Lab 3: Linear Regression

Libraries

The `library()` function is used to load *libraries*, or groups of functions and data sets that are not included in the base `R` distribution. Basic functions that perform least squares linear regression and other simple analyses come standard with the base distribution, but more exotic functions require additional libraries. Here we load the `MASS` package, which is a very large collection of data sets and functions. We also load the `ISLR` package, which includes the data sets associated with this course. `library()`

```
> library(MASS)
> library(ISLR)
```

If you receive an error message when loading any of these libraries, it likely indicates that the corresponding library has not yet been installed on your system. Some libraries, such as `MASS`, come with `R` and do not need to be separately installed on your computer. However, other packages, such as `ISLR`, must be downloaded the first time they are used. This can be done directly from within `R`. For example, on a Windows system, select the `Install package` option under the `Packages` tab. After you select any mirror site, a list of available packages will appear. Simply select the package you wish to install and `R` will automatically download the package. Alternatively, this can be done at the `R` command line via `install.packages("ISLR")`. This installation only needs to be done the first time you use a package. However, the `library()` function must be called each time you wish to use a given package.

1 Interaction Terms

It is easy to include interaction terms in a linear model using the `lm()` function. The syntax `lstat:black` tells `R` to include an interaction term between `lstat` and `black`. The syntax `lstat*age` simultaneously includes `lstat`, `age`, and the interaction term `lstat`×`age` as predictors; it is a shorthand for `lstat+age+lstat:age`.

```
> summary(lm(medv∼lstat*age,data=Boston))

Call:
lm(formula = medv ∼ lstat * age, data = Boston)

Residuals:
   Min     1Q Median     3Q    Max
-15.81  -4.04  -1.33   2.08  27.55

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 36.088536   1.469835   24.55  < 2e-16 ***
lstat       -1.392117   0.167456   -8.31  8.8e-16 ***
age         -0.000721   0.019879   -0.04    0.971
lstat:age    0.004156   0.001852    2.24    0.025 *
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 6.15 on 502 degrees of freedom
Multiple R-squared: 0.556,  Adjusted R-squared: 0.553
F-statistic:  209 on 3 and 502 DF,  p-value: <2e-16
```

2 Non-linear Transformations of the Predictors

The `lm()` function can also accommodate non-linear transformations of the predictors. For instance, given a predictor X, we can create a predictor X^2 using `I(X^2)`. The function `I()` is needed since the `^` has a special meaning in a formula; wrapping as we do allows the standard usage in `R`, which is to raise `X` to the power `2`. We now perform a regression of `medv` onto `lstat` and `lstat`^{2}. `I()`

```
> lm.fit2=lm(medv∼lstat+I(lstat^2))
> summary(lm.fit2)

Call:
lm(formula = medv ∼ lstat + I(lstat^2))

Residuals:
   Min     1Q Median     3Q    Max
-15.28  -3.83  -0.53   2.31  25.41
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 42.86201    0.87208    49.1   <2e-16 ***
lstat       -2.33282    0.12380   -18.8   <2e-16 ***
I(lstat^2)   0.04355    0.00375    11.6   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.52 on 503 degrees of freedom
Multiple R-squared: 0.641,  Adjusted R-squared: 0.639
F-statistic:  449 on 2 and 503 DF,  p-value: <2e-16
```

The near-zero p-value associated with the quadratic term suggests that it leads to an improved model. We use the `anova()` function to further quantify the extent to which the quadratic fit is superior to the linear fit. `anova()`

```
> lm.fit=lm(medv~lstat)
> anova(lm.fit,lm.fit2)
Analysis of Variance Table

Model 1: medv ~ lstat
Model 2: medv ~ lstat + I(lstat^2)
  Res.Df   RSS Df Sum of Sq   F Pr(>F)
1    504 19472
2    503 15347  1      4125 135 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here Model 1 represents the linear submodel containing only one predictor, `lstat`, while Model 2 corresponds to the larger quadratic model that has two predictors, `lstat` and `lstat`^{2}. The `anova()` function performs a hypothesis test comparing the two models. The null hypothesis is that the two models fit the data equally well, and the alternative hypothesis is that the full model is superior. Here the F-statistic is 135 and the associated p-value is virtually zero. This provides very clear evidence that the model containing the predictors `lstat` and `lstat`^{2} is far superior to the model that only contains the predictor `lstat`. This is not surprising, since earlier we saw evidence for non-linearity in the relationship between `medv` and `lstat`. If we type

```
> par(mfrow=c(2,2))
> plot(lm.fit2)
```

then we see that when the `lstat`^{2} term is included in the model, there is little discernible pattern in the residuals.

In order to create a cubic fit, we can include a predictor of the form `I(X^3)`. However, this approach can start to get cumbersome for higher-order polynomials. A better approach involves using the `poly()` function to create the polynomial within `lm()`. For example, the following command produces a fifth-order polynomial fit: `poly()`

```
> lm.fit5=lm(medv~poly(lstat,5))
> summary(lm.fit5)

Call:
lm(formula = medv ~ poly(lstat, 5))

Residuals:
    Min       1Q   Median       3Q      Max
-13.543   -3.104   -0.705    2.084   27.115

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       22.533      0.232   97.20  < 2e-16 ***
poly(lstat, 5)1 -152.460      5.215  -29.24  < 2e-16 ***
poly(lstat, 5)2   64.227      5.215   12.32  < 2e-16 ***
poly(lstat, 5)3  -27.051      5.215   -5.19  3.1e-07 ***
poly(lstat, 5)4   25.452      5.215    4.88  1.4e-06 ***
poly(lstat, 5)5  -19.252      5.215   -3.69  0.00025 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.21 on 500 degrees of freedom
Multiple R-squared: 0.682,  Adjusted R-squared: 0.679
F-statistic:  214 on 5 and 500 DF,  p-value: <2e-16
```

This suggests that including additional polynomial terms, up to fifth order, leads to an improvement in the model fit! However, further investigation of the data reveals that no polynomial terms beyond fifth order have significant p-values in a regression fit.

Of course, we are in no way restricted to using polynomial transformations of the predictors. Here we try a log transformation.

