Árboles de regresión: 1 regresor

Contents

1 Introducción

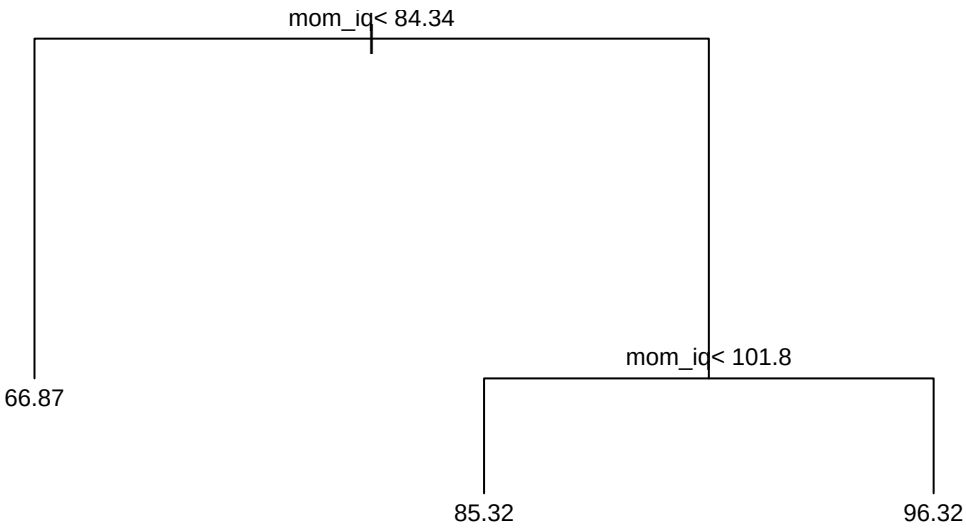
2 Arbol con un regresor cuantitativo

2.1 Construcción del árbol

```
library(rpart)
d = read.csv('datos/kidiq.csv')
```

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

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




```
print(t1)
```

```
## n= 434
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 434 180386.20 86.79724
##   2) mom_iq< 84.33641 69  29011.83 66.86957 *
##   3) mom_iq>=84.33641 365 118793.70 90.56438
##     6) mom_iq< 101.8061 191  58829.52 85.31937 *
##     7) mom_iq>=101.8061 174  48941.98 96.32184 *
```

Lo que devuelve la tabla es:

- Numero del nodo
- split: criterio para hacer la partición del nodo
- n: numero de datos que componen el nodo.
- deviance: RSS $= \sum(y_i - \hat{y}_i)^2$
- yval: predicción del nodo $= \bar{y}$
- un asterisco * para indicar un nodo terminal u hoja.

Por ejemplo, para el nodo raiz:

- Predicción:

```
(yp_root = mean(d$kid_score))
```

```
## [1] 86.79724
```

- Deviance:

```
( deviance_root = sum( (d$kid_score - yp_root)^2 ) )
```

```
## [1] 180386.2
```

Para el nodo 3:

