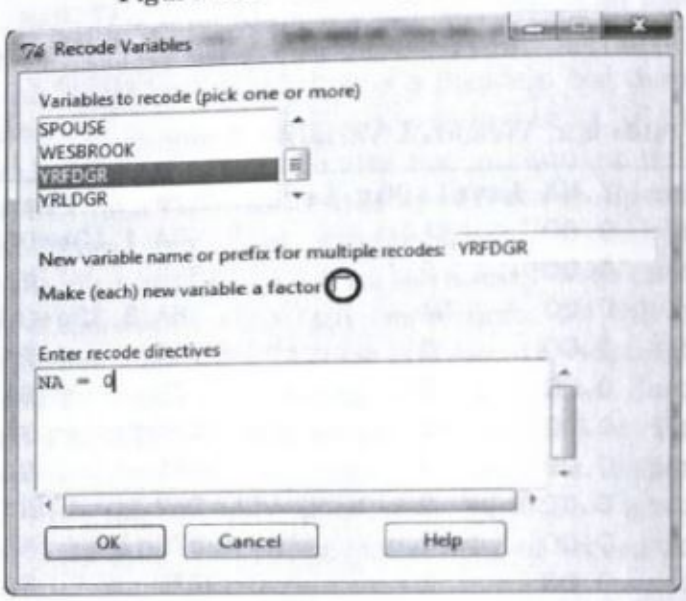
If someone did not graduate from UBC, then YRFDGR and YRLDGR will have missing values. In this instance missing values are actually due to choices made in database construction and are meaningful, in that they tell us something useful about the individual's relationship to UBC — in this case that they did not receive a degree from UBC (likely they are donors who did not even attend UBC). Similarly, FACULTY1, DEPT1, and MAJOR1, which represent the faculty, department, and major of the individual's first degree, are also meaningful indicators of no degree. However, MAJOR1 has over twice the number of missing values as the other two variables. A visual examination of these three variables (**View data set**) indicates that only certain faculties have majors. For example, it appears that Arts and Applied Sciences do have majors, while Commerce and Law do not. Sorting out how to handle MAJOR1 is likely to be a big job, so we will not use it for our good enough model. However, we should keep it in mind if it turns out we have lots of time to work on it.

Dealing with missing values for faculty and department is much less complex. Specifically, for these two variables, we can simply change the missing values to a new value that indicates that the category for that individual was

Table 9.2: Wesbrook Variable Summary

	Class	%.NA	Levels	Min.Level.Size	Mean	SD
ID	numeric	0.00	NA	NA	1.13e+05	6.71e+04
WESBROOK	factor	0.00	2	1385	NA	NA
TOTLGIVE	numeric	0.00	NA	NA	3.33e+03	7.64e+03
PARENT	factor	0.00	2	284	NA	NA
CHILD	factor	0.00	2	79	NA	NA
SPOUSE	factor	0.00	2	1037	NA	NA
SEX	factor	0.00	2	1081	NA	NA
FACSTAFF	factor	0.00	2	263	NA	NA
ATHLTCS	factor	0.00	2	179	NA	NA
BIGBLOCK	factor	0.00	1	2770	NA	NA
OTHERACT	factor	0.00	2	266	NA	NA
Sample	character	0.00	NA	NA	NA	NA
INDUPDT	factor	0.43	974	1	NA	NA
EA	numeric	0.47	NA	NA	5.66e+07	7.48e+06
PROV	factor	0.61	11	3	NA	NA
MOV_DWEL	numeric	1.01	NA	NA	3.66e-03	3.25e-02
HH_1PER	numeric	1.01	NA	NA	2.26e-01	1.67e-01
HH_2PER	numeric	1.01	NA	NA	3.24e-01	8.67e-02
HH_3PER	numeric	1.01	NA	NA	1.51e-01	5.74e-02
HH_45PER	numeric	1.01	NA	NA	2.56e-01	1.38e-01
DWEL_VAL	numeric	1.01	NA	NA	4.43e+05	2.86e+05
ENG_PCT	numeric	1.01	NA	NA	8.12e-01	1.66e-01
OWN_PCT	numeric	1.01	NA	NA	7.05e-01	2.38e-01
CNDN_PCT	numeric	1.05	NA	NA	8.82e-01	7.96e-02
AVE_INC	numeric	2.60	NA	NA	7.85e+04	3.71e+04
SD_INC	numeric	2.60	NA	NA	1.50e+05	1.48e+05
FACULTY1	factor	34.19	14	19	NA	NA
DEPT1	factor	34.19	23	7	NA	NA
YRFDGR	numeric	34.19	NA	NA	2.67e+01	1.65e+01
YRLDGR	numeric	34.19	NA	NA	2.49e+01	1.65e+01
MARITAL	factor	65.52	6	2	NA	NA
MAJOR1	factor	70.90	142	1	NA	NA

Figure 9.3: Recoding YRFDGR



"none" or "no degree."[5] The two "years since ... degree" variables are a bit more difficult to handle since it is difficult to assign a year to a degree an individual did not receive. A common way of dealing with this situation is to force the missing values of each of these two continuous variables to zero, and then add a new indicator variable that indicates that the individual did not receive a degree to adjust for the imputed zero values. In this instance, the indicator variable is likely to be redundant information since if a faculty and/or department variable enters the regression model, then the indicator variable will always equal one when the faculty/department variable equals the "no degree" category. As a result, it will serve as the indicator, and we do not need to create a separate "no degree" indicator variable for years of first and last degrees.

