# Actividad 3

Jhonatan Smith Garcia

23/11/2021

## Actividad 3

Se pide simular un modelo lineal de la forma $Y = X\beta + \epsilon$ dadas las sgts condiciones:

a) T es una vble numerica independiente

b) X1,X2,X3, variables numericas independientes

c) X4 una categorisca con 3 caegorias

d) $(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5)^\top = (1, 0.3, 0.6, -1, 1.5, -2)^\top$

e) $\epsilon \sim N\left(\mathbf{0}, \sigma^2 \boldsymbol{I}\right)$, donde $\sigma = 2$

f) n = 500

Ahora, con este modelo, compare modelo de tdodas las varuables vs variables individuales dados los metodos de validacion cruzada vistos en clase:

## Solucion:

Y = b0 +x$b1$+$x2$b2+x3$b3$+$x4$ b4+e

Este es el modelo que se pide ajustar.

Para ello, se generaran aleatoreamente valores de distribuciones para x1, x2, x3 (numericos) y x4 (categoricos)

```
# Valores para la simulacion pedida
n = 500
x1 = runif(n=n, min = 0, max=1)
x2 = rpois(n=n, lambda = 3)
x3 = rgamma(n=n, shape = 3,scale =2 ,)
x4 = c("Smith", "prueba1", "Se aman")
x4 = sample(x4, n, replace = T)
```

```
# Matriz del modelo

x = model.matrix(~ x1+x2+x3+x4)

# Vector de parametros
beta = c(1,0.3,0.6,-1,1.5,-2)
error = rt(n, 9)
```

```
# Variable respuesta
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
y = x %*% beta + error
```

```
# Base de datos con la simulacion realizada

data = data.frame(y,x[,2:6]) # No se toma intercepto pues interes matriz de parametros
```

Ahora, se utiliza el metodo de bootstrap para obtener un IC al 95% de $\sigma$, $R^2$, $R^2_{adj}$

```
library(boot)

sigma = function(data,i){
  modelo = lm(y~x, data=data, subset = i)
  summary(modelo)$sigma
}

prueba1 = boot(data, sigma, R=500)

boot.ci(prueba1, conf = 0.95, type = "all")
```

```
## Warning in boot.ci(prueba1, conf = 0.95, type = "all"): bootstrap variances
## needed for studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 500 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = prueba1, conf = 0.95, type = "all")
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.068,  1.235 )   ( 1.066,  1.240 )
##
## Level     Percentile            BCa
## 95%   ( 1.047,  1.221 )   ( 1.071,  1.232 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Se toma por convencion el IC normal, donde finalmente se tiene que la estiacion de un IC para sigma es (1.015, 1.170)

```
# Ic para R2
r_2 = function(data,i){
  modelo = lm(y~x, data=data, subset = i)
  summary(modelo)$r.squared
}
prueba2 = boot(data,r_2, R=500)
```

```r
boot.ci(prueba2, conf = 0.95, type = "all")
```

```
## Warning in boot.ci(prueba2, conf = 0.95, type = "all"): bootstrap variances
## needed for studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 500 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = prueba2, conf = 0.95, type = "all")
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.9061,  0.9381 )   ( 0.9067,  0.9390 )
##
## Level     Percentile            BCa
## 95%   ( 0.9049,  0.9372 )   ( 0.9044,  0.9370 )
## Calculations and Intervals on Original Scale
```

Ahora, el IC para el parametro de R^2 ajustado dado los datos de la simulacion con una confianza del 95% se encuentra entre 0.9179 y 0.9497

```r
# Ic para R2 ajustado
r_2_ajustado = function(data,i){
  modelo = lm(y~x, data=data, subset = i)
  summary(modelo)$adj.r.squared
}
bootstrap_1 = boot(data,r_2_ajustado, R=500)

boot.ci(bootstrap_1, conf = 0.95, type = "all")
```

```
## Warning in boot.ci(bootstrap_1, conf = 0.95, type = "all"): bootstrap variances
## needed for studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 500 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_1, conf = 0.95, type = "all")
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.9040,  0.9375 )   ( 0.9054,  0.9390 )
##
## Level     Percentile            BCa
## 95%   ( 0.9033,  0.9369 )   ( 0.8982,  0.9355 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Finalmente, un IC al 95% de confianza para el $R^2_{adj}$ es (0.9168, 0.9488)

Ahora, se pide errores con distibucion t student con 9 grados de libertad.

```r
v = 9
var_esp = v/(v-2)
var_esp
```

```
## [1] 1.285714
```

Está será la varianza esperada para los errores.