- Datos que pertenecen al nodo 3:

```
d3 = d[d$mom_iq >= 84.33641,]
```

- Predicción:

```
(yp_3 = mean(d3$kid_score))
```

```
## [1] 90.56438
```

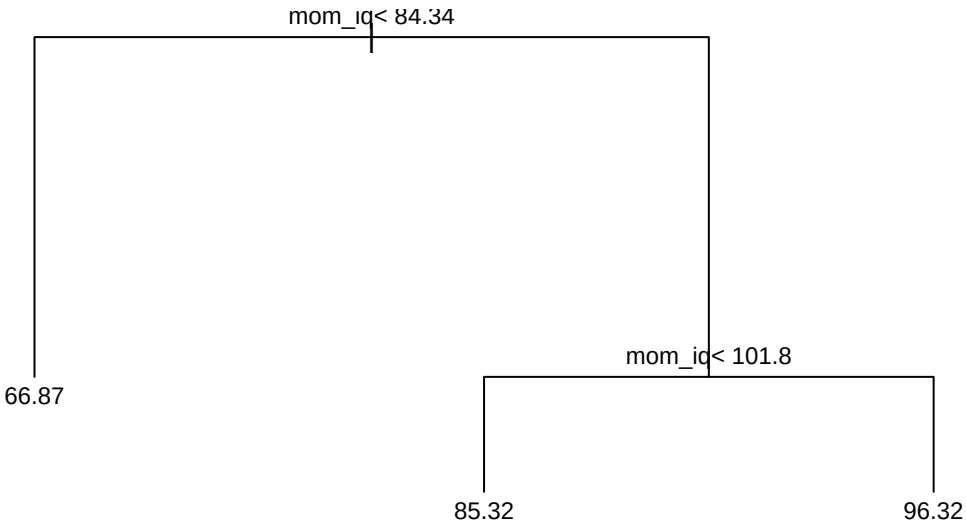
- Deviance:

```
( deviance_3 = sum( (d3$kid_score - yp_3)^2 ) )
```

```
## [1] 118793.7
```

2.2 Parámetros del árbol

```
t2 = rpart(kid_score ~ mom_iq, data = d, method = "anova",
    control = rpart.control(minsplit = 10, minbucket = 5, cp = 0.05))
plot(t2, margin = 0.02)
text(t2, cex=.75)
```



control:

- minsplit: número mínimo de observaciones del nodo para que se divida en dos (por defecto, minsplit = 20).
- minbucket: número mínimo de observaciones en un nodo terminal u hoja (por defecto, minbucket = minsplit/3).
- maxdepth: se fija el nivel máximo de cualquier nodo del árbol, siendo 0 el nivel del nodo raiz.
- cp: complexity parameter. En árboles de regresión, para que un nodo se divida, el R2 tiene que incrementarse en más de cp (por defecto, cp = 0.01). En este caso: [RSS(padre) - RSS(hijo1) - RSS(hijo2)]/RSS(raiz) > cp
- xval: número de validaciones cruzadas. Se utiliza para el podado.

```
print(t2)
```

```
## n= 434
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 434 180386.20 86.79724
##   2) mom_iq< 84.33641 69  29011.83 66.86957 *
##   3) mom_iq>=84.33641 365 118793.70 90.56438
##     6) mom_iq< 101.8061 191  58829.52 85.31937 *
##     7) mom_iq>=101.8061 174  48941.98 96.32184 *
```

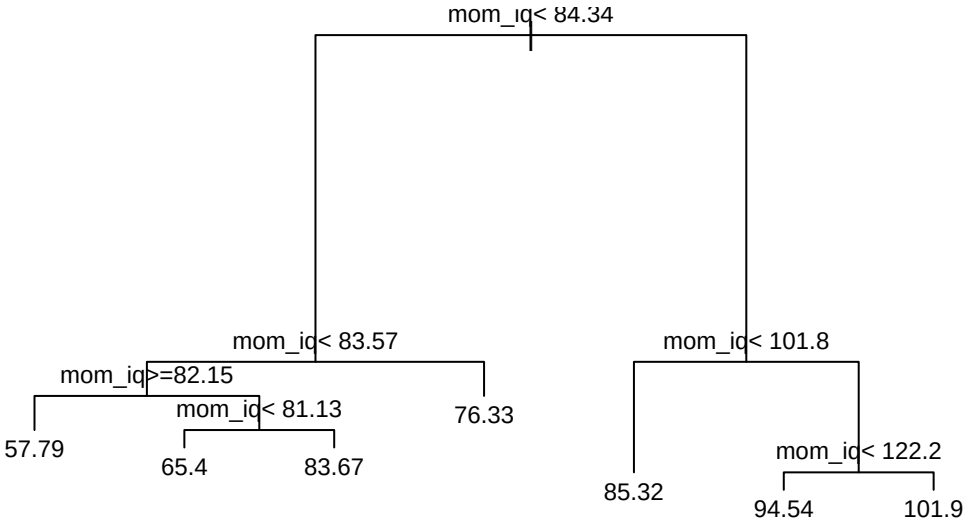
- Como vemos, en este caso el criterio que detiene el crecimiento del árbol es cp. Por ejemplo, el nodo 3 se ha dividido ya que

