Lecture 8: Fitting Data

Statistical Computing, 36-350
Wednesday October 7, 2015

In previous episodes

- Loading and saving data sets in R format
- Loading and saving data sets in other structured formats
- Intro to regression modeling

Today

- Using data frames for statistical purposes
- Basics of regression modeling

So you’ve got a data frame

Recall prostate cancer data set from last time:

```r
pros = read.table("http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data")
nrow(pros)
```

```
## [1] 97
```

```r
head(pros)
```

```
## 1 lcavol lweight age lbph svi lcp gleason pgg45 lpsa
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294 6 0 -0.4307829
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294 6 0 -0.1625189
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294 7 20 -0.1625189
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294 6 0 -0.1625189
## 5 0.7514161 3.432373 62 -1.386294 0 -1.386294 6 0 0.3715636
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294 6 0 0.7654678
```

```r
train
```

```
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

What can we do with it?

- Plot it: examine multiple variables and distributions
- Test it: compare groups of individuals to each other
• Model it: try to predict, say, one variable from the others

Recall the variables that we have measured over 97 men:

• lpsa: log PSA score
• lcavol: log cancer volume
• lweight: log prostate weight
• age: age of patient
• lbph: log of the amount of benign prostatic hyperplasia
• svi: seminal vesicle invasion
• lcp: log of capsular penetration
• gleason: Gleason score
• pgg45: percent of Gleason scores 4 or 5

Last time we plotted the distribution of each variable individually; and computed the correlations between variables; we learned that

• lcp, the log amount of capsular penetration, is very skewed; a bunch of men with little (or none?), then a big spread
• lcp and lcavol, the log cancer volume, are highly positive correlated
• gleason, which records the current Gleason score, and pgg45, which records the historical percentage of Gleason scores 4 or 5, are highly positively correlated

Visualizing multiple relationships, with pairs()

Can easily look at multiple relationships at once, using the pairs() function:

pairs(~ lcavol + lweight + lcp + lbph, data=pros)
Numerical correlations:

```r
cor(pros[,c("lcavol","lweight","lcp","lbph")])
```

```
## lcavol lweight lcp lbph
## lcavol 1.0000000 0.2805214 0.675310484 0.027349703
## lweight 0.2805214 1.0000000 0.164537142 0.442264399
## lcp 0.6753105 0.1645371 1.000000000 -0.006999431
## lbph 0.0273497 0.4422644 -0.006999431 1.000000000
```

Inspecting relationships over a subset of the observations

We see that lcp takes a bunch of really low values, that appear to have little relationship with other variables. For exploratory purposes, let’s get rid of them and see what the correlations look like

```r
pros.subset = pros[pros$lcp > min(pros$lcp),]
nrow(pros.subset) # Beware, we've lost a half of our data!
```

```
## [1] 52
```

```r
pairs(~ lcavol + lweight + lcp + lbph, data=pros.subset)
```
Numerical correlations:

```
cor(pros.subset[,c("lcavol","lweight","lcp","lbph")])
```

```
## lcavol lweight lcp lbph
## lcavol 1.0000000 0.2209352 0.8049728 -0.2407405
## lweight 0.2209352 1.0000000 0.1134501 0.3381993
## lcp 0.8049728 0.1134501 1.0000000 -0.1794318
## lbph -0.2407405 0.3381993 -0.1794318 1.0000000
```

(Go back to our doctor friend ... ask if this makes sense ... he tells us that when the recorded capsular penetration value is that small, it basically means that it couldn’t be accurately measured at that scale)

**Testing means between two different groups**

Recall that svi, the presence of seminal vesicle invasion, is binary:

```
table(pros$svi)
```

```
##
## 0  1
## 76 21
```

From [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1476128/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1476128/):
“When the pathologist’s report following radical prostatectomy describes seminal vesicle invasion (SVI) ... prostate cancer in the areolar connective tissue around the seminal vesicles and outside the prostate ... generally the outlook for the patient is poor.”

Does seminal vesicle invasion relate to the weight of the cancer?

Plot it first:

```r
pros$svi = factor(pros$svi)  # Making it a factor helps with plotting
plot(pros$svi, pros$lweight, main="lweight versus svi",
     xlab="SVI? Yes or no", ylab="Log cancer weight")
```

Tough to tell! Let’s try a simple two-sample t-test:

```r
t.test(pros$lweight[pros$svi==0], pros$lweight[pros$svi==1])
```

```r
## Welch Two Sample t-test
## data:  pros$lweight[pros$svi == 0] and pros$lweight[pros$svi == 1]
## t = -1.8266, df = 42.949, p-value = 0.07472
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```
## -0.33833495  0.01674335  
## sample estimates:
## mean of x  mean of y  
## 3.594131  3.754927  

Not much significance, it appears ...

