Simple Linear Regression in R (Matrix Form)

Oct, 2019 STA 206

Getting Started

In this lab, we are going to explore the matrix form in simple linear regression and some basic matrix manipulations in R.

Simple Linear Regression in R

We have obtained the following data:

```
case  Y  X
   1 42  7
   2 33  4
   3 75 16
   4 28  3
   5 91 21
   6 55  8
```

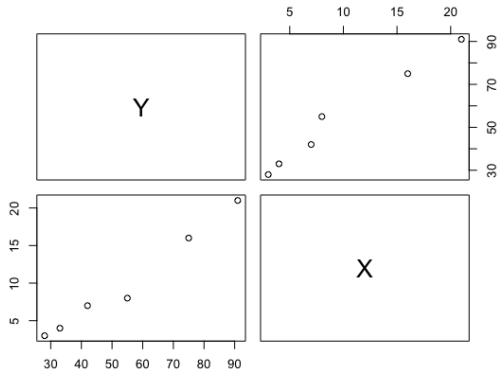
Let's start by first inputting the data into R, and then renaming the columns so that we don't get the variables confused:

```
dt = matrix(c(42, 33, 75, 28, 91, 55, 7, 4, 16, 3, 21, 8), 6, 2)
colnames(dt) = c('Y', 'X')
```

We have learned how to draw the scatter plot using the function "plot" to see what our data look like. In fact, we do have other ways of seeing how the variables are related (they are widely used in multiple linear regression):

```
pairs(dt)  ## pairwise scatter plots
cor(dt)    ## pairwise correlations
```

Here is the output for the first line:



and the second line gives us this matrix:

```
             Y         X
Y   1.0000000  0.9889386
X   0.9889386  1.0000000
```

Now that we have an idea about what our data look like, let's perform simple linear regression using matrix algebra in R. The first step is to specify the response vector $\mathbf{Y}$ and the design matrix $\mathbf{X}$ (notice here the dimensions of $\mathbf{X}$ are $n \times 2$, where 2 is the number of regression coefficients in our model (including intercept)).

```
Y = dt[,1] # create response variable
n = length(Y) # number of obeservations

X = cbind(rep(1,n), dt[,2])  #design matrix
```

Recall that the last line of code is saying "make a matrix that puts together a column of ones, along with columns 2 of our data." Similarly, we can use the *model.matrix* function:

```
X = model.matrix( dt[,1] ~ dt[,2] )
```

model.matrix is also useful for generating design matrices for higher-order terms e.g.

```
> model.matrix( dt[,1] ~ dt[,2]+I(dt[,2]^2) )
    (Intercept) dt[, 2] I(dt[, 2]^2)
1             1       7           49
2             1       4           16
3             1      16          256
4             1       3            9
5             1      21          441
6             1       8           64
attr(,"assign")
[1] 0 1 2
```

Now we have everything we need to find $\hat{\beta}$. We know that

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

and remember that $\hat{\beta}$ is now a 2×1 vector.

```
# solve t(X) %*% X %*% beta = t(X) %*% y, for beta
  beta.hat = solve(t(X) %*% X, t(X) %*% Y)
  beta.hat
          [,1]
[1,] 20.236102
[2,]  3.433617
```

As a side note, the *solve* function can also be used to invert matrices (e.g *solve(A)* will return the inverse of A, if it exists). However, it is faster to solve $Ax = b$ by *solve(A, b)* than with *solve(A)*b* (the former does not invert A which makes it more efficient).
We can also find the hat matrix, and check some of its properties (it should be symmetric and idempotent, and it should have rank $p = 2$). Recall

$$\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'.$$

```
  H = X %*% solve(t(X) %*% X, t(X))     # hat matrix
  library(Matrix)                       # load required package
  rankMatrix(H)[1]
  [1] 2                     # rank 2

  all.equal(H,t(H))
  [1] TRUE                  # symmetric
```

```
  all.equal(H,H %*% H)
  [1] TRUE                  # idempotent
```

So everything worked out.

Now we can obtain the fitted values, residuals, SSE and MSE. Recall

$$\hat{\mathbf{Y}} = \mathbf{HY},\ \mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}},\ SSE = \sum_{i=1}^{n} e_i^2,\ MSE = \frac{SSE}{n-2}.$$

```
Yhat = H %*% Y   # fitted values
e = Y - Yhat   # residuals
SSE = sum(e^2)   # SSE
MSE = SSE/(n-2)   # MSE
```

Generating Matrices

We often need to create special matrices in R for our computations. Here's some useful functions at our disposal.
The function *diag* creates a diagonal matrix with a given vector:

```
> A=rep(1,times=3)
> A
[1] 1 1 1
> diag(A)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

it can also extract the diagonal elements from a given matrix:

```
> X = matrix(1:9,3,3)
X
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> diag(X)
```

```
[1] 1 5 9
> diag( diag(X) )
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    5    0
[3,]    0    0    9
```

The function *matrix* builds a matrix from a given data set and dimensions:

```
> matrix( 0:1 , ncol=4, nrow=4  )
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    1    1    1    1
[3,]    0    0    0    0
[4,]    1    1    1    1


> matrix( 0:1, ncol=4, nrow=4, byrow=TRUE)
     [,1] [,2] [,3] [,4]
[1,]    0    1    0    1
[2,]    0    1    0    1
[3,]    0    1    0    1
[4,]    0    1    0    1
```

Dimension and Merging

The function *dim* returns the dimension of a matrix.

```
> X = matrix(1:9,3,3)
> dim(X)
[1] 3 3
```

Merging two matrices with appropriate dimensions can be done via *cbind* and *rbind*.

```
> Y = diag(1:3)
> Y
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

```
> cbind(X,Y)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    7    1    0    0
[2,]    2    5    8    0    2    0
[3,]    3    6    9    0    0    3

> rbind(X,Y)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
[4,]    1    0    0
[5,]    0    2    0
[6,]    0    0    3
```

Eigenvalues and Eigenvectors

We can also use R to compute eigenvalues and/or eigenvectors. For example

```
> B = matrix( rnorm(10*5), nrow=10,ncol=5 )
```

Here, we calculate eigenvalue and eigenvectors of $B^T B$

```
> ans = eigen( t(B)%*%B )
> ans
eigen() decomposition
$values
[1] 30.692642 21.203383  9.879783  6.090766  2.682052

$vectors
[,1]        [,2]        [,3]        [,4]       [,5]
[1,]  0.79970926  0.18795485  0.07086469  0.06128573  0.5624592
[2,] -0.06904318 -0.02434395 -0.94109994  0.26567966  0.1959225
[3,] -0.25331331 -0.58583876  0.28418430  0.54757600  0.4604619
[4,] -0.52491152  0.41015053  0.05621170 -0.37382459  0.6429151

> ans$values
```

```
[1] 30.692642 21.203383  9.879783  6.090766  2.682052
```

notice that all the eigenvalues are non-negative (why?).
The column vectors in the above matrix "$vectors" are eigenvectors

Also, *eigen* has an option to only compute the values

```
> eigen( t(B) %*% B, only.values = TRUE )$values
[1] 30.692642 21.203383  9.879783  6.090766  2.682052

$vectors
NULL
```

which is more efficient (for large matrices) if we are only interested in the eigenvalues.