Information Criteria for Model Selection

2017-02-13

Preparation

We will use the data in the *mnSchools.csv* file. These data include institutional-level attributes for several Minnesota colleges and universities. The source of these data is: http://www.collegeresults.org. The attributes include:

- `id`: Institution ID number
- `name`: Institution name
- `gradRate`: Six-year graduation rate. This measure represents the proportion of first-time, full-time, bachelor's or equivalent degree-seeking students who started in Fall 2005 and graduated within 6 years.
- `public`: Dummy variable indicating educational sector (0 = private institution; 1 = public institution)
- `sat`: Estimated median SAT score for incoming freshmen at the institution
- `tuition`: Cost of attendance for full-time, first-time degree/certificate-seeking in-state undergraduate students living on campus for academic year 2013-14.

```
mn = read.csv(file = "~/Google Drive/Documents/github/EPsy-8252/data/mnSchools.csv")
head(mn)
```

id	name	gradRate	public	sat	tuition
1	Augsburg College	65.2	0	1030	39294
3	Bethany Lutheran College	52.6	0	1065	30480
4	Bethel University, Saint Paul, MN	73.3	0	1145	39400
5	Carleton College	92.6	0	1400	54265
6	College of Saint Benedict	81.1	0	1185	43198
7	Concordia College at Moorhead	69.4	0	1145	36590

Comparison of OLS Estimates to ML Estimates

Before we turn to the `mnSchools` data, let's go back to our toy example and consider a regression model using x to predict y. We will fit the model using the `lm()` function, which utilizes OLS estimation. We will also fit the model using the `mle2()` function from the **bbmle** package. This function obtains estimates using ML estimation. First we fit the `lm()` model.

```
x = c(4, 0, 3, 4, 7, 0, 0, 3, 0, 2)
y = c(53, 56, 37, 55, 50, 36, 22, 75, 37, 42)

# Fit model using OLS
lm.1 = lm(y ~ 1 + x)

# Output the coefficients, SEs, t-values, and p-values
summary(lm.1)$coefficients
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	40.01	6.34	6.31	0.000
x	2.74	1.98	1.39	0.203

```r
# Output the estimate of sigma_e
summary(lm.1)$sigma
```

```
## [1] 14
```

```r
# -2*LL
-2 * logLik(lm.1)[1]
```

```
## [1] 78.9
```

To use the `mle2()` function, we need to provide a user-written function that returns the negative log-likelihood given a set of parameter inputs. For simple regression, recall that we need to estimate three parameters: β_0, β_1, and σ_ϵ. Below we have a function that inputs the three parameters, uses the and returns the negative of the log-likelihood for a simple regression. (Note that the `dnorm()` function uses the `log=TRUE` argument to compute log densities.)

```r
regress.ll = function(b0, b1, sigma) {
  yhats = b0 + b1 * x
  errors = y - yhats
  neg_log_lik = -sum(dnorm(errors, mean = 0, sd = sigma, log = TRUE))
  return(neg_log_lik)
}
```

Now we can implement the `mle2()` function. This function requires the argument, `minuslogl=`, which gives the user written function returning the negative log-likelihood, it also requires a list of starting values for the input parameters in the user-written function. (Here we give values close to the OLS estimates as starting values.)

```r
# Fit model using ML
library(bbmle)
mle.results = mle2(minuslogl = regress.ll, start = list(b0 = 40, b1 = 2, sigma = 15))

# View results
summary(mle.results)
```

```
## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = regress.ll, start = list(b0 = 40, b1 = 2, sigma = 15))
##
## Coefficients:
##       Estimate Std. Error z value   Pr(z)
## b0       40.01       5.67    7.05 1.8e-12 ***
## b1        2.74       1.77    1.55    0.12
## sigma    12.51       2.80    4.47 7.8e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log L: 78.9
```

Compare the coefficient (`b0` and `b1`) estimates between the two methods of estimation. They are quite similar. The estimate of σ_ϵ is different (although close) between the two methods of estimation. This is because in OLS estimation, the estimate turns out to be,

$$\hat{\sigma}_\epsilon = \frac{\left(Y_i - \hat{Y}_i\right)^2}{n-2},$$

and for ML estimation, the estimate is,

$$\hat{\sigma}_\epsilon = \frac{\left(Y_i - \hat{Y}_i\right)^2}{n},$$

The smaller denominator in OLS results in a higher estimate of the variation.

This, in turn, affects the size of the SE estimates for the coefficients. Lastly, we note that the value of -2(log-likelihood) is the same for both the ML and OLS estimated models. This is a useful result. It allows us to use `lm()` to estimate the coefficients from a model and then use its log-likelihood as if we had fitted the model using ML. To see how this works in practice, let's fit a few models using the `mnSchools` data.

Fit Candidate Models

```
lm.a = lm(gradRate ~ 1 + public, data = mn)
lm.b = lm(gradRate ~ 1 + sat, data = mn)
lm.c = lm(gradRate ~ 1 + tuition, data = mn)
lm.d = lm(gradRate ~ 1 + public + sat, data = mn)
lm.e = lm(gradRate ~ 1 + tuition + sat, data = mn)
lm.f = lm(gradRate ~ 1 + public + tuition, data = mn)
lm.g = lm(gradRate ~ 1 + public + sat + tuition, data = mn)
```

Compute Log-Likelihood

Now we can compute the log-likelihood values for each model.