Simulando el modelo anterior, se tiene que:

$\beta_{1,3} = 2$

```
set.seed(1039705595)
s = 2

# Valores para la simulacion pedida
n = 500
x1 = runif(n=n, min = 0, max=1)
x2 = rpois(n=n, lambda = 3)
x3 = rgamma(n=n, shape = 3,scale =2 ,)
x4 = c("A", "B", "C")
x4 = sample(x4, n, replace = T)

x = model.matrix(~x1+I(x1^3)+x2+x3+x4) # Asi lo pide el ejercicio
```

```
beta = c(1,0.3,2,0.6,-1,1.5,-2)
error = rnorm(n,0,s)

y = x %*% beta + error
```

```
# Base de datos simulacion 2
data2 = data.frame(y, x[,2:7])
```

Utilizando esta base de datos se separará datos de prueba y de entrenamiento

```
prop = 0.7
ni = nrow(data2)
sample1 = sample(1:n, size = prop*ni)
```

```
Train = data2[sample1,] # Datos de entrenamiento

Test = data2[-sample1,] # Datps de prueba
```

```
y_train = Train$y
y_test = Test$y
```

```
# Modelo con x1
mod1 = lm(y~., data = Train)
mse1 = mean((y_test-predict(mod1,Test))^2)

# Modelo con x1^2
mod2 = lm(y~x1+I(x1^2)+x2+x3+x4B+x4C, data = Train)
mse2 = mean((y_test-predict(mod2,Test))^2)

# Modelo con x1^3
mod3 = lm(y~x1+I(x1^3)+x2+x3+x4B+x4C, data = Train)
mse3 = mean((y_test-predict(mod3,Test))^2)

# Modelo con x1^4
mod4 = lm(y~x1+I(x1^4)+x2+x3+x4B+x4C, data = Train)
mse4 = mean((y_test-predict(mod4,Test))^2)

# Modelo con x1^5
```

```r
mod5 = lm(y~x1+I(x1^5)+x2+x3+x4B+x4C, data = Train)
mse5 = mean((y_test-predict(mod5,Test))^2)

# Modelo con x1^6
mod6 = lm(y~x1+I(x1^6)+x2+x3+x4B+x4C, data = Train)
mse6 = mean((y_test-predict(mod6,Test))^2)

# Modelo con x1^7
mod7 = lm(y~x1+I(x1^7)+x2+x3+x4B+x4C, data = Train)
mse7 = mean((y_test-predict(mod7,Test))^2)

# Modelo con x1^8
mod8 = lm(y~x1+I(x1^8)+x2+x3+x4B+x4C, data = Train)
mse8 = mean((y_test-predict(mod8,Test))^2)

# Modelo con x1^9
mod9 = lm(y~x1+I(x1^9)+x2+x3+x4B+x4C, data = Train)
mse9 = mean((y_test-predict(mod9,Test))^2)

# Modelo con x1^10
mod10 = lm(y~x1+I(x1^10)+x2+x3+x4B+x4C, data = Train)
mse10 = mean((y_test-predict(mod10,Test))^2)
```

```r
tabla = cbind(c(mse1,mse2,mse3,mse4,mse5,mse6,mse7,mse8,mse9,mse10))
colnames(tabla) = c("MSE")
rownames(tabla) = c("Grado 1", "Grado 2", "Grado 3", "Grado 4", "Grado 5", "Grado 6", "Grado 7", "Grado
```

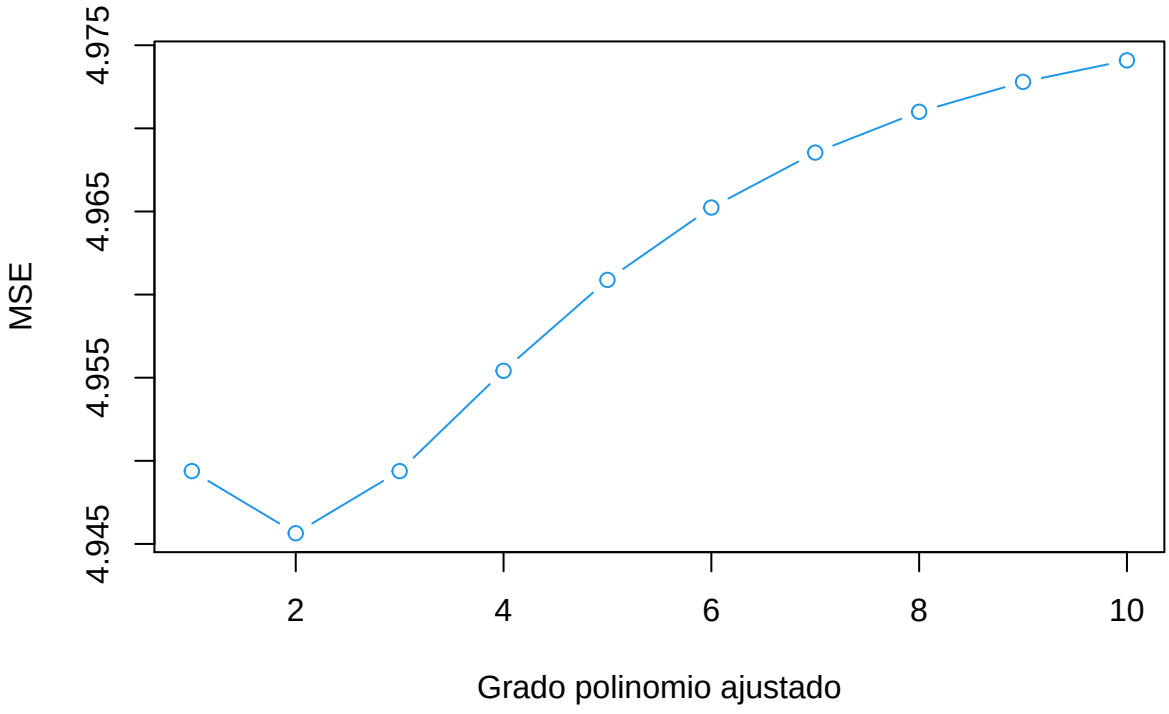
```r
tabla
```

```
##                MSE
## Grado 1  4.949381
## Grado 2  4.945643
## Grado 3  4.949381
## Grado 4  4.955415
## Grado 5  4.960883
## Grado 6  4.965237
## Grado 7  4.968543
## Grado 8  4.970998
## Grado 9  4.972795
## Grado 10 4.974091
```

