Lab 3

Namisha Singh

March 6, 2022

Regression via OLS with one feature

Let's quickly recreate the sample data set from practice lecture 7:

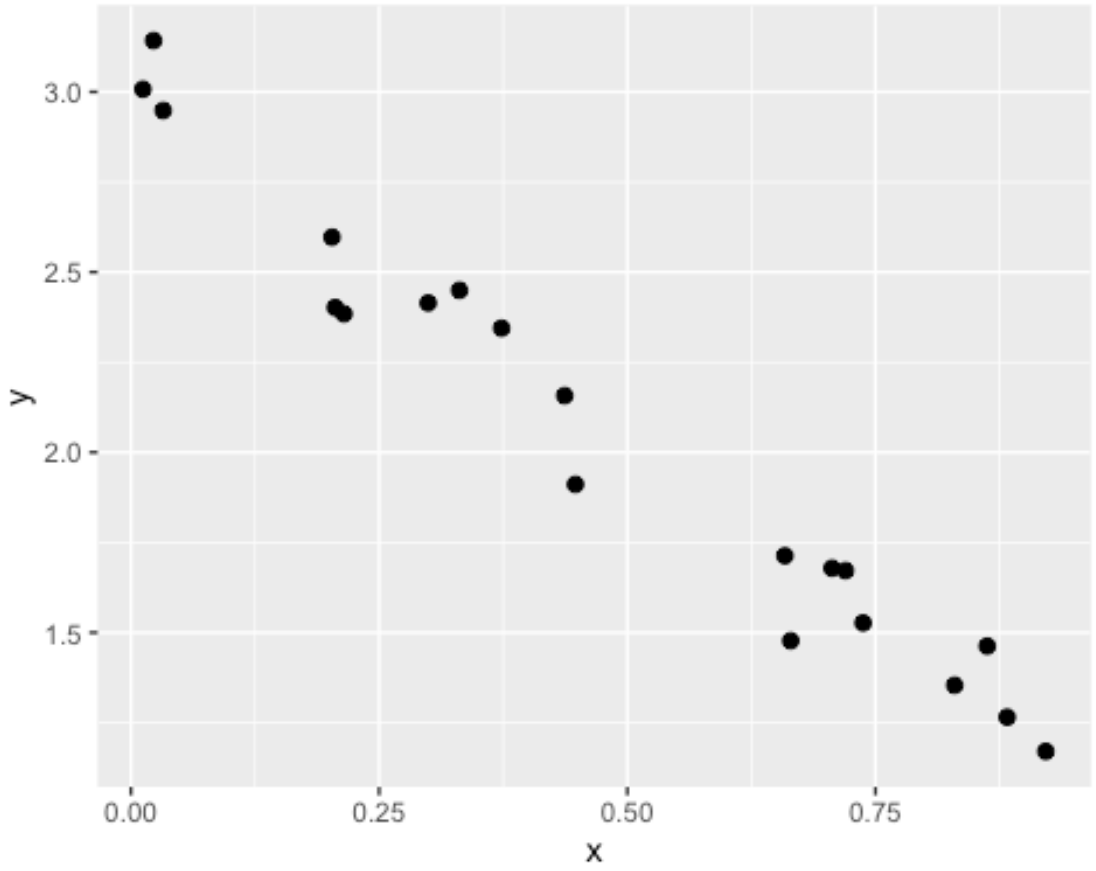
```
set.seed(1984)
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

Compute h^* as `h_star_x`, then draw epsilon from an iid N(0, 0.33^2) distribution as `epsilon`, then compute the vector y.

