

# Árboles de regresión: K regresores

## Contents

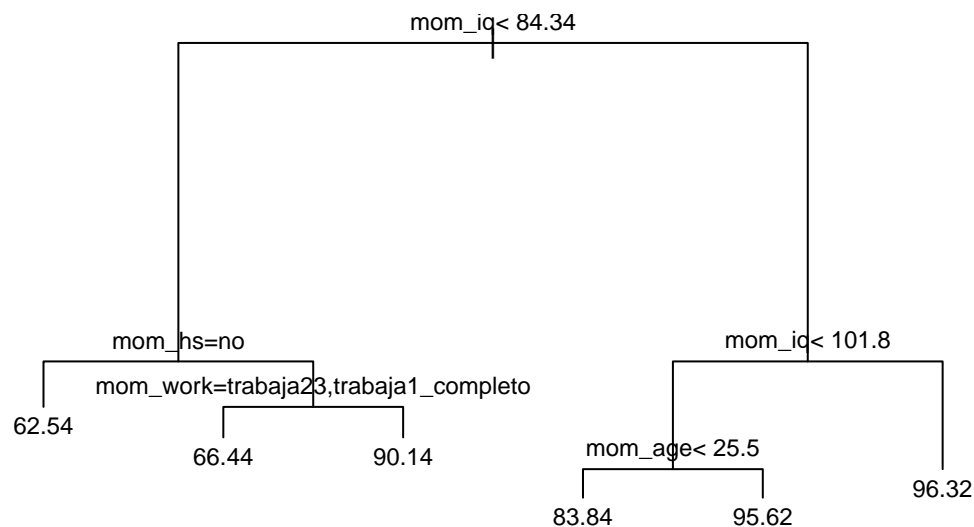
1	Arbol con regresores cuantitativos y cualitativos	1
2	Parámetros del árbol	2
3	Residuos	4
4	Podado	6
5	Prediccion	9

## 1 Arbol con regresores cuantitativos y cualitativos

```
library(rpart)
d = read.csv('datos/kidiq.csv')
d$mom_hs = factor(d$mom_hs, labels = c("no", "si"))
d$mom_work = factor(d$mom_work, labels = c("notrabaja", "trabaja23", "trabaja1_parcial", "trabaja1_completo"))

# method = "anova" para modelos de regresion
t1 = rpart(kid_score ~ ., data = d, method = "anova")

plot(t1, margin = 0.02)
text(t1, cex = 0.75, pretty = 0)
```



```
print(t1)

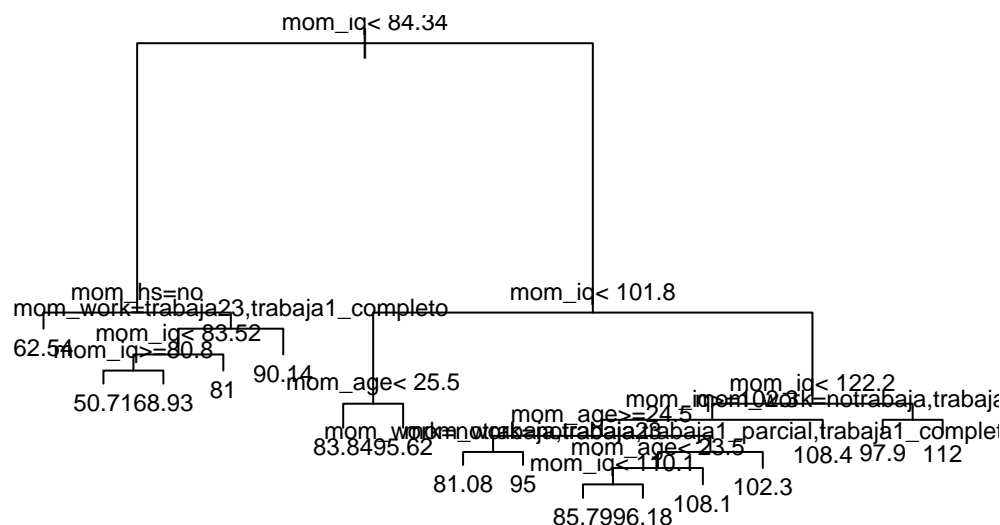
## n= 434
##
## node), split, n, deviance, yval
```

```
##      * denotes terminal node
##
## 1) root 434 180386.200 86.79724
##    2) mom_iq< 84.33641 69 29011.830 66.86957
##      4) mom_hs=no 35 11166.690 62.54286 *
##      5) mom_hs=si 34 16515.440 71.32353
##        10) mom_work=trabaja23,trabaja1_completo 27 9750.667 66.44444 *
##        11) mom_work=notrabaja,trabaja1_parcial 7 3642.857 90.14286 *
##    3) mom_iq>=84.33641 365 118793.700 90.56438
##      6) mom_iq< 101.8061 191 58829.520 85.31937
##        12) mom_age< 25.5 167 51906.630 83.83832 *
##        13) mom_age>=25.5 24 4007.625 95.62500 *
##      7) mom_iq>=101.8061 174 48941.980 96.32184 *
```

