Long Hand Regression in R

Joel S Steele

Winter 2020

Linear Regression via Least Squares

So, in the following we will be computing the least square regression solution to a simple problem. To do this we will need a little background information about least squares solutions for linear models.

$$y = Xb \tag{1}$$

So, if the above is our set up, where y is the vector of outcomes and X is the design matrix of predictors, then b is another vector of coefficients that we need to solve for. Now, it is important to keep in mind that the equation above represents mathematical shorthand notation for the following situation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

Which is even more, slightly longer, shorthand for,

$$\begin{aligned} y_1 &= b_0 x_{11} + b_1 x_{12} \\ y_2 &= b_0 x_{21} + b_1 x_{22} \\ y_3 &= b_0 x_{31} + b_1 x_{32} \\ y_4 &= b_0 x_{41} + b_1 x_{42} \end{aligned}$$

which represents how we think each observation is generated, given our knowledge about the inputs, and our choice to use a line as the model of choice.

What we want to solve for are the values in the b vector. You may remember that the estimate of a regression slope, using least squares, turns out to be,

$$b = \frac{SCP_{xy}}{SS_x}.$$

Above, the SCP_{xy} is the Sum of Cross-products between x and y, and the SS_x is the Sum of Squares of x. If we scale these by their degrees of freedom we get,

$$b = \frac{Cov_{xy}}{Var_x},$$

or the covariance between x and y over the variance of x. That should look somewhat familiar, but here we are dealing with matrices and not single columns of data. So, our challenge will be how to translate the solution into code using the matrix solution. Which, based on our first equation is,

$$[X^T X]^{-1} X^T y = b \tag{2}$$

where $[X^T X]$ is the Sum of Squares of x, the $[X^T X]^{-1}$ performs the needed division part (one over Sum of Squares), and $X^T y$ is the Sum of Cross-products between x and y.

Below we generate some fake data to work with. We will be computing not only these steps but many of the steps to check the statistical significance of our computations as well. I will do my best to provide the needed background where applicable.

```r
# regression by hand
# fake data
med = c(0,1,3,4) # mom's education level
cgpa = c(3,3.2,3.3,3.7) # child gpa
x = cbind('int'=1,med) # design matrix
x
```

```
     int med
[1,]   1   0
[2,]   1   1
[3,]   1   3
[4,]   1   4
```

```r
t(x) %*% x # sum of squares and cross products
```

```
    int med
int   4   8
med   8  26
```

```r
t(x) %*% cgpa # sum of cross products for outcome and predictors
```

```
    [,1]
int 13.2
med 27.9
```

Side note!

Something worth knowing... I hope. In equation 2, we see that instead of dividing by our matrix of X^TX we instead multiplied by the matrix inverse. This is necessary because you can't directly divide by a matrix. Instead you compute the equivalent of 1 over the matrix, *aka* the inverse. So, how does this work? Well in a simple case, like with a diagonal matrix (see below) it requires that you take 1 over each value on the diagonal. It get's unreasonably more complicated if the matrix has entries in the off-diagonal, but for now let's illustrate the simple case.

```r
test = diag(c(2,4))
test # diagonal matrix, stuff just on the diagonal
```

```
     [,1] [,2]
[1,]    2    0
[2,]    0    4
```

```r
solve(test) # one over the matrix
```

```
     [,1] [,2]
[1,]  0.5 0.00
[2,]  0.0 0.25
```

```r
# the matrix times the matrix inverse... give us the matrix equivalent of 1
solve(test) %*% test # cool.. moving on
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Back to our regularly scheduled program

We have all the parts. Let's combine them. Just a reminder the equation is,

$$[X^T X]^{-1} X^T y = b.$$

```
# estimate params
bs = solve(t(x) %*% x) %*% t(x) %*% cgpa
bs
```

```
    [,1]
int 3.00
med 0.15
```

Great! So, now we have the estimates for the intercept and slope of our best fit line through these data. Next we will start working out the statistical tests for these estimates.

Predictions based on our model estimates

Now that we have our linear model, with parameter estimates tuned to our data, we can use it to predict what the model would expect our outcomes to be based on the inputs we have. Remember that this prediction is performed by running the inputs through our model

$$\hat{y} = b_0 + b_1 x_1$$

Or in our case

$$\hat{cgpa} = b_0(1) + b_1(med).$$

We can accomplish this in one step using matrix multiplication.