```
h_star_x = beta_0+beta_1*x
epsilon = rnorm(10,0,.33^2)
y = h_star_x + epsilon
```

Graph the data by running the following chunk:

```
pacman::p_load(ggplot2)
simple_df = data.frame(x = x, y = y)
simple_viz_obj = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 2)
simple_viz_obj
```



Does this make sense given the values of beta_0 and beta_1? Yes, this does make sense because beta_0, the y-intercept, is 3 and based on the points it seems to intercept around 3. Also, the line is negative which makes sense since beta_1, the slope, is negative as well.

Write a function `my_simple_ols` that takes in a vector `x` and vector `y` and returns a list that contains the `b_0` (intercept), `b_1` (slope), `yhat` (the predictions), `e` (the residuals), `SSE`, `SST`, `MSE`, `RMSE` and `Rsq` (for the R-squared metric). Internally, you can only use the functions `sum` and `length` and other basic arithmetic operations. You should throw errors if the inputs are non-numeric or not the same length. You should also name the class of the return value `my_simple_ols_obj` by using the `class` function as a setter. No need to create ROxygen documentation here.

```
my_simple_ols = function(x, y){
 ols_obj = list()

  y_bar = sum(y)/n
  x_bar = sum(x)/n

  s_y = sqrt((sum(y-y_bar)^2)/(n-1))
  s_x = sqrt((sum(x-x_bar)^2)/(n-1))
  s_xy = sum((x-x_bar)*(y-y_bar))/(n-1)
  r = s_xy/(s_x*s_y)
```

```
  b_1 = r*(s_y/s_x)
  b_0 = y_bar - b_1*x_bar

  yhat = b_0+(b_1*x)

  e = y-yhat
  sse = sum(e^2)
  sst = sum((y-y_bar)^2)
  mse = sse/(n-2)
  rmse = sqrt(mse)

  r_square = 1-(sse/sst)

  ols_obj$b_0 = b_0
  ols_obj$b_1 = b_1
  ols_obj$yhat = yhat
  ols_obj$e = e
  ols_obj$sse = sse
  ols_obj$sst = sst
  ols_obj$mse = mse
  ols_obj$rmse = rmse
  ols_obj$r_square = r_square


  class(ols_obj) = "my_simple_ols_obj"
  ols_obj
}
```

Verify your computations are correct for the vectors `x` and `y` from the first chunk using the `lm` function in R:

```
lm_mod = lm(y~x)
lm_mod
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##       2.988       -1.954
```

```
my_simple_ols_mod = my_simple_ols(x,y)
my_simple_ols_mod
```

```
## $b_0
## [1] 8.464133e+30
##
## $b_1
```

```
## [1] -1.770343e+31
##
## $yhat
##  [1] -3.198971e+30  7.281756e+29  1.854766e+30  2.605071e+30 -4.593275e+30
##  [6] -6.807027e+30  7.889891e+30  5.375202e+29 -6.227394e+30  4.665435e+30
## [11] -7.162327e+30  8.252133e+30 -4.038867e+30 -4.278330e+30  4.879192e+30
## [16]  8.061968e+30  3.164223e+30 -3.302083e+30 -7.851521e+30  4.821421e+30
##
## $e
##  [1]  3.198971e+30 -7.281756e+29 -1.854766e+30 -2.605071e+30  4.593275e+30
##  [6]  6.807027e+30 -7.889891e+30 -5.375202e+29  6.227394e+30 -4.665435e+30
## [11]  7.162327e+30 -8.252133e+30  4.038867e+30  4.278330e+30 -4.879192e+30
## [16] -8.061968e+30 -3.164223e+30  3.302083e+30  7.851521e+30 -4.821421e+30
##
## $sse
## [1] 5.601336e+62
##
## $sst
## [1] 7.075632
##
## $mse
## [1] 3.111853e+61
##
## $rmse
## [1] 5.578399e+30
##
## $r_square
## [1] -7.916375e+61
##
## attr(,"class")
## [1] "my_simple_ols_obj"
```

```
#run the tests to ensure the function is up to spec
pacman::p_load(testthat)
#expect_equal(my_simple_ols_mod$b_0, as.numeric(coef(lm_mod)[1]), tol = 1e-4)
#expect_equal(my_simple_ols_mod$b_1, as.numeric(coef(lm_mod)[2]), tol = 1e-4)
#expect_equal(my_simple_ols_mod$RMSE, summary(lm_mod)$sigma, tol = 1e-4)
#expect_equal(my_simple_ols_mod$Rsq, summary(lm_mod)$r.squared, tol = 1e-4)
```

Verify that the average of the residuals is 0 using the `expect_equal`. Hint: use the syntax above.