Como vemos, cuando hay una variables cualitativa, se van asignando los diferentes niveles del factor a cada rama que sale del nodo hasta que se encuentra la asignación con menor RSS: - En el nodo 2, si mom\_hs = no, nos vamos hacia la izquierda; si mom\_hs = si, nos vamos a la derecha. - En el nodo 5, si mom\_work = trabaja23,trabaja1\_completo nos vamos a la izquierda; si mom\_work=notrabaja,trabaja1\_parcial nos vamos a la derecha.

## 2 Parámetros del árbol

```
t2 = rpart(kid_score ~ ., data = d, method = "anova",
           control = rpart.control(minsplit = 10, minbucket = 5, cp = 0.007))
plot(t2, margin = 0.02)
text(t2, cex=.75, pretty = 0)
```



```
print(t2)

## n= 434
##
## node), split, n, deviance, yval
##      * denotes terminal node
##
## 1) root 434 180386.2000 86.79724
##    2) mom_iq< 84.33641 69 29011.8300 66.86957
##      4) mom_hs=no 35 11166.6900 62.54286 *
```

```
##      5) mom_hs=si 34 16515.4400 71.32353
##      10) mom_work=trabaja23,trabaja1_completo 27 9750.6670 66.44444
##      20) mom_iq< 83.51527 22 8034.5910 63.13636
##      40) mom_iq>=80.79538 7 545.4286 50.71429 *
##      41) mom_iq< 80.79538 15 5904.9330 68.93333 *
##      21) mom_iq>=83.51527 5 416.0000 81.00000 *
##      11) mom_work=notrabaja,trabaja1_parcial 7 3642.8570 90.14286 *
##      3) mom_iq>=84.33641 365 118793.7000 90.56438
##      6) mom_iq< 101.8061 191 58829.5200 85.31937
##      12) mom_age< 25.5 167 51906.6300 83.83832 *
##      13) mom_age>=25.5 24 4007.6250 95.62500 *
##      7) mom_iq>=101.8061 174 48941.9800 96.32184
##      14) mom_iq< 122.2355 132 38970.8100 94.53788
##      28) mom_iq>=102.2658 127 36978.9900 93.99213
##      56) mom_age>=24.5 37 11611.2400 90.48649
##      112) mom_work=notrabaja,trabaja23 12 6398.9170 81.08333 *
##      113) mom_work=trabaja1_parcial,trabaja1_completo 25 3642.0000 95.00000 *
##      57) mom_age< 24.5 90 24726.1000 95.43333
##      114) mom_work=notrabaja,trabaja1_parcial,trabaja1_completo 69 19845.6500 93.34783
##      228) mom_age< 23.5 61 17308.7500 91.40984
##      456) mom_iq< 110.0551 28 7852.7140 85.78571 *
##      457) mom_iq>=110.0551 33 7818.9090 96.18182 *
##      229) mom_age>=23.5 8 560.8750 108.12500 *
##      115) mom_work=trabaja23 21 3594.2860 102.28570 *
##      29) mom_iq< 102.2658 5 993.2000 108.40000 *
##      15) mom_iq>=122.2355 42 8230.7860 101.92860
##      30) mom_work=notrabaja,trabaja1_completo 30 5234.7000 97.90000 *
##      31) mom_work=trabaja23,trabaja1_parcial 12 1292.0000 112.00000 *
```

