# Lec5: Árboles de Decisión

Isidoro Garcia Urquieta

2023

# Agenda

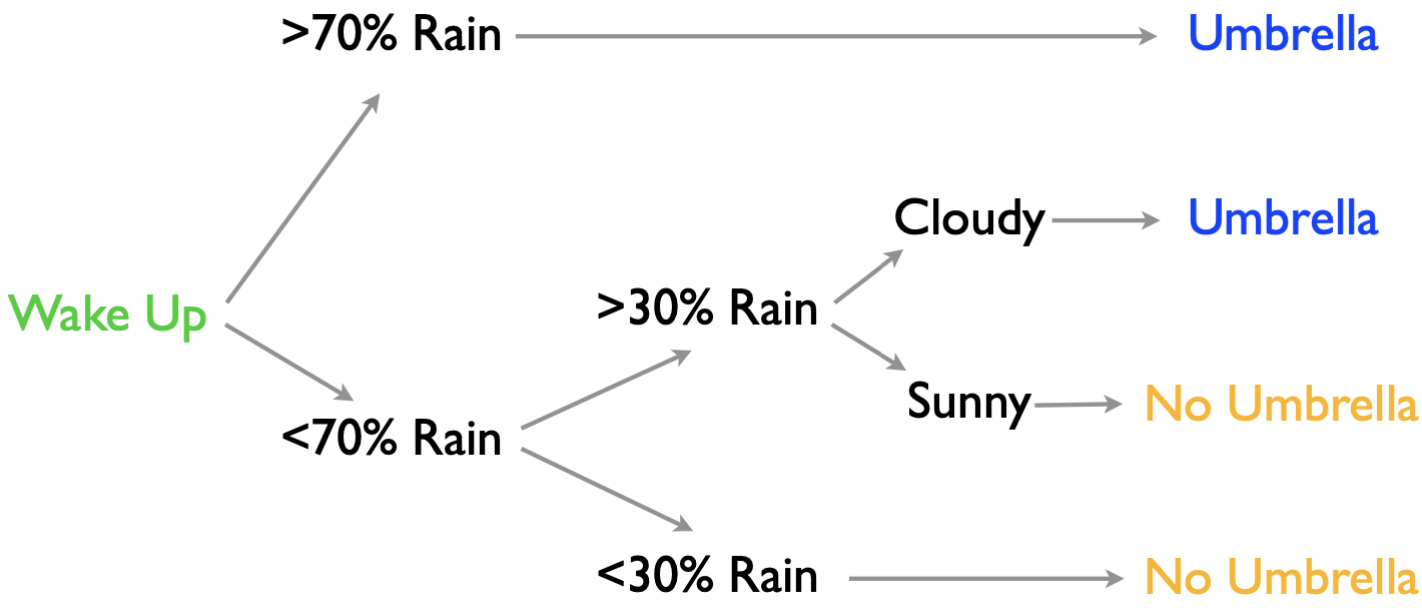
- Árboles de Clasificación
- Árboles de Regresión
- Recursive binary splitting: Decision split for each case
- Tree Pruning
- Linear vs Trees
- Boostrapping & Bagging

# Qué es un arbol de decisión

Los árboles de decisión son sistemas lógicos que mapean características $X_i$ a predicciones $Y_i$.

- Esto es una secuencia de decisiones binarias para llegar a una conclusión.
- Son jerarquicos. Siempre empiezan por el mismo lado.
- La predicción final es un nodo terminal

# Ejemplo


>70% Rain
Umbrella
Cloudy
Umbrella
Wake Up
>30% Rain
Sunny
No Umbrella
<70% Rain
<30% Rain
No Umbrella