```
> summary(lm(medv~log(rm),data=Boston))
...
```

3 *Qualitative Predictors*

We will now examine the `Carseats` data, which is part of the `ISLR` library. We will attempt to predict `Sales` (child car seat sales) in 400 locations based on a number of predictors.

```
> fix(Carseats)
> names(Carseats)
 [1] "Sales"       "CompPrice"   "Income"      "Advertising"
 [5] "Population"  "Price"       "ShelveLoc"   "Age"
 [9] "Education"   "Urban"       "US"
```

The `Carseats` data includes qualitative predictors such as `Shelveloc`, an indicator of the quality of the shelving location—that is, the space within a store in which the car seat is displayed—at each location. The predictor `Shelveloc` takes on three possible values, *Bad*, *Medium*, and *Good*.

Given a qualitative variable such as `Shelveloc`, `R` generates dummy variables automatically. Below we fit a multiple regression model that includes some interaction terms.

```
> lm.fit=lm(Sales~.+Income:Advertising+Price:Age,data=Carseats)
> summary(lm.fit)

Call:
lm(formula = Sales ~ . + Income:Advertising + Price:Age, data =
     Carseats)

Residuals:
   Min      1Q  Median      3Q     Max
-2.921  -0.750   0.018   0.675   3.341

Coefficients:
                    Estimate  Std. Error  t value  Pr(>|t|)
(Intercept)         6.575565    1.008747     6.52   2.2e-10 ***
CompPrice           0.092937    0.004118    22.57   < 2e-16 ***
Income              0.010894    0.002604     4.18   3.6e-05 ***
Advertising         0.070246    0.022609     3.11   0.00203 **
Population          0.000159    0.000368     0.43   0.66533
Price              -0.100806    0.007440   -13.55   < 2e-16 ***
ShelveLocGood       4.848676    0.152838    31.72   < 2e-16 ***
ShelveLocMedium     1.953262    0.125768    15.53   < 2e-16 ***
Age                -0.057947    0.015951    -3.63   0.00032 ***
Education          -0.020852    0.019613    -1.06   0.28836
UrbanYes            0.140160    0.112402     1.25   0.21317
USYes              -0.157557    0.148923    -1.06   0.29073
Income:Advertising  0.000751    0.000278     2.70   0.00729 **
Price:Age           0.000107    0.000133     0.80   0.42381
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.01 on 386 degrees of freedom
Multiple R-squared: 0.876,      Adjusted R-squared: 0.872
F-statistic:  210 on 13 and 386 DF,  p-value: <2e-16
```

The `contrasts()` function returns the coding that `R` uses for the dummy variables. `contrasts()`

```
> attach(Carseats)
> contrasts(ShelveLoc)
       Good  Medium
Bad       0       0
Good      1       0
Medium    0       1
```

Use `?contrasts` to learn about other contrasts, and how to set them.

`R` has created a `ShelveLocGood` dummy variable that takes on a value of 1 if the shelving location is good, and 0 otherwise. It has also created a `ShelveLocMedium` dummy variable that equals 1 if the shelving location is medium, and 0 otherwise. A bad shelving location corresponds to a zero for each of the two dummy variables. The fact that the coefficient for

ShelveLocGood in the regression output is positive indicates that a good shelving location is associated with high sales (relative to a bad location). And ShelveLocMedium has a smaller positive coefficient, indicating that a medium shelving location leads to higher sales than a bad shelving location but lower sales than a good shelving location.

4 *Writing Functions*

As we have seen, R comes with many useful functions, and still more func-tions are available by way of R libraries. However, we will often be interested in performing an operation for which no function is available. In this setting, we may want to write our own function. For instance, below we provide a simple function that reads in the ISLR and MASS libraries, called LoadLibraries(). Before we have created the function, R returns an error if we try to call it.

```
> LoadLibraries
Error: object 'LoadLibraries' not found
> LoadLibraries()
Error: could not find function "LoadLibraries"
```

We now create the function. Note that the + symbols are printed by R and should not be typed in. The { symbol informs R that multiple commands are about to be input. Hitting *Enter* after typing { will cause R to print the + symbol. We can then input as many commands as we wish, hitting *Enter* after each one. Finally the } symbol informs R that no further commands will be entered.

```
> LoadLibraries=function(){
+ library(ISLR)
+ library(MASS)
+ print("The libraries have been loaded.")
+ }
```

Now if we type in LoadLibraries, R will tell us what is in the function.

```
> LoadLibraries
function(){
library(ISLR)
library(MASS)
print("The libraries have been loaded.")
}
```

If we call the function, the libraries are loaded in and the print statement is output.

```
> LoadLibraries()
[1] "The libraries have been loaded."
```