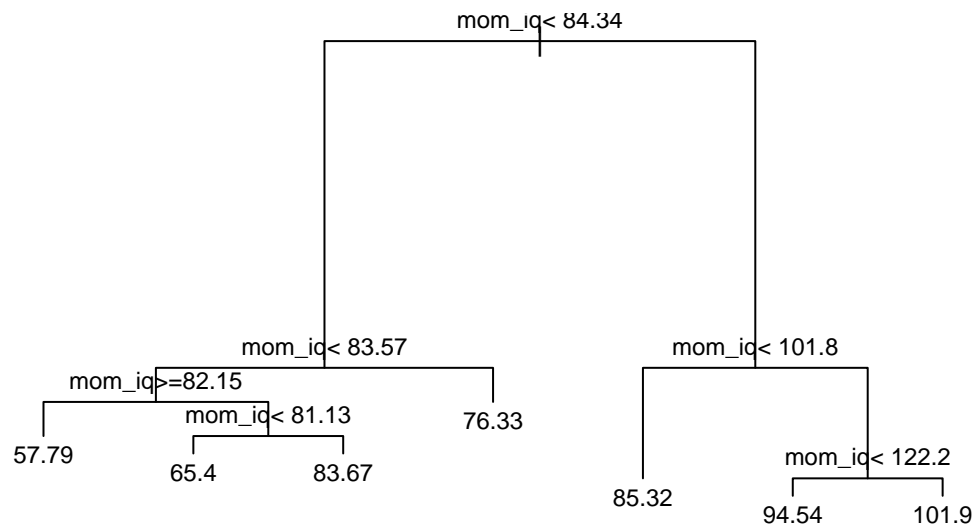
- Como vemos, en este caso el criterio que detiene el crecimiento del árbol es cp. Por ejemplo, el nodo 3 se ha dividido ya que

```
(118793.70 - 58829.52 - 48941.98)/180386.20
```

```
## [1] 0.06110334
```

- que es mayor que el límite  $cp = 0.05$ .
- Podemos construir un árbol más *profundo*:

```
t3 = rpart(kid_score ~ mom_iq, data = d, method = "anova",
  control = rpart.control(minsplit = 10, minbucket = 5, cp = 0.0069))
plot(t3, margin = 0.02)
text(t3, cex=.75)
```



```
print(t3)
```

```
## n= 434
##
## node), split, n, deviance, yval
##      * denotes terminal node
##
## 1) root 434 180386.200  86.79724
##    2) mom_iq< 84.33641 69  29011.830  66.86957
##      4) mom_iq< 83.57478 60  26044.850  65.45000
##        8) mom_iq>=82.1513 14   5136.357  57.78571 *
##        9) mom_iq< 82.1513 46  19835.830  67.78261
##          18) mom_iq< 81.13004 40  14465.600  65.40000 *
##          19) mom_iq>=81.13004 6   3629.333  83.66667 *
##      5) mom_iq>=83.57478 9   2040.000  76.33333 *
##    3) mom_iq>=84.33641 365 118793.700  90.56438
##      6) mom_iq< 101.8061 191  58829.520  85.31937 *
##      7) mom_iq>=101.8061 174  48941.980  96.32184
##        14) mom_iq< 122.2355 132  38970.810  94.53788 *
##        15) mom_iq>=122.2355 42   8230.786 101.92860 *
```

- vemos que el nodo 7 en t2 no se dividía pero en t3 si se divide ya que:

