Interpreting Logistic Regression Coefficients with Examples in R

Benjamin Mako Hill (mako@atdot.cc) and Aaron Shaw (aaronshaw@northwestern.edu)

Contents

Because many people in this course wind up conducting and interpreting logistic regressions, I wanted to provide a quick overview of how to do that. I strongly recommend this page at UCLA that covers how to fit and interpret logistic regression as well as how to create model-predicted probabilities with R. In this document, I'll show how to do it both manually and with R in a bit less detail and complexity than the UCLA statistical consulting folks do.

Fitting a logistic model

First, let's use the built-in `mtcars` dataset to create a simple logistic regression on whether or not a car will have greater-than-average gas mileage. This means we're creating a new, dichotomous version of the `mpg` variable that just indicates whether each value is greater or less than the mean of the original `mpg` distribution.

```r
mako.cars <- mtcars
mako.cars$mpg.gtavg <- mako.cars$mpg > mean(mako.cars$mpg)
```

Now we can fit a logistic regression in which we regress our dichotomous "better than average gas mileage" variable on horsepower and number of gears.

```r
m <- glm(mpg.gtavg ~ hp + gear, data=mako.cars, family=binomial("logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(m)
```

```
##
## Call:
## glm(formula = mpg.gtavg ~ hp + gear, family = binomial("logit"),
##     data = mako.cars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.56988  -0.00001   0.00000   0.00843   1.53326
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  26.7692    17.5568   1.525   0.1273
## hp           -0.3387     0.1974  -1.716   0.0862 .
## gear          3.2290     2.6792   1.205   0.2281
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.8601  on 31  degrees of freedom
## Residual deviance:  5.8697  on 29  degrees of freedom
## AIC: 11.87
##
## Number of Fisher Scoring iterations: 10
```

As you can see, the standard errors are quite large. That said, we can still intrepret the coefficients for pedagogical purposes.

Intrepreting Coefficients

Interpret coefficients in logistic regression is different from an ordinary least squares model, but still relatively straightforward. If we look at the parameter estimate of the variable `hp`, we can see that the coefficient is -0.3387. Coefficients in logistic regression are *logged odds ratios*. The easiest/usual way to intrepret these is to exponentiate them using the `exp()` function to turn the coefficient into a normal odds ratio.

For example, if we look at the estimate for `hp` we would get:

```r
exp(-0.3378)
```

```
## [1] 0.7133379
```

Interpretation: In this case, we could say that our model estimates that a one unit increase in horsepower is associated with odds of being above average in gas mileage that are 0.71 times as large (i.e., 71% of the odds) as without the increased horsepower. That's a pretty substantial change!

Predicted Probabilities By Hand

Odds ratios are way easier to interpret than log-odds ratios, but can still be difficult (unless you come from a gambling family?). I (and others) always suggest that folks interpret logistical regression in terms of specific probabilities instead of just in terms of odds ratios.

Converting to probabilities from logistic regression is a bit complicated. The idea is basically that you will plug numbers into your fitted logistic regression and report the probabilities that your model predicts for what we might think of as hypothetical — or prototypical — individauls, who you can represent as combinations of values for the model predictors. Let's walk through how to do that and then we'll come back to the conceptual side of it.

The standard logistic function is:

$$\frac{1}{1+e^{-k}}$$

In other words, to turn the results from our logistic regression above back into a probability, we plug our model in for k:

$$\frac{1}{1+e^{-1(\beta_0+\beta_1\text{hp}+\beta_2\text{gear})}}$$

In our fitted model above, we know that $\beta_0 = 26.7692$, $\beta_1 = -0.3387$, $\beta_2 = 3.2290$. In order to convert these into probabilities, we use the logistic function. There's good information on how this works in practice in the Wikipedia article on logistic regression.

For this example, let's say your interested in the predicted probabilities for two "prototypical" cars: both with three gears and one with 100 horsepower and one with 120 horse power. First, lets plug in the numbers:

```
26.7692 + (-0.3387 * 100) + (3.2290 * 3) # a car with 100 hp and 3 gears
```

```
## [1] 2.5862
```

```
26.7692 + (-0.3387 * 120) + (3.2290 * 3) # a car with 120 hp and 3 gears
```

```
## [1] -4.1878
```

These numbers are equivalent to k in the first equation. We can now plug each value for k into that equation (and remember we have to use *negative* k):

```
1/(1+exp(-1*2.5862))
```

```
## [1] 0.9299681
```

```
1/(1+exp(-1*-4.1878))
```

```
## [1] 0.01495267
```

Interpretation: In other words, our model predicts that a car with three gears and 100 horse power will have above average mileage 93% of the time and a car with 120 horsepower will have above average mileage 1.5% of the time.

Created predicted probabilities with `predict()` in R

You can do the same thing in R using the `predict()` function. First, we make a dataset that includes the predictor values for the prototypical individuals whose outcomes would like to predict. For example, if I wanted to represent the two kinds of cars described above, I could do:

```
prototypical.cars <- data.frame(gear=3, hp=c(100, 120))
prototypical.cars
```

```
##   gear  hp
## 1    3 100
## 2    3 120
```

If I had more variables, I would need columns for each of them. In general, it's a good idea to hold any control variables at the median values observed in the sample for all of my "prototypical" individuals. This allows me to focus on the predicted change in the outcome associated with a change in just one (or a few) of my key predictors.

I can now use the `predict()` function to create a new variable. We use the option `type="response"` to have it give us predicted probabilities:

```
predict(m, prototypical.cars, type="response")
```

```
##         1         2
## 0.9296423 0.0148648
```

These numbers look extremely similar. They're not exactly the same because of rounding.