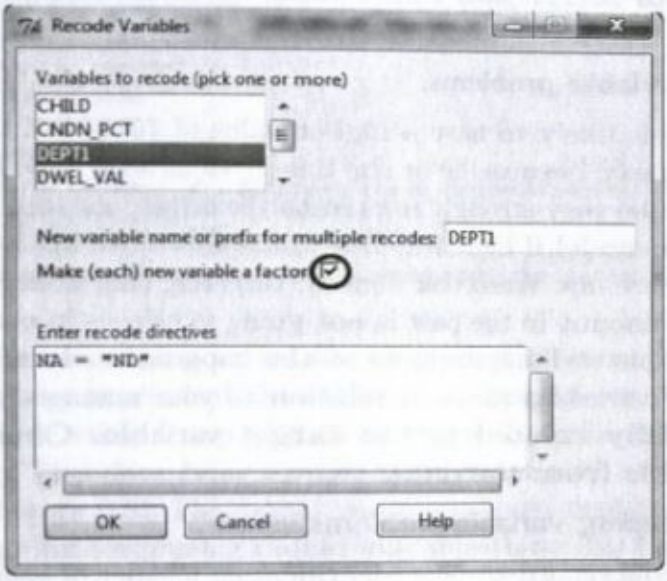
Select **Select Data → Manipulate variables → Recode variable...**. In the dialog box (Figure 9.3) **select YRFDGR, and keep the new variable name the same, YRFDGR.** As long as no mistakes are made, this will save some time as this will overwrite the old variable and we will not have to go back and delete the old variable. We want the new variable to be continuous, not categorical, so **make sure to UNCHECK THE FACTOR** box or you will be redoing the variable creation step! The recode directive is `NA=0`. Click **OK**, and agree to overwrite the variable when requested.

[5]Remember that if we leave the "NA" code in, R will read that as "missing," and remove the entire case in procedures like logistic regression.

Table 9.3: The New Year since Degree Variables

	Class	%.NA	Levels	Min.Level.Size	Mean	SD
YRFDGR	numeric	0.00	NA	NA	1.75e+01	1.84e+01
YRLDGR	numeric	0.00	NA	NA	1.64e+01	1.78e+01

Figure 9.4: Recoding DEPT1



Repeat the recoding for `YRLDGR`. View the data to make sure THE "0" replaced the "NA." Also, summarize the data again (**Data** → **Clean** → **Summarize variables**) to see if the appropriate changes were made. The result should look similar to Table 9.3 — showing mean and standard deviation rather than the levels and minimum size level of a factor, and no missing data.

For `DEPT1`, which is categorical, recode the value NA to "ND" for "No Degree." Leave the factor box checked this time. Note that when we are recoding to names like "ND" rather than numbers like "0," we put quotation marks around the recode value to indicate that these are character values. The recode dialog box is shown in Figure 9.4. Again since we kept the same variable name, it is important not to make a mistake! When it is set, click OK and agree to the overwrite query.

Repeat for `FACULTY1`, setting NA to "ND." Check the data to ensure the replacements have been made.

Missing values for `MARITAL` are due to non-reporting. They tell us nothing

about marital status, only that the individual's marital status was unknown to the UBC Development Office (the marital status of couples where both spouses went to UBC is more likely to be known by the development office). We don't have any reason to think that it will be a useful predictor, nor can we think of any compelling reason that individuals who do not report marital status would have any relation to Wesbrook-level donating behavior, so there is little to gain from treating non-reporting as a valid response. Therefore we will remove it from the data. **Select Data → Manipulate variables → Delete variables from data set... and select `MARITAL` from the variable list, Click OK.**

Trivially related variable problems.

A Wesbrook donor is likely to have a higher value of `TOTLGIVE` than a non-Wesbrook donor simply because he or she has given at least $1000 in a single year. `TOTLGIVE` will be very strongly related to the target, and will likely dominate our predictive model if included. But since we wish to apply our model to individuals who are not Wesbrook donors, targeting only donors who have given a large total amount in the past is not going to help us in our *up-selling* campaign. This is one striking example of the importance of understanding what the available variables mean in relation to your managerial problem. **`TOTLGIVE` is trivially related to the target variable. Consequently, delete this variable from the data.**

Database housekeeping variable problems.

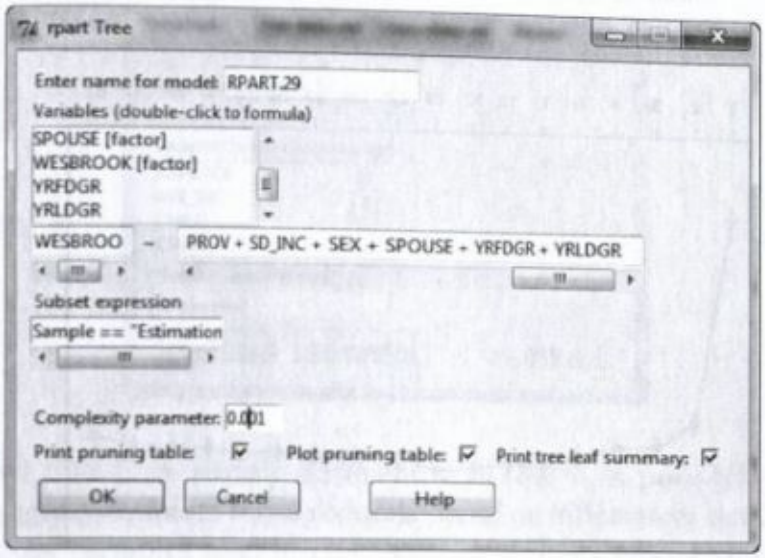
The variable ID is the record-keeping identification number of each person in the database, while EA is a code that identifies the census Enumeration Area in which an individual resides, and has the same properties as a postal code for our purposes, that is, it is not useful. The update variable INDUPDT is not only an internal variable and irrelevant to the individual donors, but also would be treated as a categorical variable with a huge number of categories, so we want to be sure it does not enter any models. **As a result, select Data → Clean → Set record names... and choose `ID` in the window. Click OK. Also, delete the variables `EA`, `INDUPDT`, and `MAJOR1` so that they are not accidentally included in a later model.**

As a final check, **View the data set** and **Variable summary for the active data set** to check that all of your changes have been made.