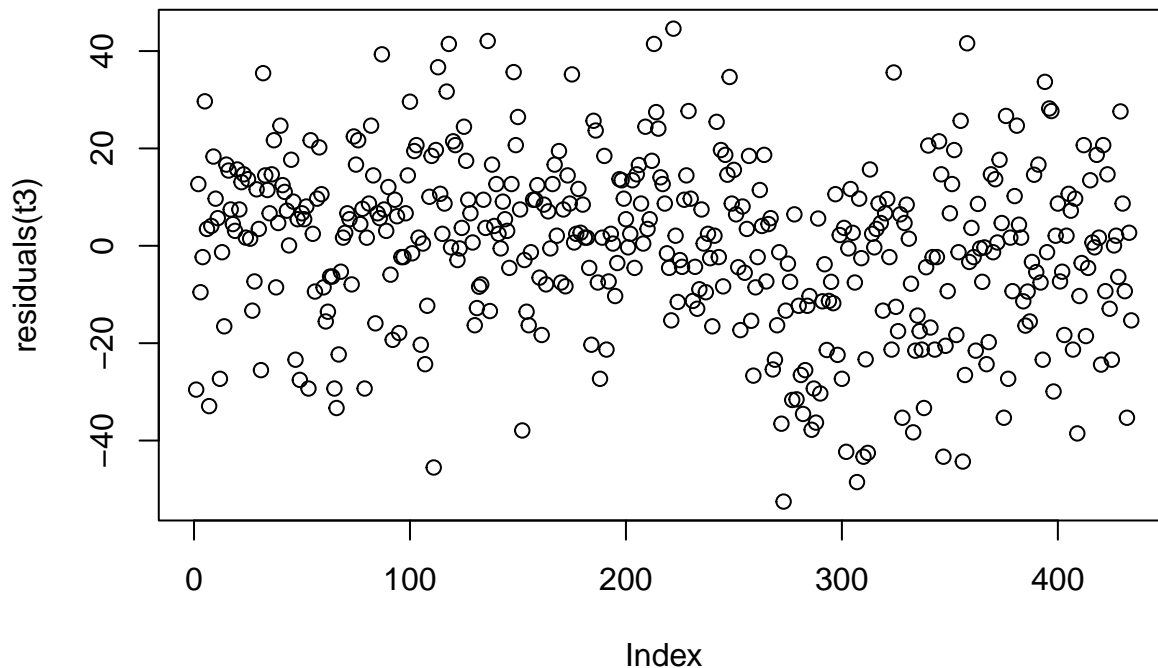
```
(48941.980 - 38970.810 - 8230.786)/180386.200
```

```
## [1] 0.009648099
```

- De nuevo cp es el parámetro más restrictivo.

### 3 Residuos

```
plot(residuals(t3))
```



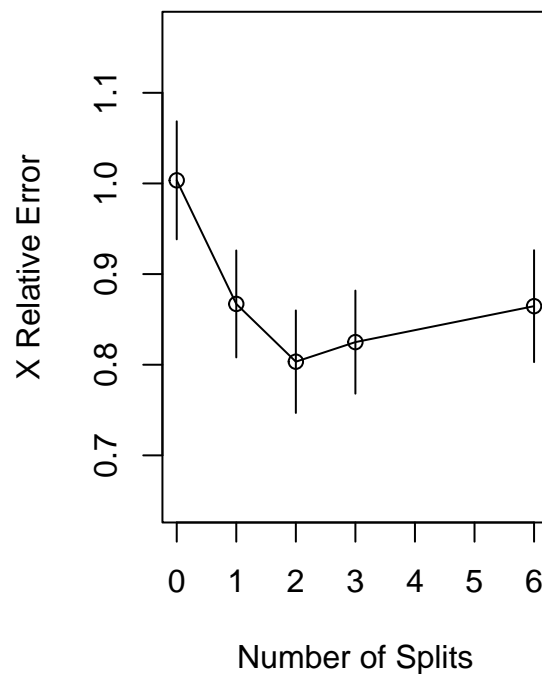
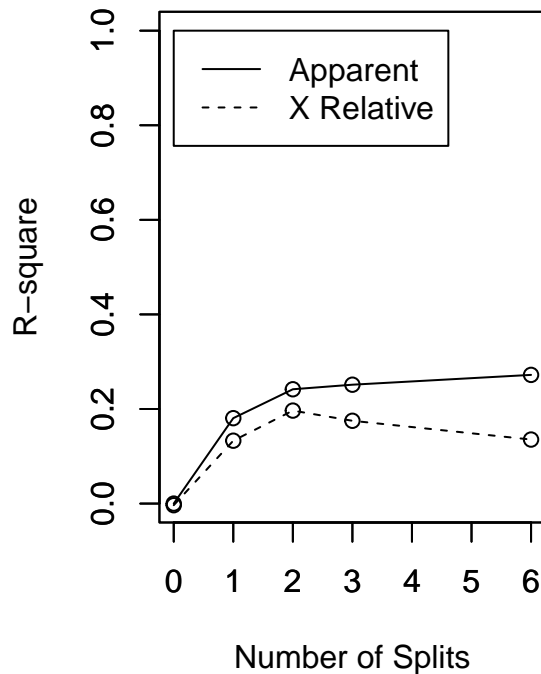
- El  $R^2$  se define a manera análoga a regresión