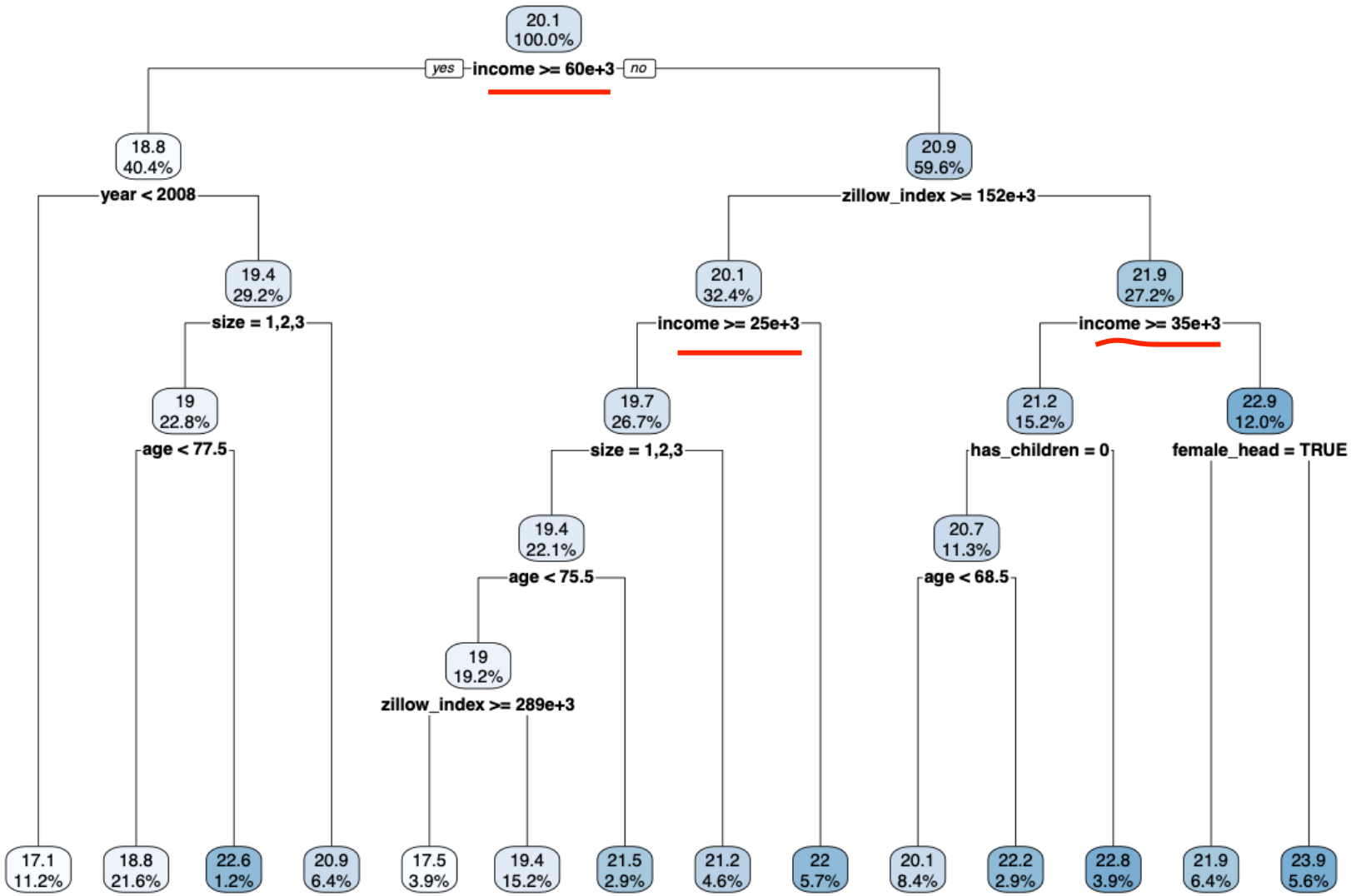
# Elementos de un Árbol de decisión

Los árboles se componen de:

- Un nodo inicial/raíz
- Nodos internos, cuya separación se marca por alguna $X_i$
- Nodos terminales/ Leafs

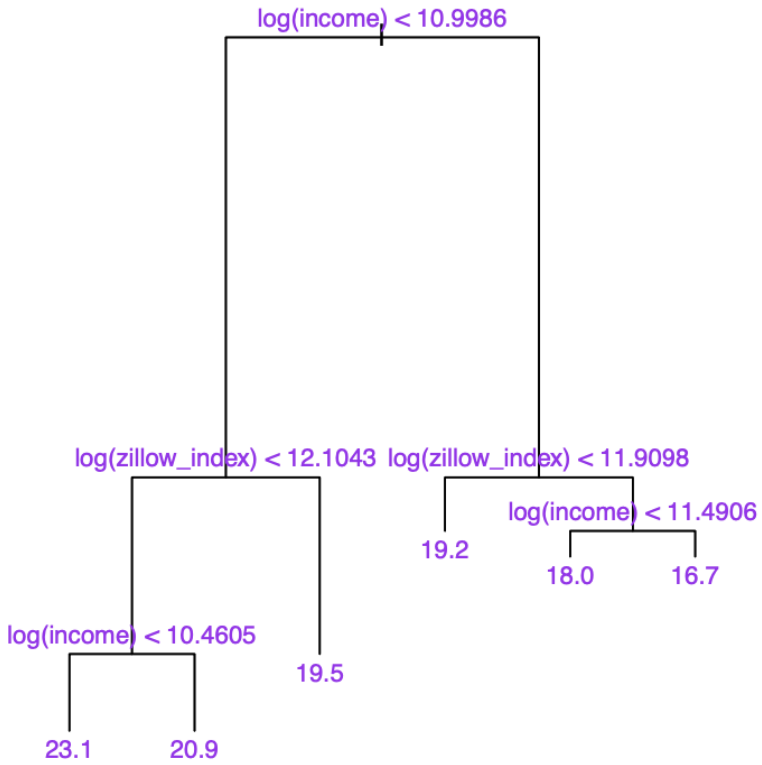
Las predicciones se marcan por algún estadístico (i.e. la media) de las observaciones de algún nodo terminal.