What about the relationship to the volume of cancer?

```r
plot(pros$svi, pros$lcavol, main="lcavol versus svi",
     xlab="SVI? Yes or no", ylab="Log cancer volume")
```

Try a two-sample t-test again:

```r
t.test(pros$lcavol[pros$svi==0], pros$lcavol[pros$svi==1])
```

```r
## Welch Two Sample t-test  
## data:  pros$lcavol[pros$svi == 0] and pros$lcavol[pros$svi == 1]  
## t = -8.0351, df = 51.172, p-value = 1.251e-10  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:
```
## -1.917326 -1.150810
## sample estimates:
## mean of x mean of y
## 1.017892 2.551959

Wow, looks very significant! Should go report this to our doctor friend . . .

(Warning: what’s wrong with carrying out 5 more t-tests, for the rest of the variables, and taking the significant ones?)

### Basic regression modeling

Let’s use `lm()` to fit a basic linear model. Recall that this is a model of the form

\[ Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \text{noise} \]

for response variable \( Y \) and predictor variables \( X_1, \ldots, X_p \). Goal is to estimate the coefficients \( \beta_0, \beta_1, \ldots, \beta_p \).

The first argument of `lm()` is a formula, of the form \( Y \sim X_1 + X_2 + \ldots + X_p \). For example:

```r
pros.lm.1 = lm(lpsa ~ lcavol + lweight, data=pros)
class(pros.lm.1)
```

```r
## [1] "lm"
```

```r
names(pros.lm.1)
```

```r
## [1] "coefficients" "residuals" "effects" "rank" 
## [5] "fitted.values" "assign" "qr" "df.residual" 
## [9] "xlevels" "call" "terms" "model"
```

```r
pros.lm.1
```

```r
## Call:
## lm(formula = lpsa ~ lcavol + lweight, data = pros)
## 
## ## Coefficients:
## (Intercept)   lcavol   lweight
##   -0.8134   0.6515   0.6647
```

### Getting an overview with `summary()`

The function `summary()` gives us a nice summary of the linear model we just fit:

```r
summary(pros.lm.1)
```
### Call:
`lm(formula = lpsa ~ lcavol + lweight, data = pros)`

### Residuals:
```
Min    1Q  Median    3Q   Max
-1.61051 -0.44135 -0.04666 0.53542 1.90424
```

### Coefficients:
```
                Estimate Std. Error t value Pr(>|t|)  
(Intercept) -0.81344    0.65309  -1.246  0.216033  
lcavol      0.65154    0.06693   9.734 6.75e-16 ***
lweight     0.66472    0.18414   3.610  0.000494 ***
```

---

**Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1**

### Residual standard error: 0.7419 on 94 degrees of freedom
### Multiple R-squared: 0.5955, Adjusted R-squared: 0.5869
### F-statistic: 69.19 on 2 and 94 DF, p-value: < 2.2e-16

This tells us:
- The quantiles of the residuals: ideally, this is a perfect normal distribution
- The coefficient estimates
- Their standard errors
- P-values for their individual significance
- (Adjusted) R-squared value: how much variability is explained by the model?
- F-statistic for the overall model significance

### Getting rid of the intercept

To get rid of the intercept term, we add `+0` to the right-hand side of the formula:
```
pros.lm.2 = lm(lpsa ~ lcavol + lweight + 0, data=pros)
summary(pros.lm.2)
```

### Call:
`lm(formula = lpsa ~ lcavol + lweight + 0, data = pros)`

### Residuals:
```
Min    1Q  Median    3Q   Max
-1.63394 -0.51181  0.00925 0.49705 1.90715
```

### Coefficients:
```
                Estimate Std. Error t value Pr(>|t|)  
lcavol      0.66394    0.06638  10.00  <2e-16 ***
lweight     0.43894    0.03249  13.51  <2e-16 ***
```

---

**Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1**
Residual standard error: 0.7441 on 95 degrees of freedom
Multiple R-squared: 0.9273, Adjusted R-squared: 0.9258
F-statistic: 606.1 on 2 and 95 DF,  p-value: < 2.2e-16

Is the intercept appropriate? In general, depends on the variables that make up the linear model

**Getting estimated coefficients with coef()**

How can we access the values of estimated regression coefficients? Well, we could inspect the names of the fitted object:

```r
names(pros.lm.1)
```

```
[1]  "coefficients"  "residuals"  "effects"    "rank"
[5]  "fitted.values"  "assign"    "qr"        "df.residual"
[9]  "xlevels"        "call"      "terms"     "model"
```

Then grab the appropriate component, as in:

```r
pros.lm.1$coefficients
```

```
(Intercept)   lcavol  lweight
-0.8134373   0.6515421  0.6647215
```

But this is not the best strategy. Instead, let's use the `coef()` function:

```r
coef(pros.lm.1)
```

```
(Intercept)   lcavol  lweight
-0.8134373   0.6515421  0.6647215
```

Learning to use R's built-in functions like `coef()` to access certain aspects of fitted statistical models is going to pay off in the long run, because most object types (beyond those fit by `lm()`) will allow for this kind of access

**Getting the fitted values with fitted()**

What does our model predict for the log PSA scores of the 97 mean in question? How do these compare to the actual log PSA scores?

To answer the first question, we could extract the coefficients and then manually multiply and sum the appropriate values:

```r
pros.coef = coef(pros.lm.1)
pros.fits.1 = pros.coef[1] + pros.coef[2]*pros$lweight + pros.coef[3]*pros$lcavol
```
This is wrong, oops! We mixed up the order of variables; forgot that `lcavol` was supposed to come before `lweight`.