$$R^2 = 1 - \frac{RSS}{TSS}$$

- donde hay que recordar de  $RSS = \text{deviance}(\text{nodo})$  y  $TSS = \text{deviance}(\text{root})$
- Se denomina error relativo al cociente  $RSS/TSS$ . Y la X indica que se ha calculado mediante validación cruzada.

```
par(mfrow = c(1,2))
rsq.rpart(t3)
```

```
##
## Regression tree:
## rpart(formula = kid_score ~ mom_iq, data = d, method = "anova",
##       control = rpart.control(minsplit = 10, minbucket = 5, cp = 0.0069))
##
## Variables actually used in tree construction:
## [1] mom_iq
##
## Root node error: 180386/434 = 415.64
##
## n= 434
##
##      CP nsplit rel error  xerror   xstd
## 1 0.1806158      0  1.00000 1.00343 0.065097
## 2 0.0611036      1  0.81938 0.86707 0.059011
## 3 0.0096481      2  0.75828 0.80337 0.056542
## 4 0.0069121      3  0.74863 0.82494 0.056862
## 5 0.0069000      6  0.72790 0.86460 0.061723
```

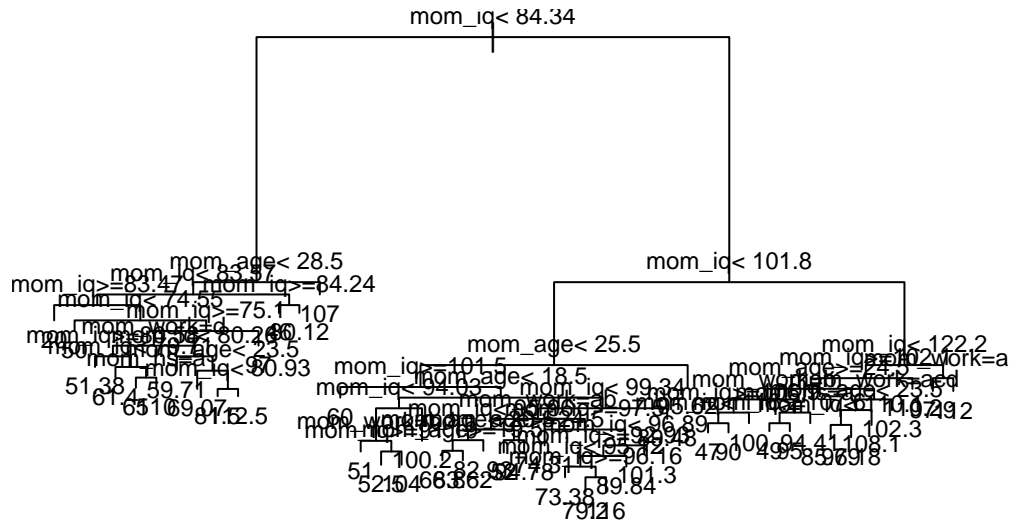


- Apparent:  $R^2$  calculado con la formula  $(1 - \text{RSS}/\text{TSS})$
- X Relative:  $R^2$  calculado con validación cruzada (como vemos, el  $R^2$  cuadrado con validación cruzada es menor que el apparent ya que uno está calculado en los datos train y otro en los datos test).
- X relative error:  $1 - \text{X Relative}$ , es decir,  $\text{RSS}/\text{TSS}$ . Está calculado con validación cruzada. Se dibuja el intervalo  $\pm \text{SE}$  calculado con validación cruzada.