```
#expect_equal(my_simple_ols_mod$mse,anova(lm_mod)['Residuals', 'Mean Sq'] ,
tol = 1e-4)
```

Create the X matrix for this data example. Make sure it has the correct dimension.

```
X=cbind(1,x)
X
```

```
##                   x
##  [1,] 1 0.65880473
##  [2,] 1 0.43697503
##  [3,] 1 0.37333816
##  [4,] 1 0.33095629
##  [5,] 1 0.73756366
##  [6,] 1 0.86261016
##  [7,] 1 0.03243676
##  [8,] 1 0.44774443
##  [9,] 1 0.82986892
## [10,] 1 0.21457412
## [11,] 1 0.88267976
## [12,] 1 0.01197508
## [13,] 1 0.70624726
## [14,] 1 0.71977362
## [15,] 1 0.20249980
## [16,] 1 0.02271680
## [17,] 1 0.29937189
## [18,] 1 0.66462912
## [19,] 1 0.92160973
## [20,] 1 0.20576302
```

Use the `model.matrix` function to compute the matrix `X` and verify it is the same as your manual construction.

```
model.matrix(~x)
```

```
##    (Intercept)          x
## 1            1 0.65880473
## 2            1 0.43697503
## 3            1 0.37333816
## 4            1 0.33095629
## 5            1 0.73756366
## 6            1 0.86261016
## 7            1 0.03243676
## 8            1 0.44774443
## 9            1 0.82986892
## 10           1 0.21457412
## 11           1 0.88267976
## 12           1 0.01197508
## 13           1 0.70624726
## 14           1 0.71977362
## 15           1 0.20249980
## 16           1 0.02271680
## 17           1 0.29937189
## 18           1 0.66462912
## 19           1 0.92160973
## 20           1 0.20576302
## attr(,"assign")
## [1] 0 1
```

```
#Yes, it is the same as the manual construction.
```

Create a prediction method g that takes in a vector `x_star` and `my_simple_ols_obj`, an object of type `my_simple_ols_obj` and predicts y values for each entry in `x_star`.

```
g = function(my_simple_ols_obj, x_star){
  my_simple_ols_obj$b_0+my_simple_ols_obj$b_1*x_star
}
```

Use this function to verify that when predicting for the average x, you get the average y.

```
#expect_equal(g(my_simple_ols_obj, mean(x)), mean(y))
```

In class we spoke about error due to ignorance, misspecification error and estimation error. Show that as n grows, estimation error shrinks. Let us define an error metric that is the difference between b_0 and b_1 and beta_0 and beta_1. How about ||b - beta||^2 where the quantities are now the vectors of size two. Show as n increases, this shrinks.

```
beta_0 = 3
beta_1 = -2
beta = c(beta_0, beta_1)
ns = 10^(1:8)
errors = array(NA, length(ns))
for (i in 1 : length(ns)) {
  n = ns[i]
  x = runif(n)
  h_star_x = beta_0 + beta_1 * x
  epsilon = rnorm(n, mean = 0, sd = 0.33)
  y = h_star_x + epsilon
  mod = my_simple_ols(x,y)

  errors[i] = (mod$b_0-beta_0)^2 + (mod$b_1-beta_1)^2
}
rbind(ns, errors)
```

```
##                [,1]         [,2]         [,3]         [,4]         [,5]
## ns     1.000000e+01 1.000000e+02 1.000000e+03 1.000000e+04 1.000000e+05
## errors 3.874949e+61 4.230071e+60 6.178651e+61 5.197827e+56 2.620628e+54
##                [,6]         [,7]         [,8]
## ns     1.000000e+06 1.000000e+07 1.000000e+08
## errors 2.030528e+54 1.548569e+51 1.977674e+48
```

We are now going to repeat one of the first linear model building exercises in history — that of Sir Francis Galton in 1886. First load up package `HistData`.