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

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

- que es mayor que el límite cp = 0.05.
- Podemos construir un arbol 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:

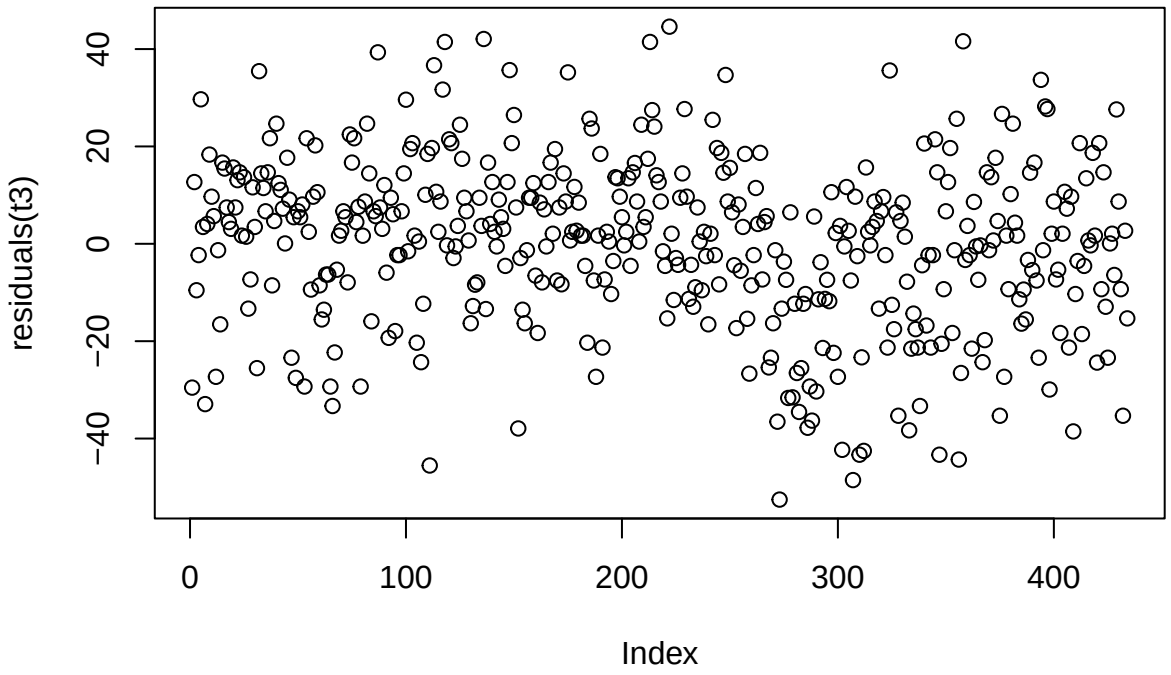
```
(48941.980 - 38970.810 - 8230.786)/180386.200
```

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

- De nuevo cp es el parámetro más restrictivo.

2.3 Residuos

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



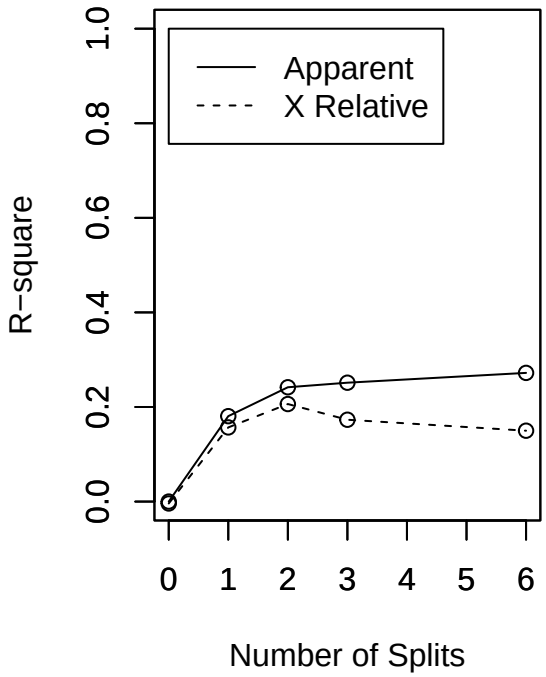
- El R2 se define a manera análoga a regresión