Step 3: Explore with Trees and Regression

We begin the third step of the framework by using a decision tree (**Model → Machine learning models→Tree model → Train rpart tree...**) to see what variables this method finds predictive of Wesbrook donors. **The target variable is `Wesbrook`.** Since we have removed most of the undesirable variables,

Figure 9.5: Estimating a Decision Tree Model



select all of the remaining variables as predictors except `Sample` in the model. The "Subset expression" should be set to **`Sample == "Estimation."`** Trying the default value for cp of 0.01 indicates that the validation error is still decreasing. Therefore, bring up the rpart tree dialog box again and **set `cp = 0.001`** (Figure 9.5). The model variables should remain the same. The cross-validation plot for pruning should look something like Figure 9.6 (it will vary because the automatic generation of holdout samples to calculate the misfit on the holdout data varies with each run). From the chart, and based on the criterion of minimum cross-validation error, we should prune the tree back to about 10 or 12 splits, corresponding to a cp value of 0.0072. **Bring up the rpart tree dialog box again,** which should still have all of the model details in it, **and change cp to 0.0072. This time, name the model `WesTree`. Click OK to Run the tree.**

Inspect the tree to see which variables were selected (**Model → Machine learning models→Tree model →Plot rpart tree...**) by the tree algorithm (Figure 9.7). These are `DEPT1`, `YRFDGR`, `DWEL_VAL`, `SD_INC`, `AVE_INC`, `YRLDGR`, and `FACSTAFF`. Happily, most variables fit our prior theory that the ability to donate and feelings toward the faculty of graduation should be relevant. The somewhat surprising variable is `SD_INC`. However, the standard deviation of income in an enumeration area increases as the average income increases, and is therefore likely a proxy for average income.

At this point we are nearly ready to run and compare logistic regression models. However, logistic regression models omit records that contain missing values for any of the variables included in the model. As a result, if two models

Figure 9.6: The Wesbrook Pruning Plot

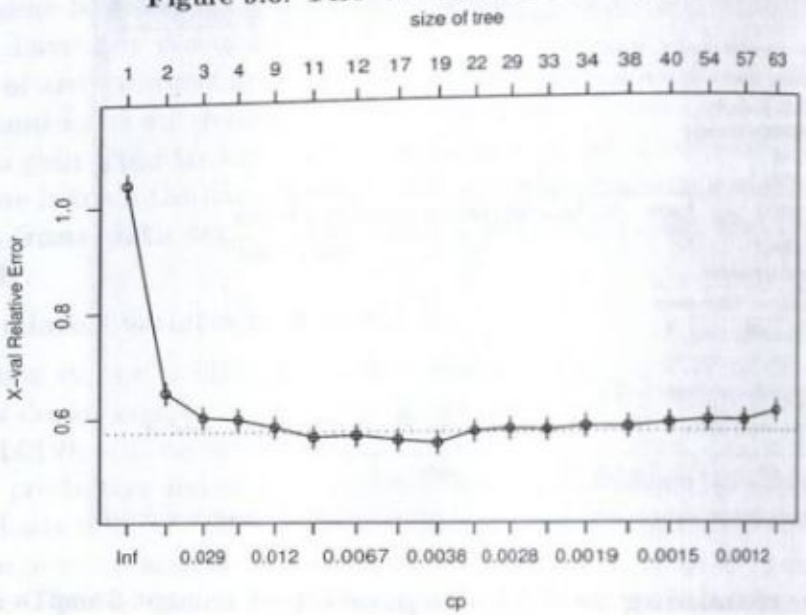


Figure 9.7: The WesTree Model Tree Diagram

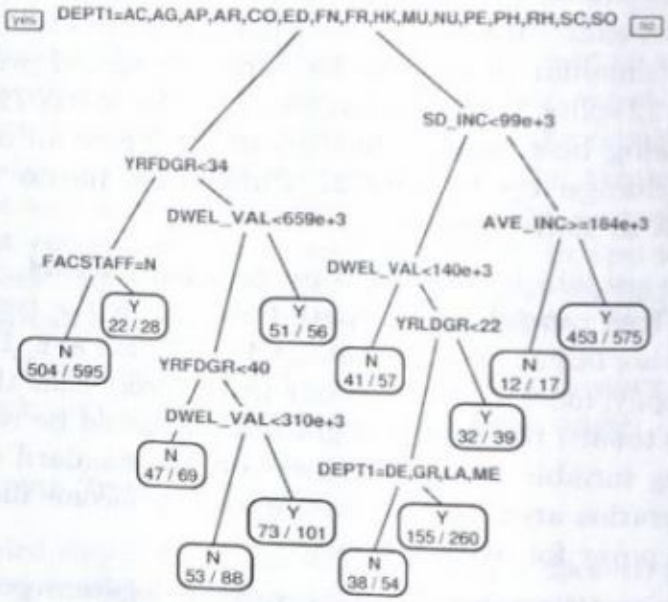
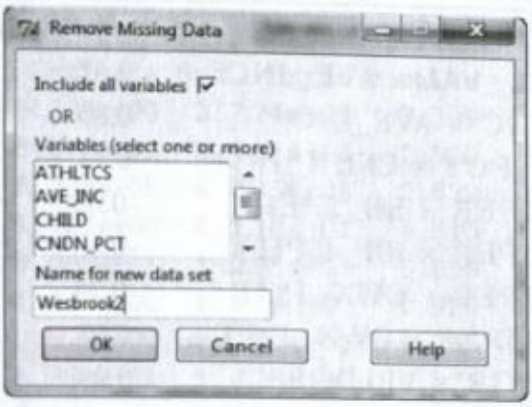