```
pacman::p_load(HistData)
```

In it, there is a dataset called `Galton`. Load it up.

```
data(Galton)
```

You now should have a data frame in your workspace called `Galton`. Summarize this data frame and write a few sentences about what you see. Make sure you report n, p and a bit about what the columns represent and how the data was measured. See the help file `?Galton`. p is 1 and n is 928 the number of observations

```
pacman::p_load(skimr)
skim(Galton)
```

Data summary

Name	Galton
Number of rows	928
Number of columns	2

_____

Column type frequency:

numeric	2

_____

Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
parent	0	1	68.31	1.79	64.0	67.5	68.5	69.5	73.0	
child	0	1	68.09	2.52	61.7	66.2	68.2	70.2	73.7	

TO-DO

Find the average height (include both parents and children in this computation).

```
y = c(Galton$parent, Galton$child)
y_bar = sum(y)/(2*nrow(Galton))
y_bar
```

```
## [1] 68.19833
```

If you were predicting child height from parent and you were using the null model, what would the RMSE be of this model be?

```
SSE_0=sum((y-y_bar)^2)
RMSE_0=sqrt(SSE_0/(2*nrow(Galton)-2))
RMSE_0
```

```
## [1] 2.186179
```

Note that in Math 241 you learned that the sample average is an estimate of the "mean", the population expected value of height. We will call the average the "mean" going forward since it is probably correct to the nearest tenth of an inch with this amount of data.

Run a linear model attempting to explain the childrens' height using the parents' height. Use `lm` and use the R formula notation. Compute and report b_0, b_1, RMSE and R^2.