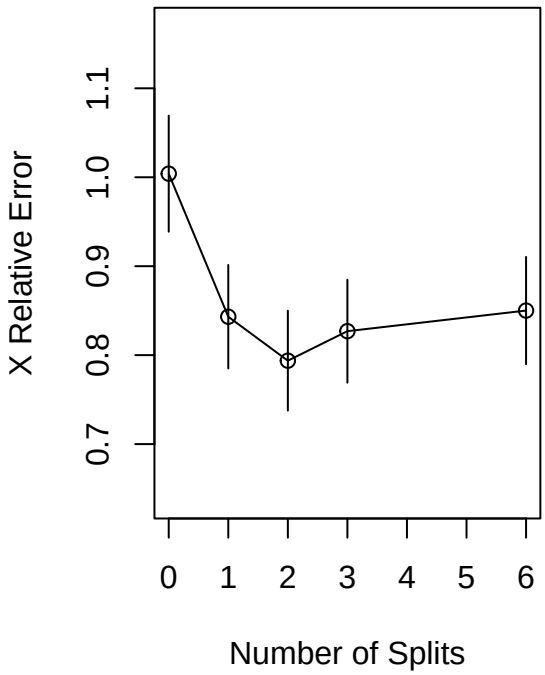
$$R^2 = 1 - \frac{RSS}{TSS}$$

- donde RSS $= \sum e_i^2 = \sum deviance(hoja_i)$ y TSS = deviance(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 1.00403 0.065259
## 2 0.0611036      1   0.81938 0.84320 0.058206
## 3 0.0096481      2   0.75828 0.79375 0.056141
## 4 0.0069121      3   0.74863 0.82694 0.057910
## 5 0.0069000      6   0.72790 0.85012 0.060295
```



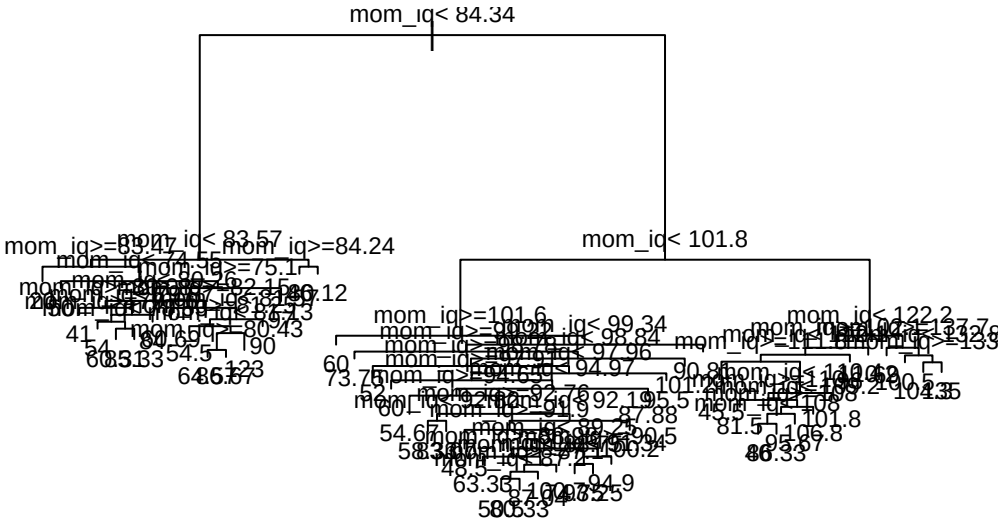


- Appatent: R2 calculado con la formula (1 - RSS/TSS)
- X Relative: R2 calculado con validación cruzada (como vemos, el R2 cuadrado con validación cruzada es menor que el apparent ya que uno esta calculado en los datos train y otro en los datos test).
- X relative error: 1 - X Relative, es decir, RSS/TSS. Está calculado con validación cruzada. Se dibuja el intervalo +/- SE calculado con validación cruzada.

3 Podado

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