Figure 9.8: Remove Missing Data Dialog



are compared that have slightly different variables, it is possible they will be based on a slightly different set of records based on differences in missing value patterns across variables used in the two models.[6] To guarantee that we are working with the same set of records when examining different models, we want to limit the database to only those records that have no missing values for the variables we are investigating, and then run all models on the same data set. Fortunately, we have done a lot to reduce the number of missing values in the database so far. To create the new data set with no missing values, select **Data → Clean → Remove records with missing data...**, which brings up the dialog box shown in Figure 9.8. In the dialog box **check the "Include all variables" box, and make the "Name for new data set"** `Wesbrook2`. After **clicking OK**, the Wesbrook2 data set will be created and will become the active data set. By only including complete records for the variables of interest, the size of the database has only been reduced by 72 records (or a little under 2.6%). This is a reasonable reduction in the size of the database.

Now **rerun the tree model using the** `Wesbrook2` **data set, setting cp = 0.0072 as before, and name the new model** `WesTree2`. **Plot the tree and check that the small reduction in the number of records has not changed the structure** beyond some minor changes in the split points.

The last thing we want to check before beginning the logistic regression analysis is to check the correlation matrix to determine which variable pairs might suffer from collinearity issues. Recall that since correlated predictor variables can appear nonsignificant even if they are both strong predictors, we will want to check for this after we have run the model. In preparation, let's find the

[6] Another, related, reason for limiting ourselves to a data set with no missing values in the variables is that the stepwise variable selection tools complain bitterly when different regression models in a step have a differing number of observations.

Table 9.4: Highly Correlated Variables in the Wesbrook Database

Variables	Correlation
DWEL_VAL × AVE_INC	0.67
SD_INC × AVE_INC	0.80
ENG_PCT × CNDN_PCT	0.73
HH_1PER × HH_3PER	-0.69
HH_1PER × HH_45PER	-0.86
HH_1PER × OWN_PCT	-0.74
HH_45PER × OWN_PCT	0.67
YRFDGR × YRLDGR	0.98

strongly correlated variables. **Select Explore and Test → Summarize → Correlation Matrix... and in the dialog box, select all of the variables. Click OK**. We scanned through the resulting matrix for numbers around 0.65 or greater (in absolute value) and found those shown in Table 9.4.

Now we are ready to run the "maximal" logistic regression model we will use as the basis for stepwise variable selection. To do this **select Model → Statistical models → Generalized Linear Model. Select** `WESBROOK` as the **target, all of the remaining variables except** `Sample` as predictors, set `Sample == "Estimation"` as the **"Subset expression,"** and name the **model** `WesLogis`. The output of doing this is shown in Table 9.5.

Based on a McFadden R^2 of 0.31, the model appears to fit well. Significant variables are `ATHLTCS`, `AVE_INC`, `CHILD`, several of the `DEPT1` indicator variables, `FACSTAFF`, `DWEL_VAL`, `HH_1PER`, `HH_3PER`, the Quebec level of `PROV`, `SD_INC`, `SEX`, and `YRFDGR`. Given the level of correlation we saw between some of the variables previously, the statistical significance for a number of them may be greater than what their z-tests suggest. `YRFDGR` and `YRLDGR` seem particularly prone to this possible problem. The "NA" values in the `FACULTY1` variables are occurring because `DEPT1` contains identical information as `FACULTY1` for these levels, and they cannot be estimated (technically, due to "singularities").

Now that we have a maximal model (and with 23 variables, it really is a *maximal* model), we are ready to find the subset of its variables that minimize the AIC. Start the stepwise variable selection process by selecting **Assess → Stepwise model selection...**, which will bring up the dialog box shown in Figure 9.9. The only setting you want to change from the defaults is to **enter** `WesStep` **in the "Enter the name of the new model:" field**. Once you have done this, **press OK**. At this point it is time to pour yourself the hot or cold beverage of your choice, call someone on your cell for a chat, or find some