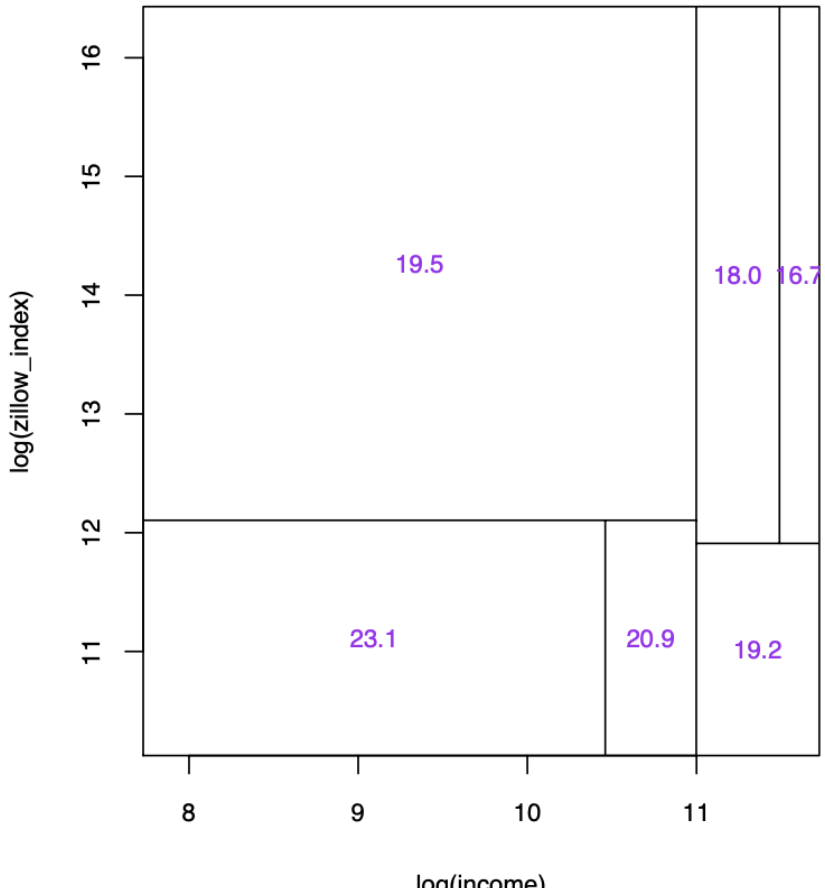
# Ejemplo II


20.1
100.0%
yes
income >= 60e+3
no
18.8
40.4%
year < 2008
20.9
59.6%
zillow_index >= 152e+3
19.4
29.2%
size = 1,2,3
20.1
32.4%
income >= 25e+3
21.9
27.2%
income >= 35e+3
19
22.8%
age < 77.5
19.7
26.7%
size = 1,2,3
21.2
15.2%
has_children = 0
22.9
12.0%
female_head = TRUE
19.4
22.1%
age < 75.5
20.7
11.3%
age < 68.5
19
19.2%
zillow_index >= 289e+3
17.1
11.2%
18.8
21.6%
22.6
1.2%
20.9
6.4%
17.5
3.9%
19.4
15.2%
21.5
2.9%
21.2
4.6%
22
5.7%
20.1
8.4%
22.2
2.9%
22.8
3.9%
21.9
6.4%
23.9
5.6%

# Arbol graficamente

En realidad, los árboles son un método de predicción **no paramétrico**. La predicción sucede mediante la estratificación del espacio de covariables



# Árboles como una piecewise function

En los árboles de regresión, $y_i$ es continua. La gran mayoría de las veces, $\hat{y_{iR}}$ va a ser la media de $y_i$ en el nodo terminal $R$.

Veamos como los árboles de regresión son como modelos de regresión piece-wise:

$$D_{iR} = 1\{i \in R\}$$

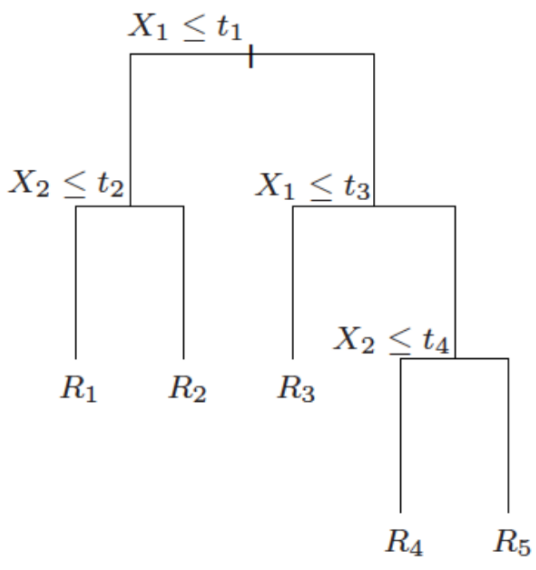
Se sigue que:

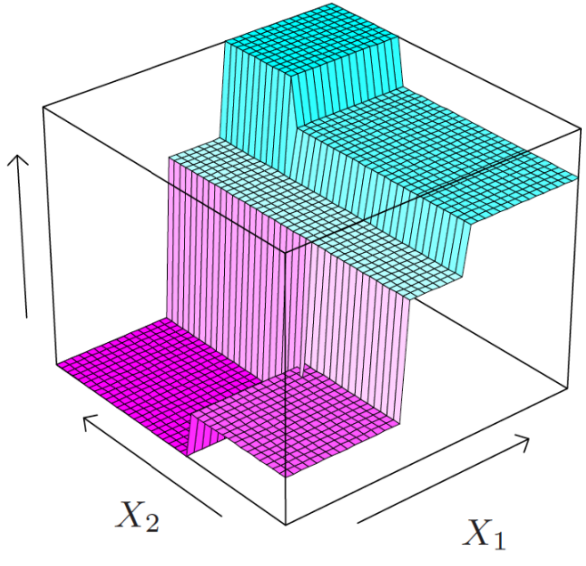
$$f(\hat{X}) = \hat{y_i} = \Sigma_{r=1}^{R} \hat{\beta}_R D_{iR} + \epsilon_i$$

Donde $\hat{\beta}_R$ varía dependiendo de si es un árbol de regresión o clasificación.

# Flexibilidad funcional

Noten como los árboles detectan las no-linealidades (interacciones en los modelos de regresión) de $f(.)$ de manera automática. Incluso algunas de alto valor polinómico.



# Funciones objetivo para árboles

Como siempre, queremos disminuir el error de predicción fuera de la muestra. En el caso de los árboles, estas son las funciones a minimizar:

Para $R$ particiones del espacio de covariables. Donde hay $j$ particiones.

- **Regresión**: $RSS = \Sigma_{j=1}^{J} \Sigma_{R_j} (y_i - \hat{y}_i)^2$. Esta es la misma desviación promedio que hemos visto para los modelos lineales.
- **Clasificación**: Para $k$ clases
  - Entropía: $-\Sigma \hat{p}_k log(\hat{p}_k)$
  - Gini: $\Sigma_k = \hat{p}_k(1 - \hat{p}_k)$

Noten como para ambas medidas estamos buscando **pureza**. Cuando $\hat{p}_k$ es cero o uno, Gini y Entropía se hacen más pequeños. En estos score predecidos, significaría que la mayoría de las observaciones en el *leaf* son de una sola clase.

# Cómo estimo los árboles?

Cómo hacer las particiones? Otra vez estamos enfrentados al problema de dimensionalidad. Es casi imposible estimar cada árbol posible.

- Cómo escoger que splits? Para cada $X$, el árbol puede evaluar un corte en cada punto del dominio. Son muchas posibilidades!
- Para una profundidad $d$, un årbol puede tener $2^d$ nodos terminales (asumiendo un árbol simétrico)

Si en cada nodo evalúa cada variable, tenemos $A_p * p * 2^d$ posibilidades (Donde A es la cardinalidad del dominio de cada variable).

Este número es gigante incluso para una profundida y base pequeña!!