## 4 Podado

- Los árboles que hemos visto se construyen de arriba hacia abajo, desde el nodo raíz hasta las hojas. Otra estrategia es construir un árbol muy profundo y luego podarlo. Construiríamos el árbol, por tanto, de abajo hacia arriba.
- Primero construimos un árbol profundo:

```
t4 = rpart(kid_score ~ ., data = d, method = "anova",
           control = rpart.control(minsplit = 2, cp = 0.005))
plot(t4, margin = 0.02)
text(t4, cex=.75)
```



- Utilizando validación cruzada (el numero de validaciones viene dado por el parámetro xval), se determina el arbol con un determinado numero de hojas que tenga el mayor R2, o de manera equivalente, el menor error relativo.

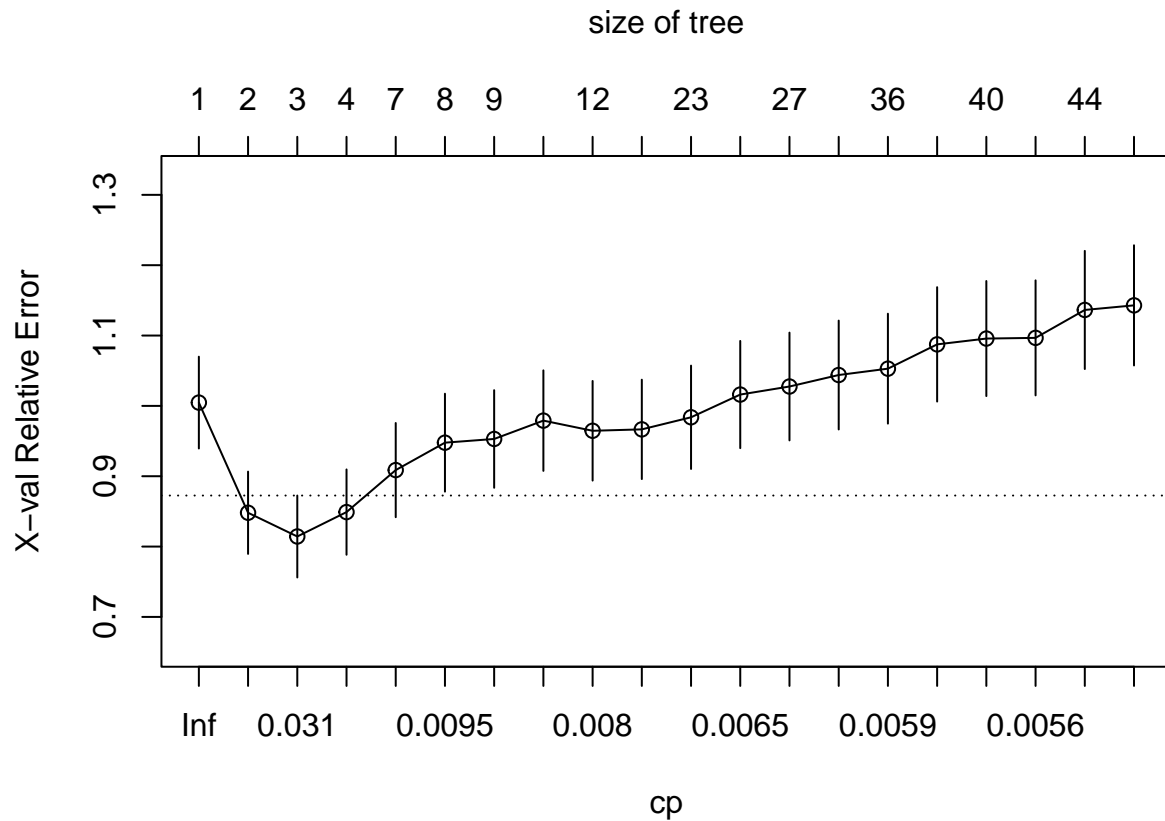
```
t4_printcp = printcp(t4) # lo guardamos en una variable para utilizarlo despues
```

```
##
## Regression tree:
## rpart(formula = kid_score ~ ., data = d, method = "anova", control = rpart.control(minsplit = 2,
##   cp = 0.005))
##
## Variables actually used in tree construction:
## [1] mom_age mom_hs mom_iq mom_work
##
## Root node error: 180386/434 = 415.64
##
## n= 434
##
##      CP nsplit rel error  xerror   xstd
## 1  0.1806158     0  1.00000  1.00469  0.065270
## 2  0.0611036     1  0.81938  0.84807  0.058498
## 3  0.0161612     2  0.75828  0.81432  0.058252
## 4  0.0123630     3  0.74212  0.84902  0.060644
## 5  0.0096481     6  0.70503  0.90871  0.066971
## 6  0.0094469     7  0.69538  0.94768  0.069611
## 7  0.0090591     8  0.68594  0.95296  0.069383
## 8  0.0085517     9  0.67688  0.97906  0.071466
## 9  0.0075110    11  0.65977  0.96461  0.070811
## 10 0.0073281    16  0.61596  0.96662  0.070646
## 11 0.0065568    22  0.57025  0.98381  0.073468
## 12 0.0065249    24  0.55713  1.01610  0.076222
## 13 0.0061137    26  0.54408  1.02755  0.076626
## 14 0.0060254    31  0.51109  1.04387  0.077404
## 15 0.0058459    35  0.48699  1.05291  0.078138
## 16 0.0057384    38  0.46945  1.08743  0.081324
## 17 0.0057263    39  0.46371  1.09572  0.081794
## 18 0.0055520    41  0.45226  1.09668  0.081778
## 19 0.0053275    43  0.44116  1.13638  0.083988
```

```
## 20 0.0050000    44  0.43583 1.14290 0.085454
```