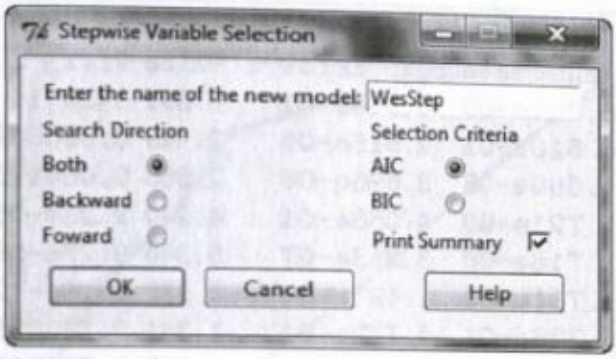
Table 9.5: Logistic Regression Results for the "Maximal" Model

```
Coefficients: (4 not defined because of singularities)
                Estimate Std. Error z value  Pr(z)
(Intercept)   -9.069e+00  2.724e+00  -3.329  0.00087 ***
ATHLTCS[T.Y]   6.679e-01  2.641e-01   2.529  0.01143 *
AVE_INC        1.234e-05  4.696e-06   2.629  0.00857 **
CHILD[T.Y]     1.782e+00  4.134e-01   4.310 1.63e-05 ***
CNDN_PCT       8.086e-01  1.233e+00   0.656  0.51184
DEPT1[T.AG]    1.407e+00  1.529e+00   0.920  0.35764
DEPT1[T.AP]   -2.829e-01  8.255e-01  -0.343  0.73178
DEPT1[T.AR]    5.894e-01  1.028e+00   0.573  0.56645
DEPT1[T.CO]    3.149e-01  1.094e+00   0.288  0.77352
DEPT1[T.DE]    4.132e+00  1.695e+00   2.438  0.01479 *
DEPT1[T.ED]   -1.405e-01  1.150e+00  -0.122  0.90272
DEPT1[T.FN]    2.435e+00  1.727e+00   1.410  0.15853
DEPT1[T.FR]    4.229e-01  2.256e+00   0.187  0.85130
DEPT1[T.GR]    1.264e+00  1.059e+00   1.194  0.23263
DEPT1[T.HK]   -1.286e+01  5.415e+02  -0.024  0.98106
DEPT1[T.LA]    1.620e+01  8.111e+02   0.020  0.98407
DEPT1[T.LI]    2.047e+00  1.401e+00   1.462  0.14387
DEPT1[T.ME]    1.749e+01  1.455e+03   0.012  0.99041
DEPT1[T.MU]    3.618e-01  1.337e+00   0.271  0.78660
DEPT1[T.ND]    5.301e+00  1.633e+00   3.246  0.00117 **
DEPT1[T.NU]   -3.497e-01  1.044e+00  -0.335  0.73778
DEPT1[T.PE]   -4.918e-01  1.431e+00  -0.344  0.73117
DEPT1[T.PH]    1.382e+00  1.794e+00   0.770  0.44107
DEPT1[T.PO]    1.799e+01  4.550e+02   0.040  0.96847
DEPT1[T.RH]    1.653e+01  1.455e+03   0.011  0.99094
DEPT1[T.SC]    2.162e-01  1.122e+00   0.193  0.84721
DEPT1[T.SO]    2.328e-01  1.465e+00   0.159  0.87375
DWEL_VAL       1.559e-06  3.960e-07   3.936 8.29e-05 ***
ENG_PCT       -1.073e+00  6.864e-01  -1.563  0.11805
FACSTAFF[T.Y]  1.652e+00  2.539e-01   6.504 7.83e-11 ***
FACULTY1[T.AP] 1.519e+00  1.434e+00   1.059  0.28955
FACULTY1[T.AR] 1.386e+00  1.329e+00   1.043  0.29684
FACULTY1[T.CO] 1.963e+00  1.392e+00   1.410  0.15860
FACULTY1[T.DE]        NA         NA      NA       NA
FACULTY1[T.ED] 2.086e+00  1.439e+00   1.449  0.14723
FACULTY1[T.FR] 1.460e+00  2.452e+00   0.595  0.55151
FACULTY1[T.GR] 2.014e+00  1.292e+00   1.558  0.11912
FACULTY1[T.LA] -1.216e+01 8.111e+02  -0.015  0.98804
```

Table 9.5: (Continued)

```
FACULTY1[T.ME] -1.436e+01  1.455e+03  -0.010  0.99213
FACULTY1[T.ND]         NA         NA      NA       NA
FACULTY1[T.PH]         NA         NA      NA       NA
FACULTY1[T.PO]         NA         NA      NA       NA
FACULTY1[T.SC]  1.375e+00  1.471e+00   0.934  0.35026
HH_1PER         4.089e+00  2.126e+00   1.923  0.05447 .
HH_2PER         3.218e+00  2.094e+00   1.537  0.12430
HH_3PER         4.980e+00  2.422e+00   2.056  0.03981 *
HH_45PER        4.995e-01  2.354e+00   0.212  0.83195
MOV_DWEL       -1.512e+00  2.160e+00  -0.700  0.48382
OTHERACT[T.Y]   7.143e-02  2.235e-01   0.320  0.74923
OWN_PCT         1.734e-01  4.358e-01   0.398  0.69068
PARENT[T.Y]     5.840e-02  2.011e-01   0.290  0.77151
PROV[T.BC]     -2.333e-01  3.705e-01  -0.630  0.52898
PROV[T.MB]     -5.824e-01  8.783e-01  -0.663  0.50730
PROV[T.NB]     -1.474e+01  7.618e+02  -0.019  0.98456
PROV[T.NF]      1.184e+00  1.379e+00   0.859  0.39061
PROV[T.NS]      1.588e-01  1.539e+00   0.103  0.91783
PROV[T.NT]     -1.353e+01  1.455e+03  -0.009  0.99258
PROV[T.ON]     -7.819e-02  4.154e-01  -0.188  0.85070
PROV[T.QC]     -2.410e+00  9.235e-01  -2.609  0.00907 **
PROV[T.SK]     -1.020e+00  8.007e-01  -1.273  0.20284
SD_INC         -1.764e-06  7.805e-07  -2.259  0.02385 *
SEX[T.M]        4.267e-01  1.311e-01   3.255  0.00113 **
SPOUSE[T.Y]     1.977e-01  1.303e-01   1.517  0.12920
YRFDGR          4.079e-02  2.102e-02   1.941  0.05231 .
YRLDGR          2.173e-02  2.128e-02   1.021  0.30714
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2614.5  on 1885  degrees of freedom
Residual deviance: 1795.9  on 1825  degrees of freedom
AIC: 1917.9

Number of Fisher Scoring iterations: 14


> 1 - (WesLogis$deviance/WesLogis$null.deviance) # McFadden R2
[1] 0.3131091
```