```
logLik(lm.a)
```

```
## 'log Lik.' -136 (df=3)
```

```
logLik(lm.b)
```

```
## 'log Lik.' -114 (df=3)
```

```
logLik(lm.c)
```

```
## 'log Lik.' -125 (df=3)
```

```
logLik(lm.d)
```

```
## 'log Lik.' -109 (df=4)
```

```
logLik(lm.e)
```

```
## 'log Lik.' -107 (df=4)
```

```
logLik(lm.f)
```

```
## 'log Lik.' -122 (df=4)
```

```
logLik(lm.g)
```

```
## 'log Lik.' -107 (df=5)
```

Compute Deviance

It is common to multiply these values by -2. This is called the *deviance*.

```
-2 * logLik(lm.a)
```

```
## 'log Lik.' 273 (df=3)
```

```
-2 * logLik(lm.b)
```

```
## 'log Lik.' 227 (df=3)
```

```
-2 * logLik(lm.c)
```

```
## 'log Lik.' 251 (df=3)
```

```
-2 * logLik(lm.d)
```

```
## 'log Lik.' 218 (df=4)
```

```
-2 * logLik(lm.e)
```

```
## 'log Lik.' 214 (df=4)
```

```
-2 * logLik(lm.f)
```

```
## 'log Lik.' 243 (df=4)
```

```
-2 * logLik(lm.g)
```

```
## 'log Lik.' 214 (df=5)
```

Compute AIC Values

Remember that the higher values of log-likelihood (lower values of deviance) indicate the parameters are more likely given the data. However, in this case we cannot directly compare the log-likelihoods/deviances since the models included a different number of parameters. To account for that, we will add a penalty term to the deviance,

$$AIC = -2 \times \text{LL} + 2(k)$$

where k is the number of parameters being estimated in the model (including the intercept and RMSE). Note that the value for k is given as *df* in the `logLik()` output. This value is called Akiake's Information Criteria (AIC). These values can be compared directly (so long as the same data is used and the same outcome). Smaller values of the AIC indicate a more likely model.

```
-2 * logLik(lm.a)[1] + 2*3
```

```
## [1] 279
```

```
-2 * logLik(lm.b)[1] + 2*3
```

```
## [1] 233
```

```
-2 * logLik(lm.c)[1] + 2*3
```

```
## [1] 257
```

```
-2 * logLik(lm.d)[1] + 2*4
```

```
## [1] 226
```

```
-2 * logLik(lm.e)[1] + 2*4
```

```
## [1] 222
```

```
-2 * logLik(lm.f)[1] + 2*4
```

```
## [1] 251
```

```
-2 * logLik(lm.g)[1] + 2*5
```

```
## [1] 224
```

Arranging these, we find that Model G (AIC = 224) is the most likely model given the data and the candidate set of models. This is the model that we should adopt.

Compute AICc Values

Although AIC has a penalty correction that should account for some bias, it turns out that when the number of parameters is large relative to the sample size, AIC is still biased in favor of models that have more parameters. This led Hurvich & Tsai (1989) to propose a second-order bias corrected AIC measure (AICc) computed as

$$\text{AIC}_\text{c} = \text{Deviance} + 2(k)\left(\frac{n}{n-k-1}\right)$$

where k is, again, the number of estimated parameters, and n is the sample size used to fit the model. Note that when n is very large (especially relative to k) that the last term is essentially 1 and the AICc value would basically reduce to the AIC value. When n is small relative to k this will add more of a penalty to the deviance. The recommendation is to pretty much always use AICc rather than AIC when selecting models. Below, we will compute the AICc for each of the six previosly fitted models. (Note that we use $n = 33$ cases for the computation for all the models in this data.)

```
n = 33

# Compute AICc for Model A, B, and C
k = 3
-2 * logLik(lm.a)[[1]] + 2 * k * n / (n - k - 1) # Model A
```

```
## [1] 280
```

```
-2 * logLik(lm.b)[[1]] + 2 * k * n / (n - k - 1) # Model B
```

```
## [1] 234
```

```
-2 * logLik(lm.c)[[1]] + 2 * k * n / (n - k - 1) # Model C
```

```
## [1] 258
```

```
# Compute AICc for Model D, E, and F
k = 4
-2 * logLik(lm.d)[[1]] + 2 * k * n / (n - k - 1) # Model D
```

```
## [1] 227
```

```
-2 * logLik(lm.e)[[1]] + 2 * k * n / (n - k - 1) # Model E
```

```
## [1] 223
```

```
-2 * logLik(lm.f)[[1]] + 2 * k * n / (n - k - 1) # Model F
```

```
## [1] 253
```

```
# Compute AICc for Model G
k = 5
-2 * logLik(lm.g)[[1]] + 2 * k * n / (n - k - 1)
```

```
## [1] 226
```

Based on the AIC_c values, we would adopt Model E. It is the most likely model given the data and the six candidate models.

Use AICc() Function

In practice, we will use the `AICc()` function from the **AICcmodavg** package to compute the AICc value directly.

```
library(AICcmodavg)

AICc(lm.a)
```

```
## [1] 280
```

```
AICc(lm.b)
```

```
## [1] 234
```

```
AICc(lm.c)
```

```
## [1] 258
```

```
AICc(lm.d)
```

```
## [1] 227
```

```
AICc(lm.e)
```

```
## [1] 223
```

```
AICc(lm.f)
```

```
## [1] 253
```

```
AICc(lm.g)
```

```
## [1] 226
```

References

Hurvich, C., & Tsai, C.-L. (1989). Regression and time series model selection in small samples, *Biometrika*, *76*, 297–307.