11.0 Linear Regression Modeling `Splus`

Notes based on those of Professor Joe Schafer

Main subject covered today:

- Linear regression modeling

11.1 lsfit()

This function is the "old" way to do linear regression in S and Splus. The syntax is

```
> result <- lsfit(x,y)
```

where x is a matrix of predictors and y is a vector of responses. The length of y should be the same as the number of rows of x. Unless you say otherwise, Splus will automatically append a column of 1s onto x so that the regression model will include an intercept. The result of this function is a list that contains various quantities summarizing the least-squares regression fit. For example,

```
> result$coef
```

will print the fitted coefficients (the $\hat{\beta}$), a vector of length ncol(x)+1, and

```
> result$residuals
```

will print the residuals, a vector of the same length as y. If you are analyzing a dataset, you will probably like to see a printout similar to that provided by other statistical packages (SAS, Minitab, ...) which includes a table of coefficients, standard errors and p-values, the R^2

of the regression, and the analysis of variance table. The function ls.print() will automatically generate this kind of printout:

```
> ls.print(result)
```

One of the big disadvantages of using lsfit() for linear regression is that if some of your predictor variables are categorical, you need to create dummy variables for them and put them into the x matrix yourself. If there are a large number of categorical predictors, this can get very tedious.

11.2 lm()

This is the newer version of linear regression in Splus. The name "lm" stands for linear model. This function operates on variables within a data frame. The syntax is

```
> result <- lm(formula, data=mydataframe)
```

where "formula" is a symbolic representation of the model to be fit (which I'll describe in a minute), and "mydataframe" is the name of the data frame in which the variables are located. If you attach the data frame to your search list, then you no longer need to supply the name of the data frame in the call to lm():

```
> attach(mydataframe,1)
>  result <- lm(formula)
```

The result of lm() is a list of objects that summarize the least-squares regression fit. To get a nice printout of the results of the regression, use the summary() function:

```
> summary(result)
```

Now we will talk about the "formula" part. Suppose that your data frame has a variable named Y which is the response, and predictors named X1, X2, etc. To build the standard additive model

$$Y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \ ... \ + \epsilon,$$

the formula statement is:

$$Y \sim X1 + X2 + \ ...$$

The tilde character ($\sim$) means, "is modeled as a function of". So the formula $Y \sim X1 + X2 + \ ...$ means that Y is modeled as an additive linear function of $X1; X2; \ ...$ To include the interaction between X1 and X2, you would say:

$$Y \sim X1 + X2 + X1 : X2 + \ ...$$

A shorthand version of the same thing is

$$Y \sim X1 * X2 + \ ...$$

where X1*X2 tells Splus to include both main effects X1 and X2 as well as the interaction X1:X2.

One advantage of using lm() rather than lsfit() is that if any of the predictor variables are factors (see notes from 10/28), then Splus automatically turns them into sets of dummy variables or contrasts.

This is true not only for main effects, but also for interactions as well. For example, suppose X1 is a numeric variable and X2 is a factor with 3 levels. The model $Y \sim X1 * X2$ will have the following predictors: X1, two dummy variables or contrasts for the main effects of X2, and two interaction terms that consist of the product of X1 with each of the terms for the main effect of X2.

11.3 Today's data

The dataset we will use today is an extract from the 1993 Fatal Accident Reporting System (FARS). FARS is a database maintained by National Highway Traffic Safety Administration which records every fatal accident that occurs on highways in the United States. There are about 40,000 accidents every year. For today's class, I extracted several variables pertaining to drivers of motorcycles. After eliminating some cases with missing data, we are left with 1450 motorcycle drivers. The data, which I already stored in a data frame called MOT, contains the following variables.

```
   Factors:

sex = driver's sex (M, F)
sev = injury severity (non-fatal, fatal)
lstat = valid driver's license? (no, yes)
day = day of accident (Mon, Tue, ..., Sun)
helmet = wearing a helmet? (no, yes)
rdwy = did accident occur on or off the roadway?  (off, on)
```

```
   Numeric variables:

age = driver's age
hr = hour of accident (1=6am, ..., 24=5am)
drrec = number of previous infractions on driver's record
bac = blood alcohol content*100
```

For today's class, we will regard bac as the response and all other variables as potential predictors, and we will attempt to model bac using techniques of ordinary linear regression.