Figure 9.9: The Stepwise Variable Selection Dialog



other way to amuse yourself for a few minutes (depending on the speed of your computer) as R chews up CPU cycles comparing different models. When R is done, the model summary shown in Table 9.6 will be printed.

The results indicate that the AIC has been reduced from 1917.9 for the maximal model to 1886.5 for the final model that emerges from the stepwise variable selection process. The McFadden R^2 is just under 0.30, still a high value. The model is also a lot easier to interpret, and is fairly consistent with our original mental model. The somewhat surprising variables are the geodemographic variables `HH_1PER`, `HH_2PER`, and `HH_3PER`. Combined they indicate that a Wesbrook-level donor is likely to live in areas where the size of the typical household is small. Our interpretation is that this is getting at the age of the donor, which is consistent with the strong effect of the years since the donor received his or her first UBC degree. In general, the results point to older, wealthy males who were involved in athletics at UBC, were in the UBC Dentistry, Law, or Medicine Faculties (or did not go to UBC at all), and have children or spouses who also went to UBC as the most likely Wesbrook donors.

Examine the cumulative response chart for the validation sample for the `WesLogis` and `WesStep` models. The target level for the lift chart is Y and the true response rate is 0.01 (9.10). The figure indicates that the reduction in variables from 23 to 13 has had no ill effect on the predictive capability of the model.

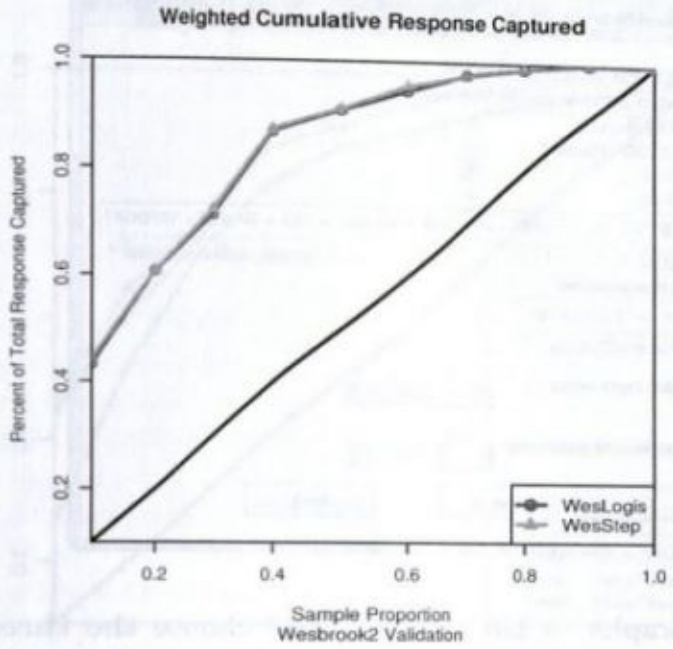
Step 4: Nonlinear Relationships

The main purpose of the neural network analysis is to determine if there are important nonlinear patterns between the target variable and the set of predictor variables we have selected based on the earlier regression and decision tree analyses. Since our objective is to assess the extent of nonlinear patterns,

Table 9.6: The Logistic Regression Results after Stepwise Variable Selection

```
Coefficients:
                Estimate Std. Error z value Pr(z)
(Intercept)    -6.521e+00  6.944e-01  -9.391  <2e-16 ***
ATHLTCS[T.Y]    6.610e-01  2.515e-01   2.629 0.008569 **
AVE_INC         1.060e-05  3.696e-06   2.868 0.004128 **
CHILD[T.Y]      1.721e+00  4.058e-01   4.240 2.23e-05 ***
DWEL_VAL        1.716e-06  3.213e-07   5.340 9.27e-08 ***
FACSTAFF[T.Y]   1.731e+00  2.497e-01   6.932 4.16e-12 ***
FACULTY1[T.AP] -6.398e-01  4.772e-01  -1.341 0.180026
FACULTY1[T.AR]  1.181e-01  4.426e-01   0.267 0.789512
FACULTY1[T.CO]  3.758e-01  4.723e-01   0.796 0.426185
FACULTY1[T.DE]  2.175e+00  6.220e-01   3.496 0.000472 ***
FACULTY1[T.ED] -9.490e-02  4.959e-01  -0.191 0.848224
FACULTY1[T.FR] -5.168e-02  6.472e-01  -0.080 0.936346
FACULTY1[T.GR]  5.769e-01  4.401e-01   1.311 0.189916
FACULTY1[T.LA]  2.092e+00  4.754e-01   4.400 1.08e-05 ***
FACULTY1[T.ME]  1.071e+00  4.804e-01   2.230 0.025756 *
FACULTY1[T.ND]  3.335e+00  4.611e-01   7.232 4.76e-13 ***
FACULTY1[T.PH] -5.532e-01  8.634e-01  -0.641 0.521742
FACULTY1[T.PO]  1.469e+01  2.651e+02   0.055 0.955802
FACULTY1[T.SC] -3.044e-01  6.444e-01  -0.472 0.636655
HH_1PER         3.089e+00  5.866e-01   5.266 1.39e-07 ***
HH_2PER         2.139e+00  7.552e-01   2.833 0.004614 **
HH_3PER         4.179e+00  1.578e+00   2.648 0.008107 **
SD_INC         -1.552e-06  7.252e-07  -2.140 0.032331 *
SEX[T.M]        4.058e-01  1.240e-01   3.273 0.001064 **
SPOUSE[T.Y]     1.960e-01  1.258e-01   1.558 0.119197
YRFDGR          6.169e-02  5.474e-03  11.268  < 2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2614.5  on 1885  degrees of freedom
Residual deviance: 1834.5  on 1860  degrees of freedom
AIC: 1886.5

Number of Fisher Scoring iterations: 13
> 1 - (WesStep$deviance/WesStep$null.deviance) # McFadden R^2
[1] 0.298345
```