# Cómo estimo los árboles?

Los árboles típicamente se estiman **top-down** y de forma **greedy**. Esto se llama **CART: Classfication and Regression Tree** algorithm de Brieman (1984):

1. Empiezas en el nodo inicial
2. Escoges el punto de corte & $x_i$ que disminuyen más el error de predicción
3. Divides la muestra en los nodos determinados en (2).
4. Repites 2 y 3 iterativamente

# Cómo estimo los árboles?

Veamos que significa cada parte:

**Top-Down**

- Son modelos jerárquicos. Una vez hecho un split, este no se puede deshacer.
- Los primeros splits juegan un rol super importante.

**Greedy**

- En cada splt, el árbol busca el mejor splt posible para disminuir el error de predicción. Ojo! No busca la secuencia de split; el mejor split en el margen. Por eso se les denomina greedy.

# Hasta donde estimar el árbol

El algoritmo descrito en las láminas anteriores tiene mucha propensión a caer en overfitting. Por ende, necesita reglas para detener el crecimiento del árbol.

Las dos más comunes son:

- Min final node `minsize`: Estima el árbol hasta que haya z o más observaciones en cada nodo terminal (típicamente 10)
- Min node size/ `min cut`: Cada split debe dejar al menos `min cut` observaciones de cada lado (típicamente 5)
- Minimum loss improvement / `min dev`: Estima el árbol hasta que la disminución de cada split llegue a un corte de disminución.

Ambas, especialmente la tercera estrategia tienen un problema de miopía: Un split mediocre puede ser el preludio de un split mucho mejor.

# Árboles de Regresión

Los árboles de regresión pueden ser entendidos como una función piecewise de las variables $X$

1. $y_i$ es continua
2. La decisión de split es la disminución del Residual Sum of Squares (RSS)
3. Las predicciones finales son medias $\frac{1}{N_r}\Sigma_{i \in r} y_i$

# Estimación de árboles regresión

Esto es, para una $X_k$ y un punto de corte $s$ definimos dos espacios $R_l$ y $R_r$:

$$R_l(k,s) : \{i : x_{ik} < s\} \qquad R_r(k,s) : \{i : x_{ik} \geq s\}$$

El RSS resultante es ahora:

$$RSS^1 = \Sigma_{i \in R_l}(y_i - \hat{y}_i)^2 + \Sigma_{i \in R_r}(y_i - \hat{y}_i)^2$$

Donde $RSS^1 < RSS(\textit{modelo anterior})$

El proceso sigue hasta que se llega a un criterio de finalización. Noten como para una variable $X \in \{1, ..., 100\}$ el árbol puede evaluar cortes en 1,2,3,4..... en cada valor!

# Árboles en R

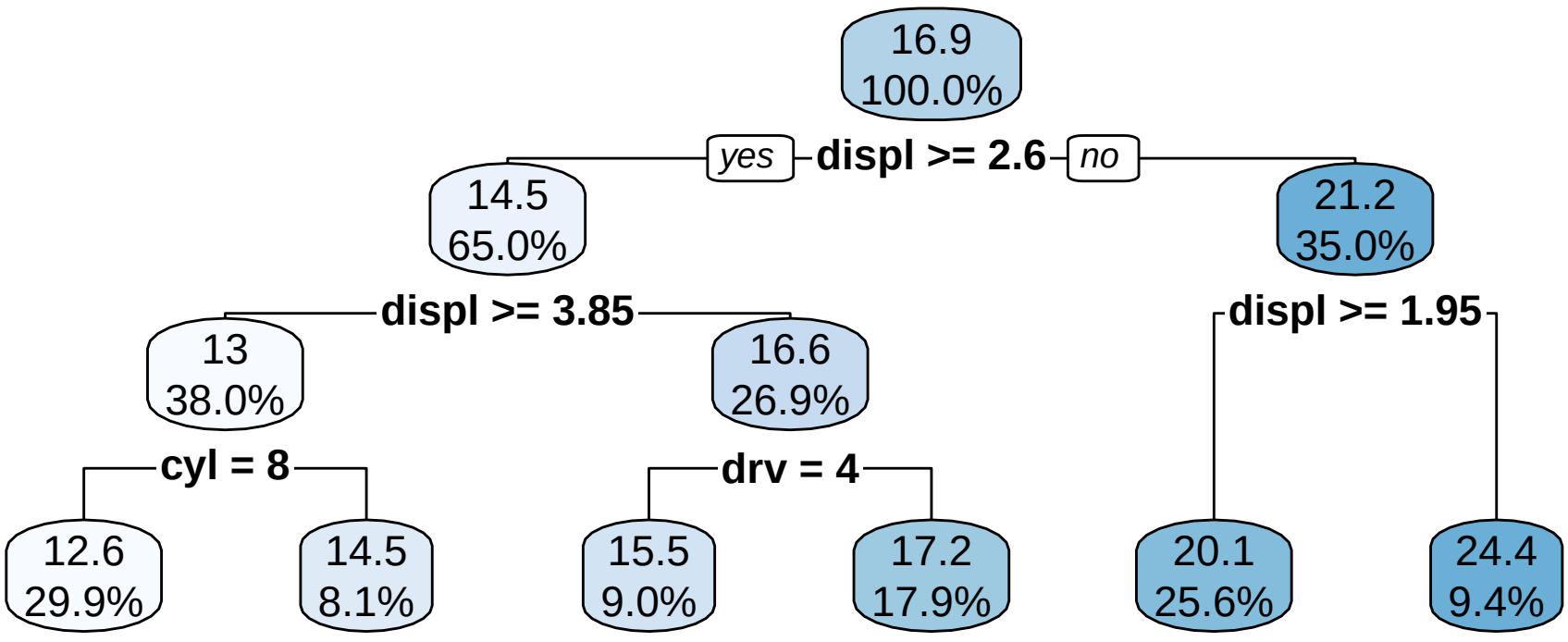
Las librerías para estimar árboles es `tree` y `rpart`. La primera es la más sencilla; la segunda produce árboles más bonitos de graficar.

```
library(MASS)
library(tidyverse)
library(tree)
library(rpart)
library(broom)
library(rpart.plot)
data('mpg', package='ggplot2')
mpg <- mpg %>%
  mutate(drv=factor(drv),
         cyl = factor(cyl) )
```