11.4 A problem with the response variable

If you generate a histogram for the bac variable,

```
> hist(MOT$bac)
```

you will find that it has an unusual distribution: about half of the observed variables are zero, and the rest are distributed in a continuous fashion between 1 and 44. This type of distribution is "semicontinuous", a combination of a dichotomous variable and a continuous variable. To apply ordinary linear regression models to a semicontinuous variable would be incorrect, because the model's assumptions about the error distribution are clearly violated.

To solve this problem, we will eliminate the drivers with bac=0 from consideration. That is, we will build linear regression models for the reduced dataset

```
> MOT.nonzero <- MOT[MOT$bac!=0,]
```

with the understanding that our models will be predicting BAC not for all drivers, but only for those drivers who have been drinking.

In a future lecture, we will model the binary indicator for bac = 0 versus bac ¿ 0 using logistic regression.

11.5 Exploring the univariate distributions

Before fitting regression models, we will examine histograms for each of the variables in the data frame MOT.nonzero. We could simply try to write a loop that plots a histogram for each variable in this data frame. But that would result in some error messages, because some of the variables in this data frame are factors, and you cannot make a histogram from a factor. That is, if X1 is a factor variable, then

```
> hist(X1)
```

will produce an error message. One way to handle a factor variable is to make a barplot in which the heights of the bars are proportional to the number of observations in each level, with names under the bars corresponding to the factor levels:

```
> barplot(table(X1),names=levels(X1))
```

Here is a function that will take a data frame and create a histogram for each numeric variable and a barplot for each factor:

```
> allhists <- function(x)
+  for(i in 1:length(x))
+   if(is.factor(x[[i]]))
+    barplot(table(x[[i]]),names=levels(x[[i]]),main=names(x)[i],
+            density=-1)
+   else
+    hist(x[[i]],main=names(x)[i],density=-1)
        +
        +
+  invisible()
        +
```

Let's apply this function to our data frame MOT.nonzero:

```
> motif()
> par(mfrow=c(5,4))
> allhists(MOT.nonzero)
```

From these plots, we note the following features. First, there are essentially no females:

```
> table(MOT.nonzero$sex)
   M F
 709 8
```

Also, there are essentially no drivers who did not suffer fatal injuries:

```
>  table(MOT.nonzero$sev)
 non-fatal fatal
        28   689
```

So we probably won't be able to use these variables in our regression model. Also, the variable drrec has a very long right-hand tail:

```
> table(MOT.nonzero$drrec)
   0   1  2  3  4  5  6  7 8 9 10 11 12 13 14 19
```

```
 310 152 86 71 42 12 18 13 2 3  1  1  3  1  1  1
```

There is very little information in the dataset to distinguish among those with a large value of drrec, so let's recode this variable as a factor with only three levels: 0, 1, and 2+. Finally, we note that the distribution of bac is skewed to the right, so we should probably take some kind of transformation. For theoretical reasons it would be nice to use log(bac), because a linear model for log(bac) will never predict negative values for bac. But if we look at the histogram for log(bac),

```
> hist(log(MOT.nonzero$bac))
```

we find that it is skewed to the left. The transformation log(bac)$\hat{2}$ looks better,

```
> hist(log(MOT.nonzero$bac)2̂)
```

and log(bac)$\hat{3}$ looks even better:

```
> hist(log(MOT.nonzero$bac)3̂)
```

So we will work with log(bac)$\hat{3}$ rather than bac itself. Let's go into the MOT.nonzero data frame and make the necessary changes:

```
> attach(MOT.nonzero,1)
> sex <- NULL
> sev <- NULL
> logbac3 <- log(bac)3̂
> bac <- NULL
> drrec[drrec>2] <- 2
> drrec <- factor(drrec)
> levels(drrec) <- c("0","1","2+")
> detach(1,save="MOT.nonzero")
```

11.6 Exploring the relationships between the response and predictors

Before building a regression model, it helps to have some idea how each of the predictors is individually related to the response. We can do this using the plot() command.

