ESL- linear regression notes

James Chuang

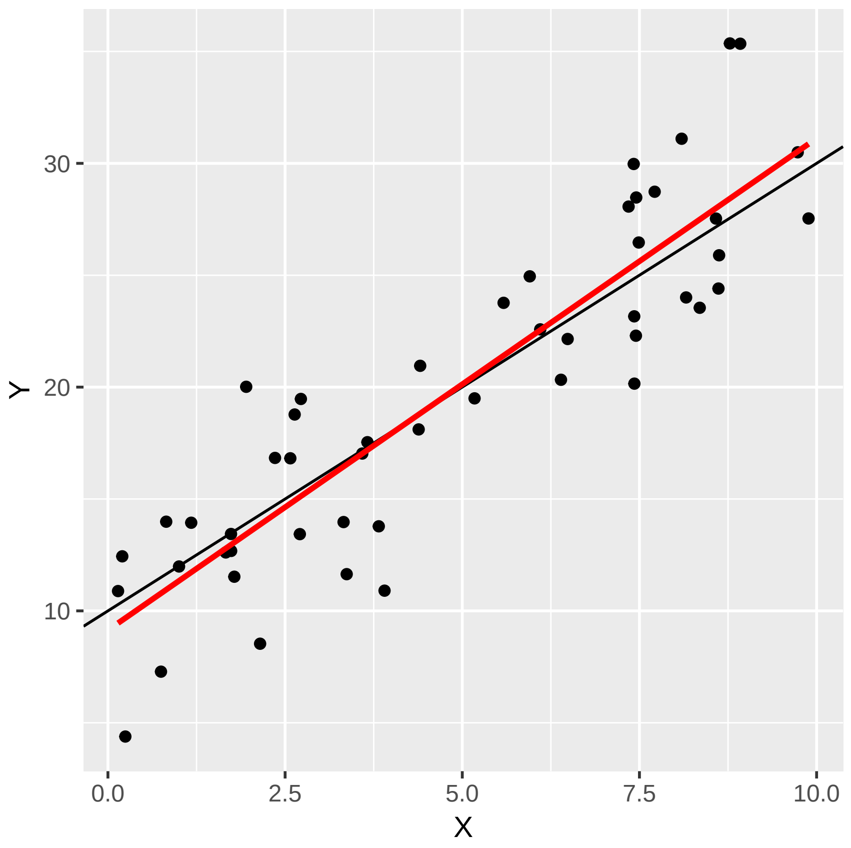
December 19, 2016

First, load some libraries and randomly generate $N = 50$ data points that are linearly related by $y = 10 + 2x$ with Gaussian noise:

```r
library(ggplot2)
library(viridis)

# N=number of samples or data points
N = 50

# randomly pick N x-values uniformly distributed from 1 to 10
X = matrix(runif(n=N, min=0, max=10))

# generate y-values for each x, y=3+2x with Gaussian noise
Y = matrix((10+2*X) + rnorm(n=N, mean = 0, sd = 3))
```

Plot the data to see what it looks like:

```r
lm.plot = ggplot(data = data.frame(cbind(X,Y)), aes(x=X1, y=X2)) +
            geom_point() +
            geom_abline(intercept = 10, slope = 2) +
            geom_smooth(method=lm, se=FALSE, color="red") +
            xlab("X") + ylab("Y")
lm.plot
```



Each data point is represented as a black dot. The black line shows the true relationship between X and Y from which the data were generated. With data from the real world, we do not know the true relationship, but would like to estimate it from the data. One way to do this is using linear regression by least squares. The red line shows the estimate of the true relationship by linear regression, known as the regression line. Since the data were generated from a linear relationship, the regression line is close to the true relationship, however, the regression line does not perfectly lie on the data generating line due to the noise or error present in the observed samples.