```
# check predictions
pred1 = x %*% bs
pred1
```

```
     [,1]
[1,] 3.00
[2,] 3.15
[3,] 3.45
[4,] 3.60
```

Now we can compute error as OBSERVED - EXPECTED.

```
err = cgpa - pred1
data.frame('obs'=cgpa, 'est'=pred1, 'err'=err) # quick look
```

```
  obs  est   err
1 3.0 3.00  0.00
2 3.2 3.15  0.05
3 3.3 3.45 -0.15
4 3.7 3.60  0.10
```

Recall that degrees of freedom for a linear model are computed as the difference between the number of input cases n and the number of model estimates p.

$$df = n - p$$

```
# degrees of freedom
# df = n           - p
mdf = length(med) - length(bs)
mdf
```

```
[1] 2
```

Given that we now have a vector of errors between our predictions and our observations, we can compute the Residual Standard Error. This is based on the following formula

$$S.E._{resid} = \sqrt{MS_{resid}} = \sqrt{\frac{\sum e_i^2}{df}}.$$

```
# residual sum of squares
errvar = sum(err^2)/mdf
errvar
```

```
[1] 0.0175
```

```
sqrt(errvar) # Resid std err
```

```
[1] 0.1322876
```

The residual standard error, or rather the variance of the error ($MS_{resid} = S.E._{resid}^2$), is also used to compute the standard errors for the beta estimates, based on the following formula,

$$S.E._b = \sqrt{\frac{MS_{resid}}{SS_x}}$$

```
# Standard Errors for betas
bvar = errvar * solve(t(x) %*% x)
bvar # we only care about the stuff on the diagonal
```

```
          int      med
int  0.011375 -0.00350
med -0.003500  0.00175
```

```
bse = sqrt(diag(bvar))
bse
```

```
      int       med
0.1066536 0.0418330
```

With our newly computed standard errors for the vector of bs, we can move onto our statistical test of these estimates. Below we compute a t statistic to test against the null hypothesis that these estimates are actually equal to zero.

$$t_b = \frac{b}{S.E._b}$$

```
# t-value
tcomp = bs/bse
tcomp
```

```
         [,1]
int 28.128434
med  3.585686
```

We test these values using a t-distribution with the appropriate degrees of freedom.

```
# p-values (two-tailed)
pcomp = 2*pt(abs(tcomp),mdf,lower.tail = F)
```

Let's pack this all up so it's easy to see.