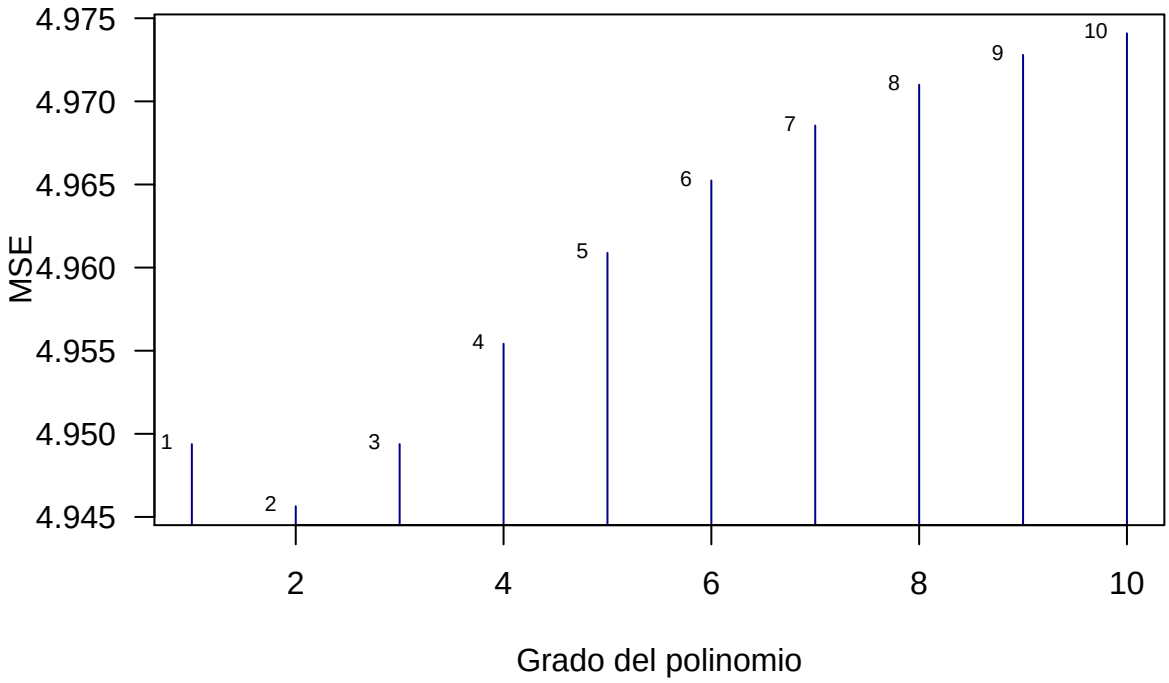
De la anterior tabla,se obtienen los valores de los respectivos MSE para cada polinomio ajustado segun el grado. Se desea el modelo de menor grado posible a menor MSE; teniendo en cuenta siempre el principio de parsimonia.

ASi, se selecciona el modelo de grado 1, pues su medida de MSE es la menor y es el modelo con el grado mas bajo.

```r
plot(1:10, tabla, xlab = "Grado polinomio ajustado", ylab = "MSE", type = "b", col=4)
```




```
plot (c(mse1,mse2,mse3,mse4,mse5, mse6,mse7,mse8,mse9,mse10), xlab = "Grado del polinomio", ylab = "MSE
, pch=19, col = "darkblue", type = "h")
labels = c("1","2","3","4","5","6","7","8","9","10")
text(c(mse1,mse2,mse3,mse4,mse5, mse6,mse7,mse8,mse9,mse10), labels,cex = 0.7, pos=2)
```




De la misma manera, de forma grafica se observa igualmente que el mejor polinomio es el de grado 2.