How is the location of the linear regression line calculated from the data? Practically, getting the linear regression only takes a single line in R:

```
lm.fit = lm(Y~X)      # fit a linear model to Y using X as the predictor, aka "regress Y on X"
```

We can look at what the fitted model contains:

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8135 -2.7650 -0.0167  2.6056  6.9316
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.1397     0.9250   9.881 3.74e-13 ***
## X             2.1977     0.1631  13.474  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 3.409 on 48 degrees of freedom
## Multiple R-squared:  0.7909, Adjusted R-squared:  0.7865
## F-statistic: 181.5 on 1 and 48 DF,  p-value: < 2.2e-16
```

I'll start by looking at the coefficient estimates for the intercept $(\hat{\beta}_0 = 9.1397)$ and the X variable $(\hat{\beta}_1 = 2.1977)$, which are pretty close to the actual $\beta_0 = 10$ and $\beta_1 = 2$ which generated the data. Let's see how these values are actually calculated.

The following is ripped from ESL Chapter 3.2. The linear regression model has the following form, and assumes that a linear model is a reasonable approximation for the regression function $\mathbf{E}(Y|X)$.

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

The β_j are unknown parameters to be estimated by linear regression. In the simple univariate case, $p = 1$, and so β_0 and β_1 are the intercept and slope of the regression line, respectively.

Typically we have a set of training data $(x_1, y_1)...(x_N, y_N)$ from which to estimate the parameters β. Each $x_i = (x_{i1}, x_{i2}, ..., x_{ip})^T$ is a vector of feature measurements for the *i*th case.

The most popular estimation method is *least squares*, in which the coefficients $\beta = (\beta_0, \beta_1, ..., \beta_p)^T$ are chosen to minimize the residual sum of squares RSS(β):

$$\begin{aligned}
\text{RSS}(\beta) &= \sum_{i=1}^{N} (y_i - f(x_i))^2 \\
&= \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2
\end{aligned}$$

If we include the constant variable 1 in X and include β_0 in the vector of coefficients β, RSS(β) can be represented in matrix format, where $\mathbf{X}$ is the $N \times (p+1)$ matrix with each row an input vector (with 1 in the first position), and $\mathbf{y}$ is the N-vector of outputs in the training set:

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

This is a quadratic function in the $p + 1$ parameters. In order to find the minimum, first differentiate with respect to β:

$$\begin{aligned}
\text{RSS}(\beta) &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \\
\text{RSS}(\beta) &= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta \\
\frac{\partial \text{RSS}}{\partial \beta} &= -2\mathbf{y}^T\mathbf{X} + 2\mathbf{X}^T\mathbf{X}\beta \\
\frac{\partial \text{RSS}}{\partial \beta} &= -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\beta \\
\frac{\partial \text{RSS}}{\partial \beta} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) \\
\frac{\partial \text{RSS}}{\partial \beta \partial \beta^T} &= 2\mathbf{X}^T\mathbf{X}
\end{aligned}$$

Assuming that $\mathbf{X}$ has full column rank, then $\mathbf{X}^T\mathbf{X}$ is positive definite. This means that the quadratic function RSS(β) is strictly convex, and so has a unique global minimum that we can solve for (in the univariate case, this is finding the minimum of a convex parabola). The position of the minimum can be found by setting the first derivative/gradient to zero:

$$\begin{aligned}\frac{\partial \text{RSS}}{\partial \beta} = -2\mathbf{X}^T(\mathbf{y}-\mathbf{X}\beta) = 0 \\ 2\mathbf{X}^T\mathbf{X}\beta = 2\mathbf{X}^T\mathbf{y} \\ \hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\end{aligned}$$

We can use this to calculate the least squares estimate for $\hat{\beta}$

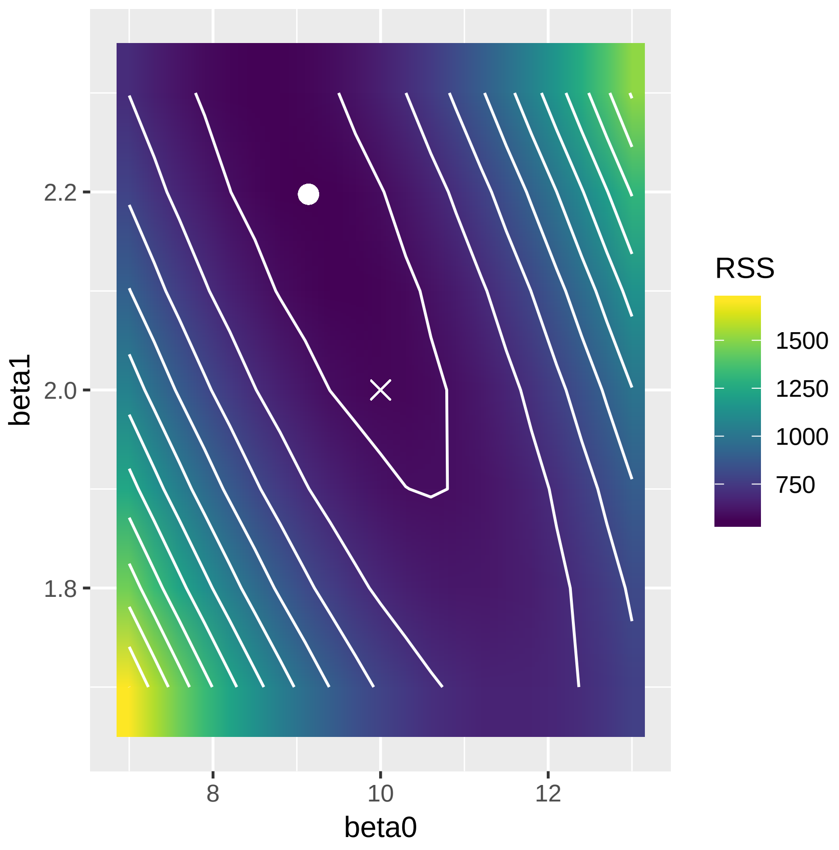
```r
X.int = cbind(1, X)       #add column of ones to X to account for intercept in beta vector
p = dim(X)[2]             #get dimension of the input (p=1)

