Lab 3

Aracely Menjivar

3/6/2022

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 #true intercept
beta_1 = -2 #true slope
```

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 #true function
epsilon = rnorm(n, mean = 0, sd = 0.33)
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
```

Aracely-Menjivar_Lab3_files/figure-latex/unnamed-chunk-3-1.pdf

Does this make sense given the values of beta_0 and beta_1? yes it does make sense given that beta1 is negative

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){
  if(is.numeric(x)== FALSE || is.numeric(y)== FALSE){
    return("ERROR: inputs are non-numeric")
  }
```

```
  if(length(x) != length(y)){
    return("ERROR: length of the arguments are not the same")
  }
 ols_obj = list()
 n<- length(x) #x and y are the same length so just choose 1

 y_bar <- sum(y)/n
 x_bar<- sum(x)/n

 r <- s_xy/(s_x+ s_y) #correlation

 b_1 = (sum(x*y)-n*x_bar*y_bar) / (sum(x^2)-n*x_bar^2)
 b_0 = y_bar - b_1*x_bar

 y_hat= b_0 + b_1 * x
 e= y - y_hat

 SSE= sum(e^2)
 SST= SSE- (n-1) * s_y^2
 MSE= SSE/n
 RMSE= sqrt(MSE)
 Rsq= r^2 #proportion in var in y explained by x

ols_obj=list(b_0 = b_0, b_1= b_1, x_bar= x_bar, e=e, SSE= SSE, SST= SST, MSE= MSE, RMSE= RMSE, Rsq= Rsq
  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)
#my_simple_ols_mod = my_simple_ols(x,y)
#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$MSE, summary(lm_mod)$sigma^2)
```

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

```
X= cbind(1,x) # y-intercept cols of y
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
```

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){
  b_0= my_simple_ols_obj$b_0
  b_1= my_simple_ols_obj$b_1
  y_star= b_0 + b_1* x_star
  y_star
}
```

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

```
#expect_equal(g(my_simple_ols_mod, 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 = c(10, 15, 100, 200, 500)
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

 y_bar <- sum(y)/n
 x_bar<- sum(x)/n

 b_1 = (sum(x*y)-n*x_bar*y_bar) / (sum(x^2)-n*x_bar^2)
 b_0 = y_bar - b_1*x_bar
 b = c(b_0, b_1)
  errors[i] = sum((beta - b)^2)

  print(b_1)
}
```

```
## [1] -1.377926
## [1] -2.171626
## [1] -2.03122
## [1] -1.947703
## [1] -2.064797
```

```
errors
```

```
## [1] 0.446660145 0.053091840 0.004020336 0.005161013 0.004262119
```

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)
```

Table 1: Data summary

Name	Galton
Number of rows	928
Number of columns	2

Table 1: Data summary

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	âââââ
child	0	1	68.09	2.52	61.7	66.2	68.2	70.2	73.7	âââââ

TO-DO

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

```
avg_height= mean(c(Galton$parent, Galton$child))
avg_height
```

```
## [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?

```
n = nrow(Galton)
SST = sum((Galton$child - mean(Galton$child))^2)
rmse = sqrt(SST/(n-1))
```

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.

```
mod = lm(Galton$child~Galton$parent, Galton)
b_0 = coef(mod)[1]
b_1 = coef(mod)[2]
b_0
```

```
## (Intercept)
##    23.94153
```

```
b_1
```

```
## Galton$parent
##     0.6462906
```

```
summary(mod)$sigma #rmse
```

```
## [1] 2.238547
```

```
summary(mod)$r.squared #r squared
```

```
## [1] 0.2104629
```

Interpret all four quantities: b_0, b_1, RMSE and R^2. Use the correct units of these metrics in your answer. b_0 represents the average height is 23 b_1 is the rate at which the height of the parent grows when the childs height stays constant RMSE is the root mean squared error of the heights R^2 is .21 and represents the variance TO-DO

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

This model is soso. We can make this model better by putting in more features so that it can better predict.

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 that parents and their children have the same height because this model only uses the parents height as a feature to predict children's height.

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?

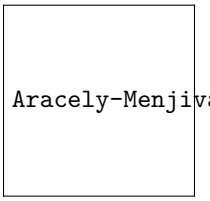
beta_0 would be 0 and beta_1 be 1.

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 = b_0, slope = b_1, color = "blue", size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = avg_height, 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 91 rows containing missing values (geom_point).
```



Fill in the following sentence:

TO-DO: 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 call it "Regression towards mediocrity in hereditary stature" because the the heights shrunk towards the mean

Why should this effect be real?

if we compare our model to the null model, out model gives a better result

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.

continuous linear representation

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 = sample(1:15, size = 30, replace = TRUE)
y = sample(1:15, size = 30, replace = TRUE)
Xy = data.frame(x = x, y = y)
mod= lm(Xy$x ~ Xy$y, data = Xy)
summary(mod)$r.squared
```

```
## [1] 0.02778465
```

```
summary(mod)$sigma
```

```
## [1] 4.573078
```

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 = sample(1:30, size = 30, replace = TRUE)
y = sample(1:30, size = 30, replace = TRUE)
Xy = data.frame(x = x, y = y)
mod= lm(Xy$x ~ Xy$y, data = Xy)
summary(mod)$r.squared
```

```
## [1] 0.0002922648
```

```
summary(mod)$sigma
```

```
## [1] 10.04604
```

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){
  if(is.numeric(X)== FALSE || is.numeric(y)== FALSE){
    return("ERROR: inputs are non-numeric")
  }
  if(nrow(X) != nrow(y)){
    return("ERROR: length of the arguments are not the same")
```

```
  }
  if ( n <= 2){
    return("The dimensions of X is not suitable.")
  }
  nrow(X)
  p= ncol(X) +1
  X = as.matrix(cbind(1, X[, 1:p]))
  colnames(X)[1]= "intercept"
  XtX = t(X)%*%X #matrix multiplication
  XtXinv= solve(XtX)
  b= XtXinv %*% t(X) %*% y #the p+1-sized column vector of OLS coefficients
  y_hat= X %*% b #vector of n predictions
  e= y - y_hat #the vector of n residuals
  df=  #for degrees of freedom of the model
  SSE= t(e) %*% e
  s_y= var(y)
  SST= (n-1) * s_y
  MSE= (1/(nrow(X)- ncol(X))) * SSE
  RMSE= sqrt(MSE)
  Rsq= 1- SSE/ SST
  model = list(b=b, y_hat=y_hat, e=e, SSE=SSE, SST=SST, MSE=MSE, RMSE=RMSE, Rsq=Rsq)
  class(model) = "my_ols"
  model
}
```

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
anova_mod= lm(Price ~ Type, cars)
print(coef(anova_mod))
```

```
## (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){
  model = my_ols(X, y)
  x_star %*% model$b
  }
```