Figure 9.10: The WesLogis and WesStep Cumulative Captured Response Chart

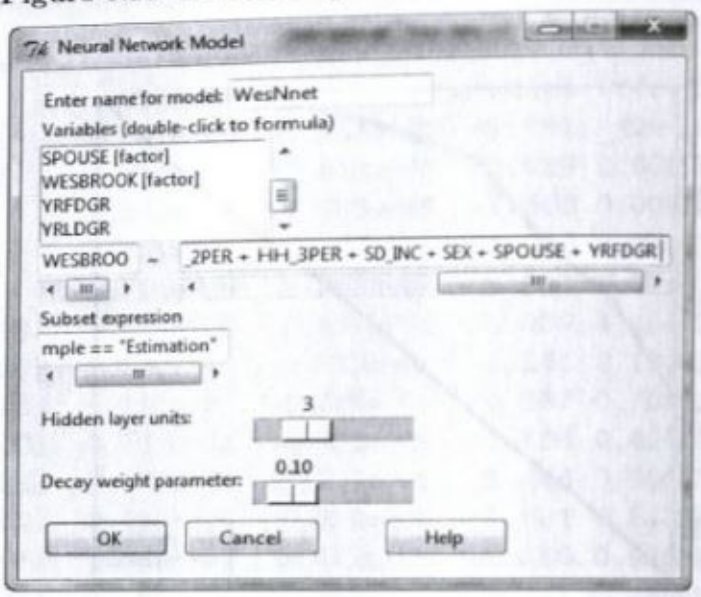


it is appropriate to use a network architecture that allows for a reasonably high level of nonlinearity, so we will use three nodes in the hidden layer. We will also only use the variables in the model obtained through stepwise variable selection: (1) `AHLTCS`; (2) `AVE_INC`; (3) `CHILD`; (4) `DWEL_VAL`; (5) `FACSTAFF`; (6) `FACULTY1`; (7) `HH_1PER`; (8) `HH_2PER`; (9) `HH_3PER`; (10) `SD_INC`; (11) `SEX`; (12) `SPOUSE`; and (13) `YRFDGR` (incidentally, this also includes the complete set of variables in our initial tree model).

You can try other variables as well, and probably would if time permitted. However, it is unlikely that you would find much improvement, and our goal is to find a "reasonably good" model. Note that the more hidden nodes you have, and the more variables, the longer it will take the neural network model to run. Moreover, and more important, the more likely the model will converge to a local optimum as opposed to a global optimum; in other words, the more likely the neural network model will fail for numerical reasons.

Select **Model → Machine Learning → Neural network model...** and in the dialog box name the model `WesNnet`, and make the subset expression `Sample == "Estimation,"` next select `WESBROOK` as the target, and the above variables selected in the stepwise regression as predictors. Set 3 hidden layer units, and the decay rate to 0.10 (Figure 9.11).

Figure 9.11: Estimating a Neural Network Model



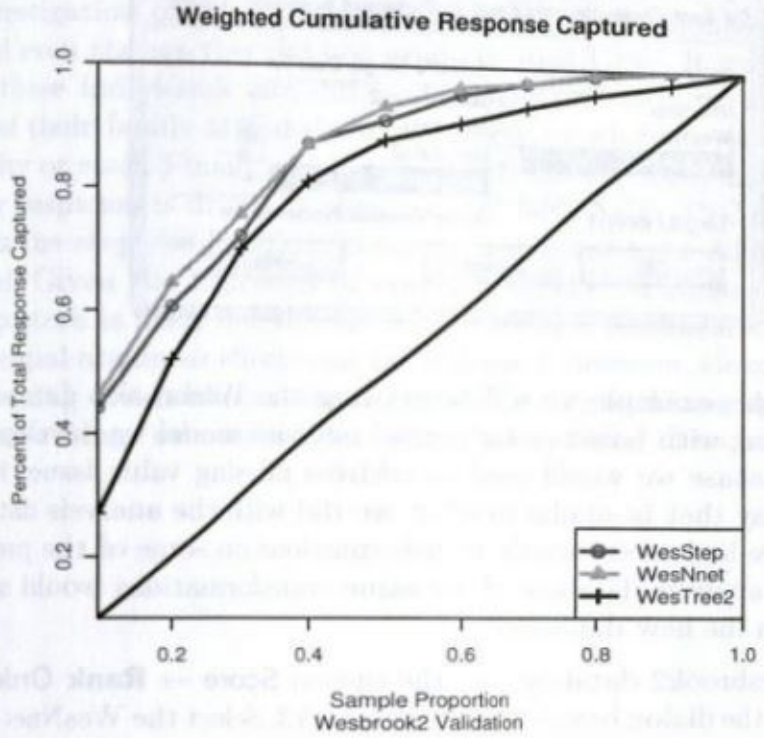
Select **Assess → Graphs → Lift chart...** and choose the three models `WesNnet`, `WesTree2`, and `WesStep`. Select `Wesbrook2` as the data set, and make sure that `Summary == "Validation"` is the subset expression. Set the remaining parameters as before. Click **OK**, and be patient, as it will take several seconds to plot the chart (Figure 9.12). An examination of this particular lift chart indicates that the neural network model does slightly better in the validation sample than the logistic regression model, which means there are some nonlinear patterns in the data that we could consider capturing.