beta.hat = (solve(t(X.int) %*% X.int)) %*% t(X.int) %*% Y
print(beta.hat)
```

```
##          [,1]
## [1,] 9.139706
## [2,] 2.197699
```

You can check that these coefficients are the same as those output by R's linear model fit above.

Here's a small aside in R to illustrate what least squares did. The code is not so important for understanding the points below.

```r
#generate matrix of beta vectors
beta = as.matrix(expand.grid(x=seq(7,13,by=0.3), y = seq(1.7,2.3, by=0.1)))
Y.broadcast = matrix(Y, nrow=N, ncol=dim(beta)[1])

#calculate RSS for each beta
RSS = diag(t(Y.broadcast-X.int%*%t(beta)) %*% (Y.broadcast-X.int%*%t(beta)))
RSS.data = data.frame(cbind(beta, matrix(RSS)))
names(RSS.data) = c("beta0","beta1","RSS")

RSS.plot = ggplot(data = RSS.data, aes(x=beta0, y=beta1, z = RSS, fill=RSS)) +
                    geom_raster(interpolate=TRUE) +
                    geom_contour(color="white") +
                    geom_point(x=10, y=2, shape=4, size=3, color="white") +
                    geom_point(x=coef(lm.fit)[1],
                               y=coef(lm.fit)[2],
                               size=3,
                               color="white") +
                    scale_fill_viridis()
RSS.plot
```



Here RSS(β) is plotted over a range of β_0 and β_1 values around their true values of 10 and 2. As promised by the equations above, RSS(β) is a convex quadratic function of β_0 and β_1 (think of a bowl or a valley). Least squares directly solves for the minimum error as measured by RSS by solving for the point $\hat{\beta}$ where the first derivative with respect to β is zero. This point is denoted by the white circle. It can be seen that in this case, the point $\hat{\beta}$ is pretty close to the true value of $\beta = (10, 2)^T$ represented by the white X, though the estimate is not perfect due to the noise in the training data.

Section 2

Now that we have $\hat{\beta}$, the fitted values at the training inputs are:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

How confident are we in the estimation? We will make a few assumptions in order to calculate the sampling properties of $\hat{\beta}$. First, we assume that the observations y_i are uncorrelated and have constant variance σ^2, and that the x_i are fixed (nonrandom). The variance-covariance matrix of the least squares parameter estimates is derived as follows:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \text{ and } y \sim N(\mathbf{X}\beta, \sigma^2\mathbf{I})$$

Therefore, $\hat{\beta}$ is also normally distributed, with mean

$$\mathbf{E}[\hat{\beta}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta$$
$$\mathbf{E}[\hat{\beta}] = \beta$$

and variance

$$\text{Var}(\hat{\beta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T$$
$$\text{Var}(\hat{\beta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}((\mathbf{X}^T\mathbf{X})^{-1})^T$$
$$\text{Var}(\hat{\beta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$$

Typically, the population measure σ^2 is unknown, so we estimate it by the sample variance $\hat{\sigma}^2$

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

, where the $N-p-1$ rather than N in the denominator makes $\hat{\sigma}^2$ an unbiased estimator of σ^2: $\mathbf{E}[\hat{\sigma}^2] = \sigma^2$