We already learned that if X1 and Y are vectors, plot(X1,Y) will create a scatterplot with X1 on the horizontal axis and Y on the vertical axis. But there is another handy syntax that we can use with plot() that is well-suited to variables in a data frame. If you say plot(Y ∼ X1), then Splus will do the same thing as plot(X1,Y)–i.e. create a scatterplot–if Y and X1 are both numeric variables. But if X1 is a factor, plot(Y ∼ X1) will create side-by-side boxplots of Y within each level of X1; that is, it will do the same thing as boxplot(split(Y,X1)).

Let's look at a plot of logbac3 versus age:

```
> par(mfrow=c(1,1))
> attach(MOT.nonzero,1)
> plot(logbac3 ~ age)
```

We can add the least-squares regression line to the plot like this:

```
> abline(lsfit(age,logbac3))  # plot the least-squares line
```

This line has a positive slope, indicating that there is a general tendency for logbac3 to increase with age. But we shouldn't necessarily believe that a linear model is appropriate here. To see whether a linear model is plausible, we can apply a nonparametric scatterplot smoother to display the trend. We can do this with the function scatter.smooth():

```
> scatter.smooth(logbac3 ~ age)
```

It is clear from this plot that logbac3 increases with age up to about age 40 and then starts to decrease, so in the regression model we should probably include linear and quadratic terms for age.

Let's look at plots of logbac3 versus each of the potential predictors. For the numeric predictors, we will plot the trends using scatter.smooth().

```
> par(mfrow=c(3,3))
> scatter.smooth(logbac3 ~ age)
> plot(logbac3 ~ day)
> plot(logbac3 ~ drrec)
> plot(logbac3 ~ helmet)
> scatter.smooth(logbac3 ~ hr)
> plot(logbac3 ~ lstat)
> plot(logbac3 ~ rdwy)
```

11.7 Performing the regression

Now we're ready to do the regression. First we'll create linear and quadratic terms for age. Instead of using age and age$\hat{2}$, which will be highly correlated, let's use the de-meaned versions:

```
> agel <- age-mean(age)
> ageq <- agel2̂
```

Now we're ready to fit a model.

```
> result <- lm(logbac3 ~ agel + ageq + day + drrec + helmet +
+    hr + lstat + rdwy)
> summary(result)
```

```
Call: lm(formula = logbac3 ~ agel + ageq + day + drrec +
         helmet + hr + lstat + rdwy)
Residuals:
    Min     1Q Median    3Q   Max
 -23.89 -7.832 0.1294 7.164 39.01

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  16.4620   1.4702    11.1974   0.0000
       agel   0.2629   0.0482     5.4512   0.0000
       ageq  -0.0165   0.0034    -4.8697   0.0000
       day1  -1.4827   0.9290    -1.5961   0.1109
       day2   0.4480   0.5387     0.8317   0.4059
       day3  -0.5629   0.3609    -1.5595   0.1193
       day4  -0.3024   0.2375    -1.2731   0.2034
       day5  -0.0748   0.1617    -0.4625   0.6439
       day6   0.0848   0.1380     0.6145   0.5391
     drrec1  -0.6799   0.5252    -1.2944   0.1960
     drrec2  -0.1663   0.2941    -0.5653   0.5721
     helmet  -0.0244   0.3997    -0.0611   0.9513
         hr   0.3053   0.0877     3.4825   0.0005
      lstat  -0.9495   0.4741    -2.0027   0.0456
       rdwy  -1.5782   0.4016    -3.9294   0.0001

Residual standard error: 10.51 on 702 degrees of freedom
Multiple R-Squared: 0.1046
F-statistic: 5.855 on 14 and 702 degrees of freedom, the
   p-value is 6.137e-11
```

We can see that Splus turned each factor into a set of dummy variables or contrasts. For example, the seven-level factor day was entered as six regressors: day1, ..., day6. What are these regressors?