---

Try again:

```r
pros.fits.2 = pros.coef[1] + pros.coef[2]*pros$lcavol +
              pros.coef[3]*pros$lweight
```

This strategy is laborious and error-prone; it’s better to use `fitted()`, as in:

```r
pros.fits.3 = fitted(pros.lm.1)
max(abs(pros.fits.3 - pros.fits.2)) # They are the same
```

## [1] 1.332268e-15

For the same reasons as before, it’s good practice to use `fitted()`, a function that works with many common statistical objects in R.

---

Let’s plot the actual values versus the fitted ones.

```r
plot(pros.fits.3, pros$lpsa, main="Actual versus fitted",
     xlab="Fitted values", ylab="Log PSA values")
abline(a=0, b=1, lty=2, col="red")
```

![Actual versus fitted](image)

Appears to be a decent fit.
Running diagnostics with `plot()`

We can use the `plot()` function to run a series of diagnostic tests for our regression:

```r
plot(pros.lm.1)
```
The results are pretty good

- Residuals versus fitted plot: points appear randomly scattered, no particular pattern
• Normal Q-Q plot: points are more or less along the 45 degree line, so residuals look close to a normal distribution
• Scale-location and residuals versus leverage plots: points appear to be groups, no points are too far from the center

There is a science (and an art?) to interpreting these; you’ll learn a lot more in 36-401

Making predictions with \texttt{predict()}

Suppose we had a new observation from a man whose log cancer volume was 4.1, and log cancer weight was 4.5. What would our model estimate his log PSA score would be?

Two ways: manual way, and preferred way using \texttt{predict()}: 

\begin{verbatim}
pros.pred.1 = pros.coef[1] + pros.coef[2]*4.1 + 
             pros.coef[3]*4.5 
pros.new = data.frame(lcavol=4.1, lweight=4.5) 
pros.pred.2 = predict(pros.lm.1, newdata=pros.new) 
abs(pros.pred.2 - pros.pred.1) # They are the same
\end{verbatim}

\begin{verbatim}
## 1
## 0
\end{verbatim}

Building a bigger model

We only used two variables in our regression; what happens if we use all 8?

\begin{verbatim}
pros.lm.3 = lm(lpsa ~ lcavol + lweight + age + 
             lbph + svi + lcp + gleason + pgg45, 
             data=pros) 
summary(pros.lm.3)
\end{verbatim}

\begin{verbatim}
## Call:## lm(formula = lpsa ~ lcavol + lweight + age + 
##     lbph + svi + lcp + gleason + pgg45, data = pros)## ## Residuals:## ## Min  1Q Median  3Q Max## -1.76644 -0.35510 -0.00328 0.38087 1.55770## ## ## Coefficients:## ## (Intercept) 0.181561 1.320568 0.137 0.89096## lcavol 0.564341 0.087833 6.425 6.55e-09 ***## lweight 0.622020 0.200897 3.096 0.00263 **## age -0.021248 0.011084 -1.917 0.05848 .## lbph 0.096713 0.057913 1.670 0.24115## svi 0.761673 0.241176 3.158 0.00218 **## lcp -0.106051 0.089868 1.180 0.24115
\end{verbatim}
It’s unclear whether the addition variables added much, based on their p-values

**Comparing models based on test data**

A very robust a sensible way to pit models against each other is to compare their *test errors*. That is, we hold out some data, and predict the log PSA values of the unseen observations.

That last column in *pros*? Looks like the designers of the data set already defined training and test sets for us.

```r
head(pros)
```

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294 6 0 -0.4307829
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294 6 0 -0.1625189
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294 7 20 -0.1625189
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294 6 0 -0.1625189
## 5 0.7514161 3.432373 62 -1.386294 0 -1.386294 6 0 0.3715636
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294 6 0 0.7654678

```r
table(pros$train)
```

```r
 # FALSE TRUE
 # 30  67
```

Let’s build our two regression models, using only 67 men—our training data:

```r
pros.train = pros[pros$train==TRUE,]
pros.test = pros[pros$train==FALSE,]
pros.lm.train.1 = lm(lpsa ~ lcavol + lweight, data=pros.train)
pros.lm.train.2 = lm(lpsa ~ lcavol + lweight + age +
```
lbph + svi + lcp + gleason + pgg45, data=pros.train)
pros.pred.1 = predict(pros.lm.train.1, newdata=pros.test)
pros.pred.2 = predict(pros.lm.train.2, newdata=pros.test)
test.err.1 = mean((pros.test$lpsa - pros.pred.1)^2)
test.err.2 = mean((pros.test$lpsa - pros.pred.2)^2)

test.err.1
## [1] 0.4924823
test.err.2
## [1] 0.521274

We may as well go with the simpler model! Aside from being simpler (which is a plus!) it predicts better, too

**Summary**

- Plotting, exploring, and performing basic statistical tests in R is easy to do with data frames
- R has powerful regression modeling tools; using them is easy, using them properly requires some training and care (you’ll see a lot more in 36-401 and 36-402)