# Árboles en R

Una estimación básica

```
t <- rpart( cty ~ displ + year + cyl + drv, data=mpg)
rpart.plot(t, digits =3)
```



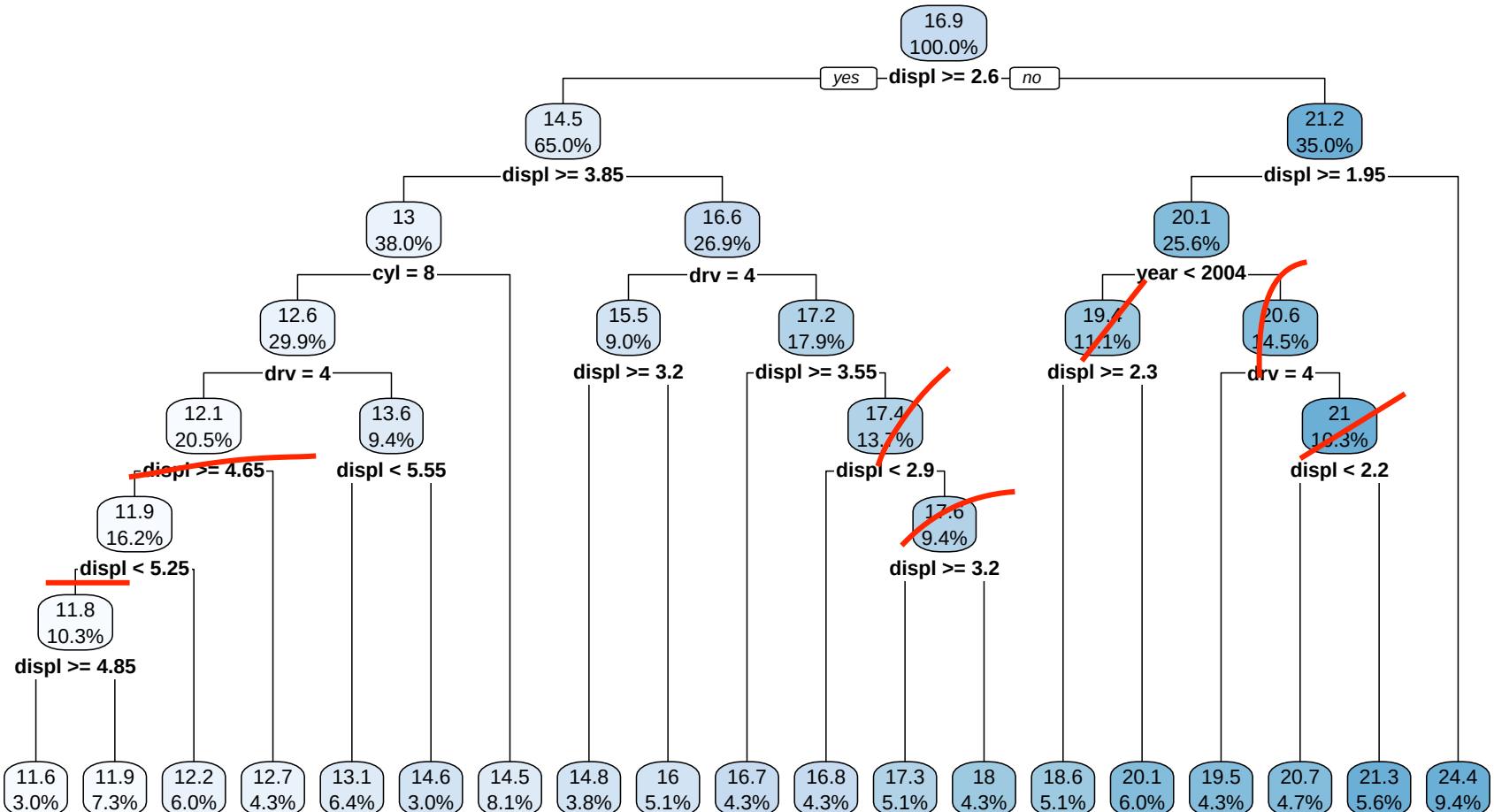
# Árboles de Regresión en R

```
summary(t)
```

```
## Call:
## rpart(formula = cty ~ displ + year + cyl + drv, data = mpg)
##   n= 234
##
##           CP nsplit rel error    xerror       xstd
## 1 0.57193143      0 1.0000000 1.0084882 0.12198973
## 2 0.11620091      1 0.4280686 0.4328425 0.06157101
## 3 0.06988131      2 0.3118677 0.3320610 0.06029697
## 4 0.01353184      3 0.2419864 0.2690801 0.04344369
## 5 0.01002137      4 0.2284545 0.2597797 0.04357175
## 6 0.01000000      5 0.2184331 0.2667722 0.04353982
##
## Variable importance
## displ   cyl   drv
##    47    38    14
##
## Node number 1: 234 observations,    complexity param=0.571931
```