- También se puede utilizar `plotcp()`:

```
plotcp(t4)
```



- A veces este gráfico tiene un mínimo, por lo que deberíamos seleccionar ese árbol. En caso contrario, elegimos el tamaño donde el error se estabilice.
- Según el gráfico y la tabla anterior, un árbol de 3 hojas parece razonable.

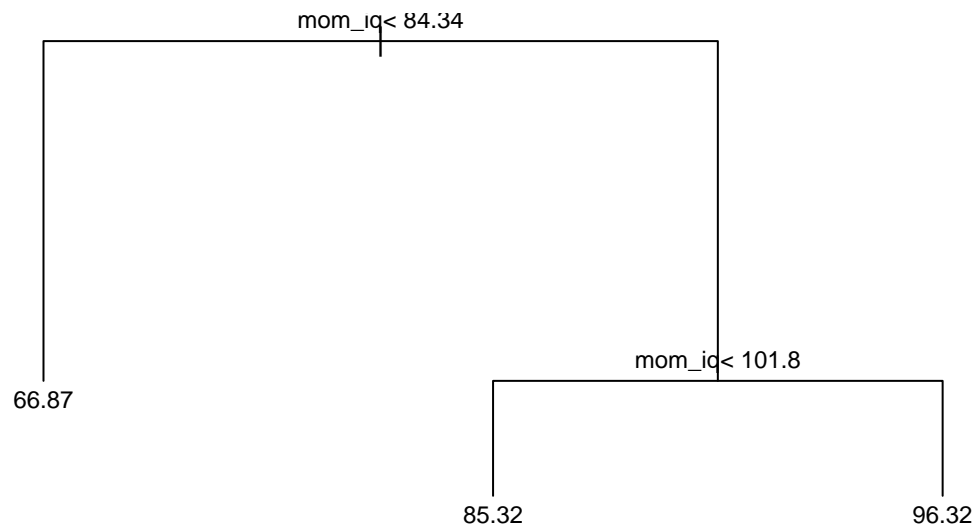
```
(t4_cp = t4_printcp[3,"CP"])
```

```
## [1] 0.01616121
```

- Ahora podemos el árbol:

```
t4_prune = prune(t4, cp = t4_cp)
plot(t4_prune, margin = 0.02)
text(t4_prune, cex=.75)
```





Ojo, estamos seleccionando el arbol con mayor R2 de acuerdo a validación cruzada (variable xerror). Si nos fijamos en el árbol con menor error de acuerdo a la variable *rel error* tendríamos que elegir el árbol de 45 hojas!

## 5 Prediccion

```

xp = data.frame(mom_iq = 95, mom_age = 30, mom_hs = "si",
                mom_work = "notrabaja")
predict(t4_prune, newdata = xp)

```

```

##          1
## 85.31937

```

- Mirando el arbol se puede verificar fácilmente la predicción.