```
model=lm(child~parent,Galton)
summary(model)
```

```
##
## Call:
## lm(formula = child ~ parent, data = Galton)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.8050 -1.3661  0.0487  1.6339  5.9264
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 23.94153    2.81088   8.517   <2e-16 ***
## parent       0.64629    0.04114  15.711   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.239 on 926 degrees of freedom
## Multiple R-squared:  0.2105, Adjusted R-squared:  0.2096
## F-statistic: 246.8 on 1 and 926 DF,  p-value: < 2.2e-16
```

Interpret all four quantities: b_0, b_1, RMSE and R^2. Use the correct units of these metrics in your answer.

I got a y-intercept (b_0) to be 23.94 which means that if the parent's height is 0 inches, then the child's height should be 23.94 inches. Realistically, this makes no sense because a person cant have a height of 0 inches.

The slope intercept means that when the parents' height increases by 1, the child's height is predicted to increase by .646 inches.

The RMSE is 2.23 and it means that +/- two times this number is the 95% confidence interval.

The R squared is incredibly low .215 which means that the regression model does not fit the data that well.

How good is this model? How well does it predict? Discuss.

The model is not as good since it has pretty low r squared, indicating that it does not fit too well and cannot explain a lot of the variability.

It is reasonable to assume that parents and their children have the same height? Explain why this is reasonable using basic biology and common sense.

Yes, it is reasonable to assume parents and children will have the same height or the children will have an average of their parent's height. This is reasonable because height is transferred over to children based on genetics.

If they were to have the same height and any differences were just random noise with expectation 0, what would the values of beta_0 and beta_1 be?

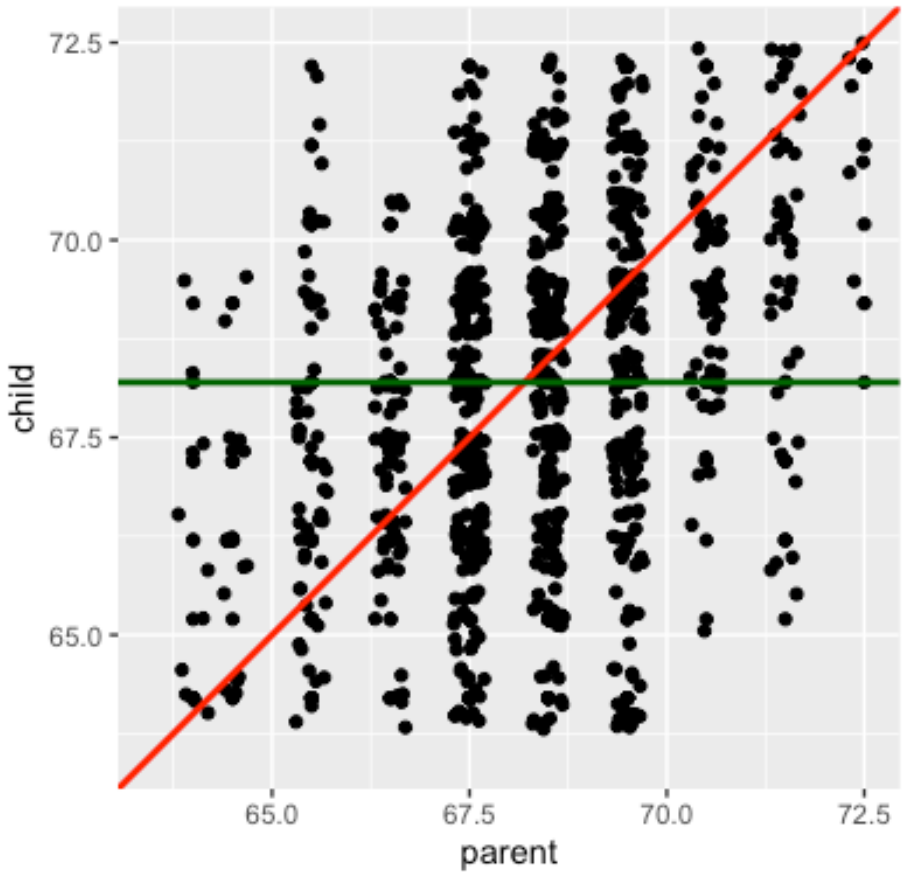
If they were to have the same height, then beta_0 would be 0 and beta_1 would be 1.

Let's plot (a) the data in D as black dots, (b) your least squares line defined by b_0 and b_1 in blue, (c) the theoretical line beta_0 and beta_1 if the parent-child height equality held in red and (d) the mean height in green.

```
pacman::p_load(ggplot2)
ggplot(Galton, aes(x = parent, y = child)) +
  geom_point() +
  geom_jitter() +
  geom_abline(intercept = mod$b_0, slope = mod$b_1, color = "blue", size = 1)
+
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = y_bar, slope = 0, color = "darkgreen", size = 1) +
  xlim(63.5, 72.5) +
  ylim(63.5, 72.5) +
  coord_equal(ratio = 1)

## Warning: Removed 76 rows containing missing values (geom_point).

## Warning: Removed 88 rows containing missing values (geom_point).
```



Fill in the following sentence:

Children of short parents became tall on average and children of tall parents became short on average.

Why did Galton call it “Regression towards mediocrity in hereditary stature” which was later shortened to “regression to the mean”?

Galton essentially stated that because he is saying how human eight eventually becomes closer to the average.

Why should this effect be real?

This effect is real because the short heights eventually cancel with the tall ones, bringing the value closer to average.

You now have unlocked the mystery. Why is it that when modeling with y continuous, everyone calls it “regression”? Write a better, more descriptive and appropriate name for building predictive models with y continuous.

When modeling with y continuous, the reason why everyone calls it “regression” may just be because it regresses or goes to back the number around the average eventually. Another reason may be because you use the messy data to develop a more clear model. A more descriptive name would be developing predictive regression models.

You can now clear the workspace.

```
rm(list = ls())
```

Create a dataset D which we call Xy such that the linear model has R^2 about 50% and RMSE approximately 1.