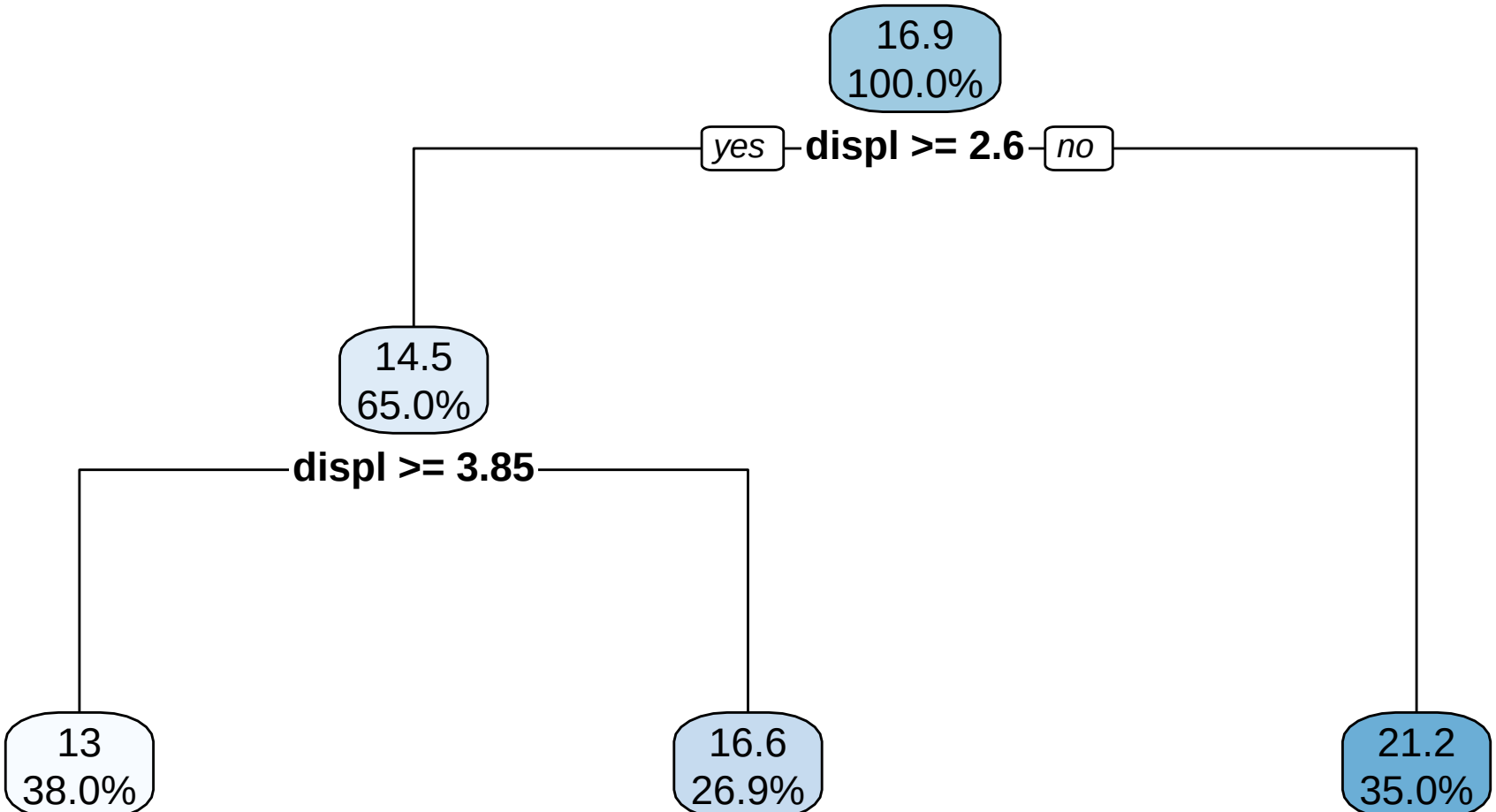
# Movemos `tree::min.dev` `rpart::cp`

```
t2 <- rpart( cty ~ displ + year + cyl + drv, data=mpg,
             cp = 0.0001)
rpart.plot(t2, digits =3)
```



# Movemos `rpart::minbucket`

```
t3 <- rpart( cty ~ displ + year + cyl + drv, data=mpg,
             minbucket = 50)
rpart.plot(t3, digits =3)
```



# Árboles de Clasificación

Los árboles de clasificación son muy similares a los árboles de regresión. Las diferencias son:

1. $y_i \in \{1, 2, 3, ...K\}$
2. La decisión de split es Gini o Entropy (vs RSS)
3. Las predicciones NO son medias.

Como son las predicciones entonces? Los árboles de clasificación te arrojan dos tipos:

a. Predicción categórica: El árbol predice la clase $k$ más común en $r \in R$

b. Predicción continua: El árbol te da las probabilidades o scores como la proporción de casos en $r \in R$

# Estimación de árboles de clasificación

Buscamos minimizar el error de clasificación de entropía $-\Sigma\hat{p}_k log(\hat{p}_k)$ o Gini $\Sigma_k = \hat{p}_k(1-\hat{p}_k)$

Esto es, para una $X_k$ y un punto de corte $s$ definimos dos espacios $R_l$ y $R_r$:

$$R_l(k,s) : \{i : x_{ik} < s\} \qquad R_r(k,s) : \{i : x_{ik} \geq s\}$$

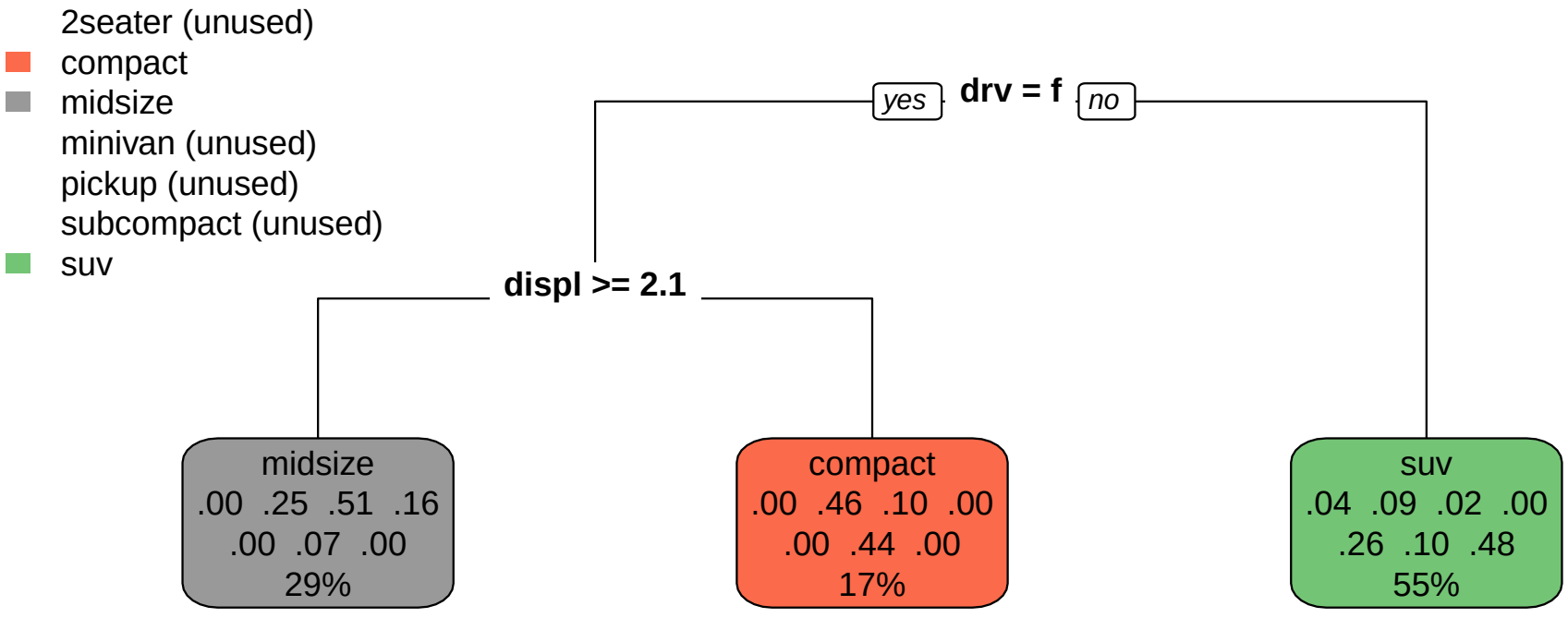
El estadístico resultante es ahora:

$$Gini^1 = \Sigma_{i \in R_l}\Sigma_k \hat{p}_k(1-\hat{p}_k) + \Sigma_{i \in R_r}\Sigma_k \hat{p}_k(1-\hat{p}_k)$$

Donde $Gini^1 < Gini(modelo\ sin\ split)$

# Ejemplo Clasificación

```
t3 <- rpart( class ~ displ + year + cyl + drv, data=mpg,
             cp=0.05)
rpart.plot(t3, digits =2,type = 0)
```



# Ejemplo Clasificación

```
predict(t3)[1:5, ]
```

```
##   2seater   compact   midsize   minivan pickup subcompact suv
## 1       0 0.4615385 0.1025641 0.0000000      0 0.43589744   0
## 2       0 0.4615385 0.1025641 0.0000000      0 0.43589744   0
## 3       0 0.4615385 0.1025641 0.0000000      0 0.43589744   0
## 4       0 0.4615385 0.1025641 0.0000000      0 0.43589744   0
## 5       0 0.2537313 0.5074627 0.1641791      0 0.07462687   0
```

# Tree pruning

Hasta ahora vimos que estimamos el árbol hasta que `mincut` o `min dev` lo marquen. Esto genera tres problemas: Es díficil no caer en overfitting y no consideramos la posibilidad de split mediocres seguidos de grandes splits; Hay muchas posibilidades!

Por lo anterior, es mejor estrategia estimar un árbol grande y despuer podarlo (prune!). Como? .... Cross-Validation!

- Estimemos varios árboles y, usando validación cruzada (k-fold) escojamos el mejor modelo.

Preguntas:

- Por qué no podemos usar criterios de información aquí como AICc?
- Por qué no tenemos un buen estimado de los grados de libertad (No paramétrico!)
- Cuales son las posibilidades de árboles a estimar?
- Aquí es más díficil reducir el problema de **modelos candidatos** vs

# Cost complexity pruning

Cost complexity pruning (Weakest link pruning) es la manera más común de reducir el número de modelos candidatos y aplicar k-fold CV.

El proceso funciona así:

1. Estima árbol de manera greedy hasta llegar a `min obs`.
2. Aplica un parámetro de complejidad $\alpha$ que castigue la complejidad del árbol. Esto te dará una secuencia de subárboles como función de $\alpha$.
3. Aplica k-fold CV para escoger $\alpha$

# Cost complexity pruning

Veamos la formula de coxt complexity pruning:

$$\Sigma_{j=1}^{J}\Sigma_{R_j}(y_i - \hat{y}_i)^2 + \alpha|T|$$

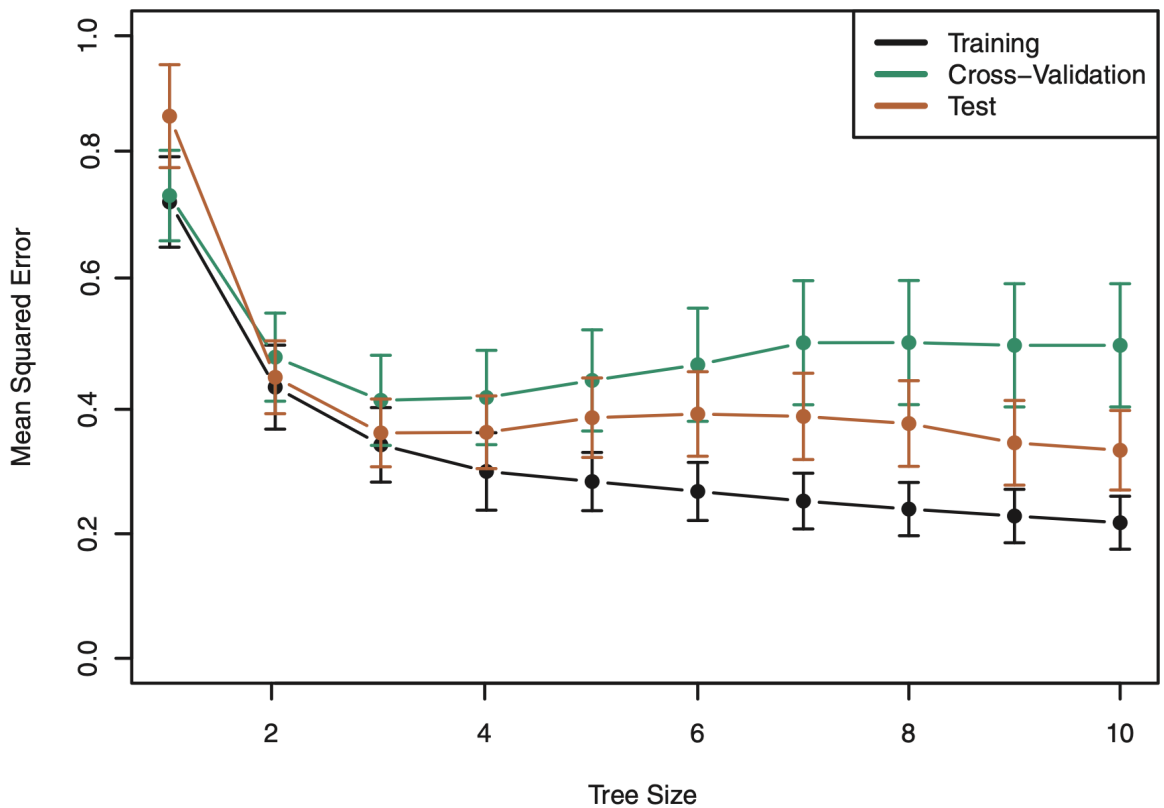
Donde $T$ es el número de nodos terminales. Se ve familiar?