```
Y.trainpred = X.int %*% beta.hat
RSS.full = t(Y-Y.trainpred) %*% (Y-Y.trainpred)
sample.var = RSS.full/(N-p-1)
```

To test the hypothesis that a particular coefficient $\beta_j = 0$, we form the standardized coefficient or *T-score*

$$t_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}$$

, where v_j is the jth diagonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$. Under the null hypothesis that $\beta_j = 0$, $t_j \sim t_{N-p-1}$ (a t distribution with $N-p-1$ degrees of freedom). In R:

```
v = diag(solve(t(X.int)%*%X.int)) #calculate the matrix (X^T X)^-1
beta.std.err = (sample.var*v)^.5  #calculate standard errors, the denominator of the t-scores
beta.tstat = beta.hat/beta.std.err #calculate the t-scores
##get 2-tailed probability from t distribution
beta.tprob = 2*pt(q=beta.tstat, df=(N-p-1), lower.tail=FALSE)

print(beta.std.err)
```

```
## [1] 0.9249846 0.1631116
```

```
print(beta.tstat)
```

```
##           [,1]
## [1,]  9.880928
## [2,] 13.473594
```

```
print(beta.tprob)
```

```
##              [,1]
## [1,] 3.743588e-13
## [2,] 6.266499e-18
```

We can see that the values match those returned by R's linear fit:

```
coef(summary(lm.fit))
```

```
##             Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 9.139706  0.9249846  9.880928 3.743588e-13
## X           2.197699  0.1631116 13.473594 6.266499e-18
```

Often, we need to test for the significance of groups of coefficients simultaneously. To do this, we use the F statistic,

$$F = \frac{(\text{RSS}_0 - \text{RSS}_1)/(p_1 - p_0)}{\text{RSS}_1/(N - p_1 - 1)}$$

where RSS_1 is the residual sum-of-squares for the last squares fit of the bigger model with $p_1 + 1$ parameters, and RSS_0 the same for the nested smaller model with $p_0 + 1$ parameters, having $p_1 - p_0$ parameters constrained to be zero. In the example I made in R, the full model only has the two parameters β_0 and β_1. We can use the F-statistic to compare the full model with a smaller model which only uses β_0, i.e. a linear model with zero slope and y-intercept given by the mean of the data. Since the difference is only a single parameter, we will find that the significance of the F-statistic is exactly the same as the significance of β_1 calculated by the t-score above.

```r
Y.mean = apply(Y, 2, mean)  #smaller model using only the intercept
RSS.null = t(Y-Y.mean) %*% (Y-Y.mean) #RSS of smaller model
beta.fstat = ((RSS.null-RSS.full)/(p-0))/(RSS.full/(N-p-1)) #F-statistic full vs null models
#get probability from F-distribution
beta.fprob = pf(q=beta.fstat, df1 = p-0, df2 = N-p-1, lower.tail=FALSE)

print(beta.fstat)
```

```
##          [,1]
## [1,] 181.5377
```

```r
print(beta.fprob)
```

```
##              [,1]
## [1,] 6.266499e-18
```

As expected, the F-statistic and associated probability match those generated by R's linear fit, and the probability matches that calculated from the t-score of the slope.

```r
print(summary(lm.fit)$fstatistic)
```

```
##    value    numdf    dendf
## 181.5377   1.0000  48.0000
```

```r
print(summary(lm.fit))
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8135 -2.7650 -0.0167  2.6056  6.9316
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.1397     0.9250   9.881 3.74e-13 ***
## X             2.1977     0.1631  13.474  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.409 on 48 degrees of freedom
## Multiple R-squared:  0.7909, Adjusted R-squared:  0.7865
## F-statistic: 181.5 on 1 and 48 DF,  p-value: < 2.2e-16
```