```
t4 = rpart(kid_score ~ mom_iq, 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.

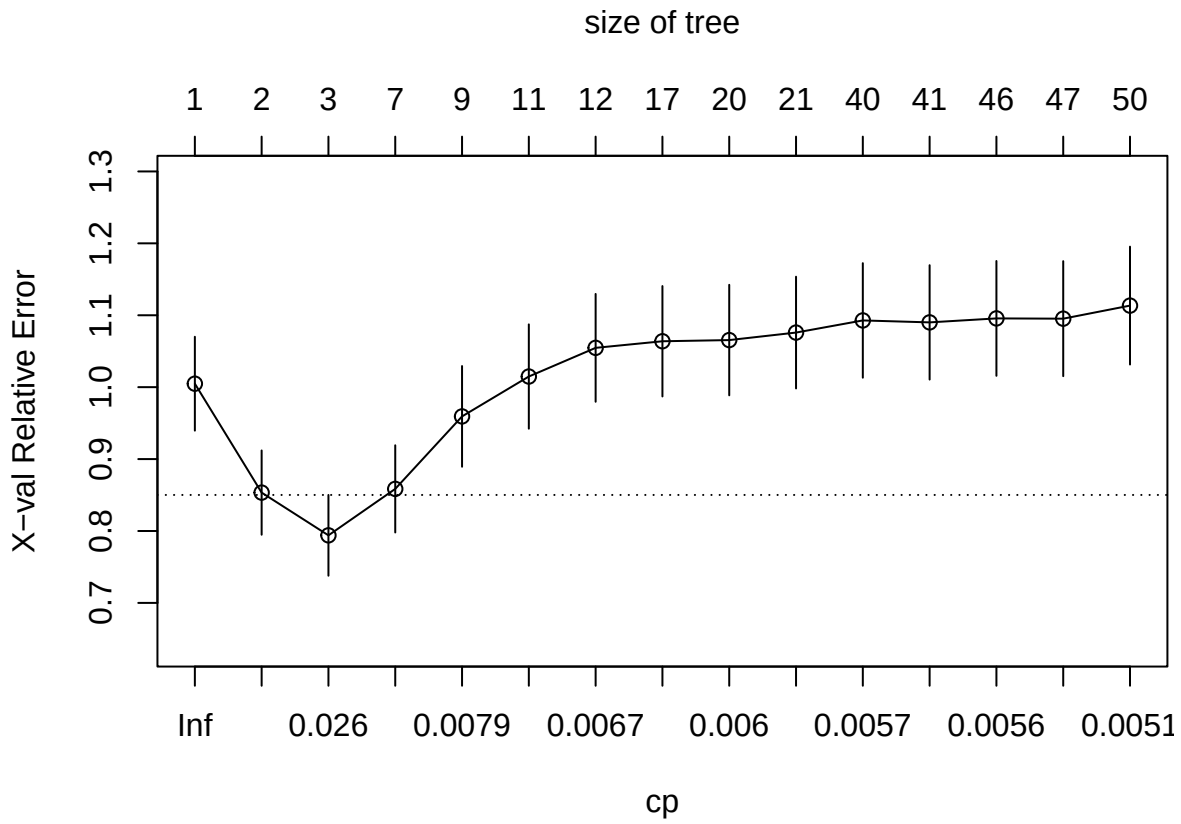
```
t4_printcp = printcp(t4) # lo guardamos en una variable para utilizarlo despues
```

```
##
## Regression tree:
## rpart(formula = kid_score ~ mom_iq, data = d, method = "anova",
##     control = rpart.control(minsplit = 2, cp = 0.005))
##
## 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 1.00482 0.065279
## 2  0.0611036      1   0.81938 0.85340 0.058618
## 3  0.0106614      2   0.75828 0.79395 0.056156
## 4  0.0083922      6   0.71563 0.85854 0.060647
## 5  0.0074483      8   0.69885 0.95935 0.070017
## 6  0.0071830     10   0.68395 1.01480 0.072563
## 7  0.0062379     11   0.67677 1.05467 0.074986
## 8  0.0062370     16   0.64239 1.06383 0.076750
## 9  0.0057384     19   0.62368 1.06548 0.076854
## 10 0.0057042     20   0.61794 1.07591 0.077703
## 11 0.0056835     39   0.49145 1.09280 0.079662
## 12 0.0056244     40   0.48577 1.09009 0.079539
## 13 0.0055191     45   0.45765 1.09562 0.079850
## 14 0.0052498     46   0.45213 1.09526 0.079982
## 15 0.0050000     49   0.43638 1.11343 0.081977
```