An annoying technical detail is that you may not get exactly this result. In estimating the model, the neural network algorithm can get stuck in a local optimum, which is not the global optimum. In other words, it has not found the best solution. If you see a lift chart that is quite a bit below the others, the algorithm has been trapped in a local optimum. You may be able to find the global optimum, or at least close, by rerunning the model with a higher decay weight, say 0.2.

Step 5: Find the Source of Nonlinearities

If the neural network gives better lift, and we cared about which variables were having what type of effect (which we should, so that we can check that the model doesn't go greatly against managerial experience), we would continue with the methods we learned in Chapters 4 and 5). Specifically we would use the various graphical exploratory tools to examine relationships between WESBROOK and the predictor variables to try and find variables that might

Figure 9.12: The WesLogis and WesNnet Cumulative Response Chart

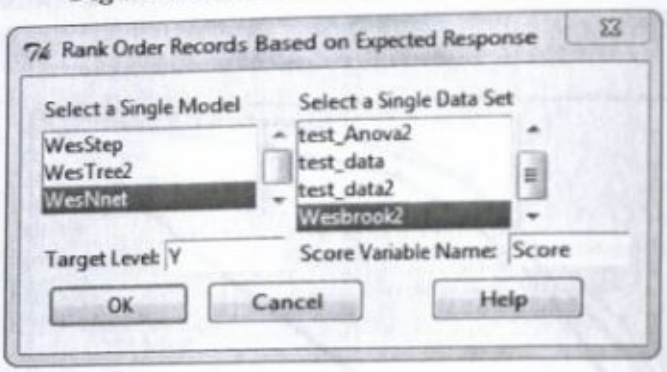


be transformed and entered into the logistic regression. We would work to find a logistic regression model that had the same validation lift as our best neural network, and which could then be interpreted and discussed with management. We will not repeat that analysis here, although the improvement in Figure 9.12 is just enough that it could be worthwhile if there were large numbers of customers.

Scoring the Database

The lift chart helps to decide how many to contact, but does not identify exactly whom they are. To really put it all together, you need to know whom to contact. The prediction and sorting that is done internally to create the lift chart needs to be added to the data set so that an identification variable of the most likely customers (donors in this case) can be matched up with their contact information, often located in another database to protect customers' privacy. Adding this additional information is referred to as *scoring* a database. The chosen predictive model can be used to score any database that has the same variables as used in the model, either the original database, or a brand new database where we do not have the target variable (and hence want to

Figure 9.13: Score a Database Dialog



predict it). In this example we will be scoring the Wesbrook2 database we have been working with based on the neural network model we developed. To score a new database we would need to address missing value issues in that database in a way that is similar to what we did with the analysis database. In addition, if we had used variable transformations on some of the predictor variables in the analysis database, those same transformations would need to be performed on the new database.

To score the Wesbrook2 database, use the option **Score** → **Rank Order** ..., which brings up the dialog box shown in Figure 9.13. Select the WesNnet model in the model selector, and Westbrook2 in the data set selector. The remaining default values should be appropriate. Once you have done this press **OK**. This will cause the new variable `Score` to be appended to the Wesbrook2 database. **View the data set to check to see if this was properly done**. The Score variable provides a rank order number that runs from 1 (best prospect) to 2698 (worst prospect) for each record in the Wesbrook2 database.

You may also wish to have access to the predicted probabilities, rather than merely the ordinal ranking score. To add those to the data set, select **Score→ Sample Estimated Probabilites . . .**, which will bring up a similar dialog box, and add the estimated response probabilities using the selected model and predictors from the selected data. If the sample has been oversampled and you wish to correct for that, select **Score→Adjusted Estimated Probabilities**, and put the true response rate in the dialog (as in the lift chart dialog). The estimated response rates in the original data will now be added to the data set.

Can a Better Model for the Wesbrook Data be Developed?

This model for the Wesbrook data is actually likely a very good "80%" model. We can think of three possible directions to follow to improve this model. First,

we would look at ways of sorting out the differences between DEPT1, FACULTY1, and MAJOR1. The second issue that can probably be fruitfully explored is a deeper investigation of the behavior of those people who donate at the Wesbrook level even though they did not graduate from UBC. It may make sense to divide these individuals into different subgroups based on whether other members of their family attended the university, or whether they are current UBC faculty or staff. Finally, our hunch is that income likely has a nonlinear effect. Our suspicion is driven by the fact that both AVE_INC and SD_INC are kept in the stepwise logistic regression model, but have different signs in that model. Given the high level of correlation between these variables, the observed pattern is likely due the model mimicking a nonlinear income effect. These potential nonlinear effects can be addressed. However, cleanly interpreting the meaning of a nonlinear transformation for a geo-demographic variable (which is an average over an area) is actually something of a challenge.