```
# pack it all up
disp = data.frame(bs,bse,tcomp,pcomp)
disp
```

```
      bs       bse     tcomp       pcomp
int 3.00 0.1066536 28.128434 0.001261498
med 0.15 0.0418330  3.585686 0.069739491
```

Model evaluation

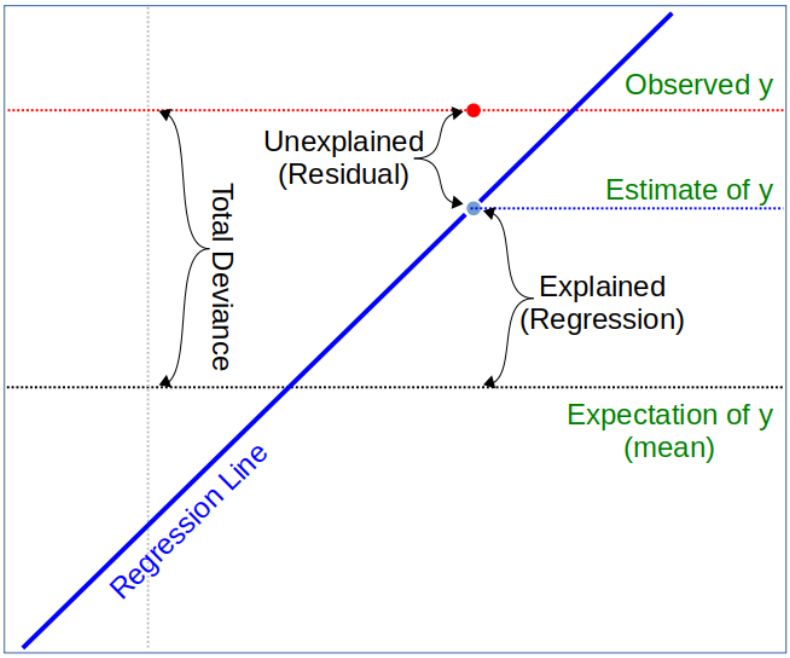
Great, so we have the estimates, how precise these estimates are, and how off our predictions would be by using these estimates. Our next question will be, is it worth it? Does using a linear model provide more information than just using the mean of the outcome?

For this we are comparing the amount of variance between our predictions and the mean, which is our explained portion, and comparing this to the difference between our observation and our prediction, which is our unexplained portion. Added together these equal the total variance of the observations. Have a look at Figure 1 for a visual of how things break down.

$$MS_{model} = \sigma^2_{model} = \frac{\sum(\hat{y} - \bar{y})^2}{p - 1}$$

where, $\hat{y}$ are the model predictions of y, $\bar{y}$ is the mean of y (or an uniformed model) and $p - 1$ are the degrees of freedom for the model, where p equals the number of predictors.

Figure 1: Model explained variance.



Note that in Figure 1, the red dot (*and line*) is an observation, the blue dot (*and line*) represents a prediction based on the regression model. Lastly, the black line represents the average of all y values or the mean of y. So, much like how we can split variance up in ANOVA into explained and unexplained, we do the same with regression predictions. The figure illustrates this breakdown for one observation.

```r
# F-test
# SS-model/regression sum((yhat-ybar)^2)
ss_mod=sum((pred1 - mean(cgpa))^2)
df_mod = length(bs)-1
# MS-model/reg
ms_mod = ss_mod/df_mod
ms_mod
```

```
[1] 0.225
```

We can now use this value in an F-test, were we see if the amount of variance in our regression is bigger than the variance of the errors from our model.

$$F_{regression}(df_{model}, df_{resid}) = \frac{MS_{model}}{MS_{resid}}$$

```r
# MS-resid
ms_res = errvar
df_res = mdf
# F value MS_mod/MS_resid
F_val = ms_mod/ms_res
F_val
```

```
[1] 12.85714
```

```r
p_val = pf(F_val,df_mod,mdf,lower.tail=F)
# display using our table from earlier, plus fanciness from knitr::kable()
kable(disp,digits=c(3,5,3,5))
```

	bs	bse	tcomp	pcomp
int	3.00	0.10665	28.128	0.00126
med	0.15	0.04183	3.586	0.06974

```r
# new display
cat(
  sprintf("Resid S.E.: %.4f on %d df\nF-value: %.2f on %d and %d df, p-value: %.5f\n"
          ,sqrt(ms_res), df_res, F_val, df_mod, df_res,p_val))
```

```
Resid S.E.: 0.1323 on 2 df
F-value: 12.86 on 1 and 2 df, p-value: 0.06974
```

Let's just double check to see if our computations make sense. Here we use the `lm()` function.

```r
# using the built in R function lm()
lm1 = lm(cgpa ~ med)
summary(lm1)
```

```
Call:
lm(formula = cgpa ~ med)

Residuals:
         1          2          3          4
-1.596e-16  5.000e-02 -1.500e-01  1.000e-01

Coefficients:
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.00000    0.10665  28.128  0.00126 **
med          0.15000    0.04183   3.586  0.06974 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1323 on 2 degrees of freedom
Multiple R-squared:  0.8654,    Adjusted R-squared:  0.7981
F-statistic: 12.86 on 1 and 2 DF,  p-value: 0.06974
```

Looks pretty good. Now you know.