It turns out that the Splus's default method for coding factors is something called Helmert contrasts. To see how a particular factor variable was coded, simply type contrasts(variable):

```
> contrasts(day)
```

```
    [,1] [,2] [,3] [,4] [,5] [,6]
Mon   -1   -1   -1   -1   -1   -1
Tue    1   -1   -1   -1   -1   -1
Wed    0    2   -1   -1   -1   -1
Thu    0    0    3   -1   -1   -1
Fri    0    0    0    4   -1   -1
Sat    0    0    0    0    5   -1
Sun    0    0    0    0    0    6
```

From this table, we see that day1 is a regressor that is equal to -1 if day=Monday, 1 if day=Tuesday, and 0 otherwise; day2 is a regressor equal to -1 if day=Monday or day=Tuesday, 2 if day=Wednesday, and 0 otherwise; and so on. Thus the coefficient for day1 is a contrast for Tuesday versus Monday; the coefficient for day2 is a contrast for Wednesday versus the average of Monday and Tuesday; and so on.

For most users, Helmert contrasts are not the most familiar coding method. Splus has four built-in coding methods: polynomials, Helmert contrasts, “sum contrasts” (which many people would call “effect-coding”), and “treatment contrasts” (which many would call “dummy coding”). To see what these are, let’s print out the various coding schemes for a factor variable with 4 levels:

```
> contr.poly(4)
             .L   .Q         .C
[1,] -0.6708204  0.5 -0.2236068
[2,] -0.2236068 -0.5  0.6708204
[3,]  0.2236068 -0.5 -0.6708204
[4,]  0.6708204  0.5  0.2236068
> contr.helmert(4)
  [,1] [,2] [,3]
1   -1   -1   -1
2    1   -1   -1
3    0    2   -1
4    0    0    3
> contr.sum(4)
```

```
  [,1] [,2] [,3]
1    1    0    0
2    0    1    0
3    0    0    1
4   -1   -1   -1
> contr.treatment(4)
  2 3 4
1 0 0 0
2 1 0 0
3 0 1 0
4 0 0 1
```

Polynomial contrasts are appropriate for factors whose levels are ordered. For a 4-level factor, for example, the contrasts will give linear, quadratic, and cubic trends. “Sum contrasts” give the deviations of levels 1, 2, and 3 from the overall mean. “Treatment contrasts” treat the first level as a baseline and give the deviations of levels 2, 3, and 4 from this baseline.

To see what the default types of contrasts are, type options()$contrasts:

```
> options()$contrasts
          factor      ordered
 "contr.helmert" "contr.poly"
```

This tells us that the default contrasts for factors are Helmert contrasts, and the default contrasts for ordered factors are polynomial contrasts. (We haven’t talked about ordered factors yet. An ordered factor is like a factor, except that Splus considers the levels to be intrinsically ordered.) Suppose you want to make “treatment contrasts” the default. You would do this by saying

```
> options(contrasts=c("contr.treatment","contr.poly"))
```

which will change the default coding for factors to treatment contrasts for the duration of your Splus session.

11.8 Regression diagnostics

Suppose we fit a regression model like this:

```
> result <- lm(logbac3 ~ age + day + drrec + helmet + hr + lstat
        +   + rdwy)
```

The object result is a list whose elements summarize the regression. The functions fitted.values() and residuals() will extract the fitted values and residuals, respectively, from this list. So we can plot the residuals against the fitted values like this:

```
> plot(fitted.values(result),residuals(result))
```

Any good textbook on regression will mention a number of diagnostic quantities that allow you to assess the influence of each observation in the regression. The function lm.influence() will extract these quantities from result; see help(lm.influence) for details.

Suppose you want an ANOVA table with sequential sums of squares for each variable. You can get it by fitting the model with aov() rather than lm(), and applying summary() to the result:

```
> aovresult <- aov(logbac3 ~ age + day + drrec + helmet + hr +
        +   lstat + rdwy)
> summary(aovresult)
```

Suppose you want Splus to select a "best" subset of regressors using a stepwise procedure. The function step(), when applied to the result of an lm() call, will do this:

```
> result <- lm(logbac3 ~ age + day + drrec + helmet + hr +
        +   lstat + rdwy)
> step(result)
```