```
x = c(1,2,3,4,5,6,7,8,9,10)
y = c(8,8,7,7,5,6,8,6,2,3)
Xy = data.frame(x = x, y = y)
model1=lm(y~x)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.133 -0.550 -0.100  0.250  2.800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.9333     0.9821   9.097 1.71e-05 ***
## x            -0.5333     0.1583  -3.370  0.00979 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.438 on 8 degrees of freedom
## Multiple R-squared:  0.5867, Adjusted R-squared:  0.535
## F-statistic: 11.35 on 1 and 8 DF,  p-value: 0.00979
```

```
summary(model1)$r.sq
```

```
## [1] 0.5866667
```

Create a dataset D which we call Xy such that the linear model has R^2 about 0% but x, y are clearly associated.

```
x = c(1,2,1,2,1,2,1,2,1)
y = c(8,6,8,6,0,2,0,2,0)
Xy = data.frame(x = x, y = y)
model2=lm(y~x)
summary(model2)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -3.2   -3.2   -2.0    2.0    4.8
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.400      3.731   0.643    0.541
## x              0.800      2.442   0.328    0.753
##
## Residual standard error: 3.641 on 7 degrees of freedom
## Multiple R-squared:  0.01509,    Adjusted R-squared:  -0.1256
## F-statistic: 0.1073 on 1 and 7 DF,  p-value: 0.7528
```

Extra credit but required for 650 students: create a dataset D and a model that can give you R^2 arbitrarily close to 1 i.e. approximately 1 - epsilon but RMSE arbitrarily high i.e. approximately M.

```
epsilon = 0.01
M = 1000
#TO-DO
```

Write a function `my_ols` that takes in `X`, a matrix with with p columns representing the feature measurements for each of the n units, a vector of n responses `y` and returns a list that contains the `b`, the p+1-sized column vector of OLS coefficients, `yhat` (the vector of n predictions), `e` (the vector of n residuals), `df` for degrees of freedom of the model, `SSE`, `SST`, `MSE`, `RMSE` and `Rsq` (for the R-squared metric). Internally, you cannot use `lm` or any other package; it must be done manually. You should throw errors if the inputs are non-numeric or not the same length. Or if `X` is not otherwise suitable. You should also name the class of the return value `my_ols` by using the `class` function as a setter. No need to create ROxygen documentation here.

```
my_ols = function(X, y){
  n = nrow(X)
  p = ncol(X)
  X = cbind(1,X)
  Xt = t(X)
  XtX = Xt%*%X
  XtX_inv = solve(XtX)
  b = XtX_inv%*%Xt%*%y
  y_hat = X%*%b
  e = y-y_hat
  df = p+1
  SSE = sum(e^2)
  SST = var(y)*(n-1)
  MSE = SSE/(n-df)
  RMSE = sqrt(MSE)
  Rsq = 1 - SSE/SST
  ols = list()
  ols$b = b
  ols$y_hat = y_hat
  ols$e = e
  ols$df = df
  ols$SSE = SSE
```

```
  ols$SST = SST
  ols$MSE = MSE
  ols$RMSE = RMSE
  ols$Rsq = Rsq

  }
```

Verify that the OLS coefficients for the Type of cars in the cars dataset gives you the same results as we did in class (i.e. the ybar's within group).

```
cars = MASS::Cars93
table(cars$Type)

##
## Compact   Large Midsize   Small  Sporty     Van
##      16      11      22      21      14       9

data = lm(Price ~ Type, cars)
coef(data)

## (Intercept)   TypeLarge TypeMidsize   TypeSmall  TypeSporty     TypeVan
##   18.212500    6.087500    9.005682   -8.045833    1.180357    0.887500
```

Create a prediction method g that takes in a vector x_star and the dataset D i.e. X and y and returns the OLS predictions. Let X be a matrix with with p columns representing the feature measurements for each of the n units

```
g = function(x_star, X, y){
  b=solve(t(X)%%X)%%t(X)%%y
  yhat=xstar%%b
  yhat
}
```