- Salida de printcp():
  - col3: numero de divisiones del arbol obtenido. Numero de hojas = numero divisiones + 1
  - col2: el valor de cp que hay que utilizar para obtener ese árbol

  - col3: error relativo del arbol podado, RSS/TSS
  - col4: error relativo calculado con validación cruzada.
  - col5: desviación típica de xerror
- También se puede utilizar plotcp():

```
plotcp(t4)
```



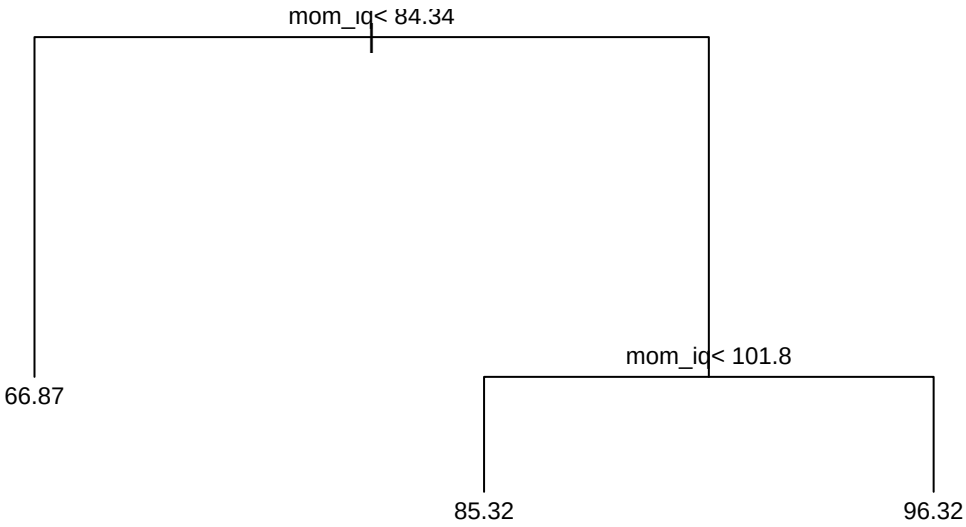
- A veces este gráfico tiene un mínimo, por lo que deberíamos seleccionar ese arbol. En caso contrario, elegimos el tamaño donde el error se estabilice.
- Según el gráfico y la tabla anterior, un arbol de 3 hojas (nsplit =2) parece razonable.

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

```
## [1] 0.01066143
```

- Ahora podamos el arbol:

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



En este caso se ha obtenido la misma solución que construyendo el árbol con los parámetros por defecto.

4 Prediccion

```
xp = data.frame(mom_iq = 95)
predict(t4_prune, newdata = xp)
```

```
##        1
## 85.31937
```

- Mirando el arbol se puede verificar fácilmente la predicción.