Resulta que si un $\alpha$ muy grande, tendremos un árbol con $T_0 \subset T$. Mientras $\alpha$ tiende a cero $T_0 \to T$.

$\alpha$ nos ayuda a tener un set de candidatos de modelo estimable vía K-fold CV!!

# Grafica de desempeño de CV tree

Noten como el CV es un buen estimador del desempeño fuera de la muestra (3 nodos terminales). Adicionalmente, noten como en el training set, el árbol siempre parece que va a estimar mejor mientras más grande haga los árboles.



# Ejemplo de CV tree

```
cpus.ltr2 <- tree(log10(perf) ~ syct+mmin+mmax+cach
                  +chmin+chmax, cpus, minsize = 1)
(cv_tree<-cv.tree(cpus.ltr2, K= 5))
```

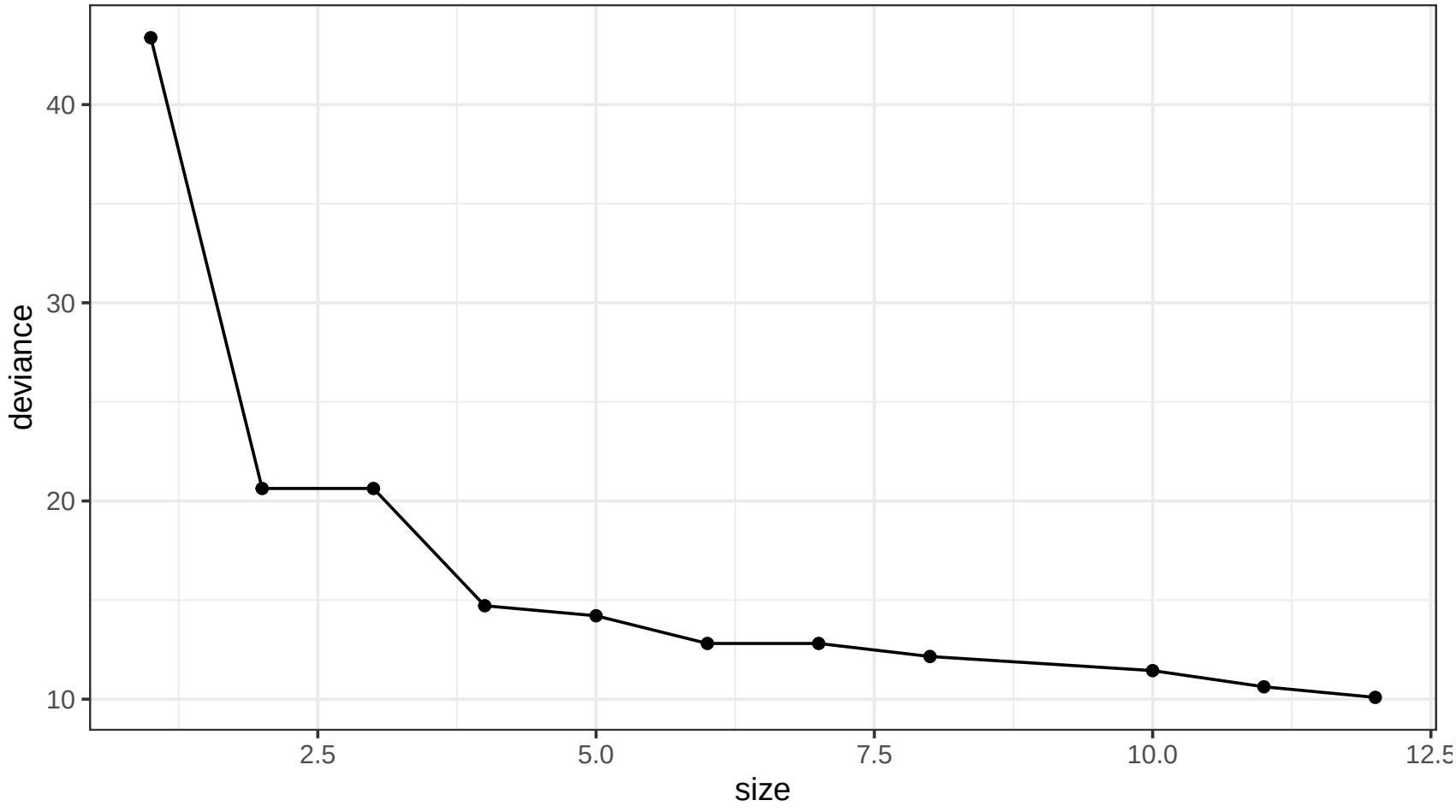
```
## $size
##  [1] 12 11 10  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 10.08943 10.62383 11.44054 12.15251 12.81032 12.81032 14
##  [9] 20.63110 20.63110 43.37863
##
## $k
##  [1]       -Inf  0.4528334  0.5246933  0.6808309  0.7435687
##  [7]  1.1607588  1.4148749  3.7783549  3.8519002 23.6820624
##
## $method
## [1] "deviance"
##
## attr( "class")
```

# Ejemplo de CV tree

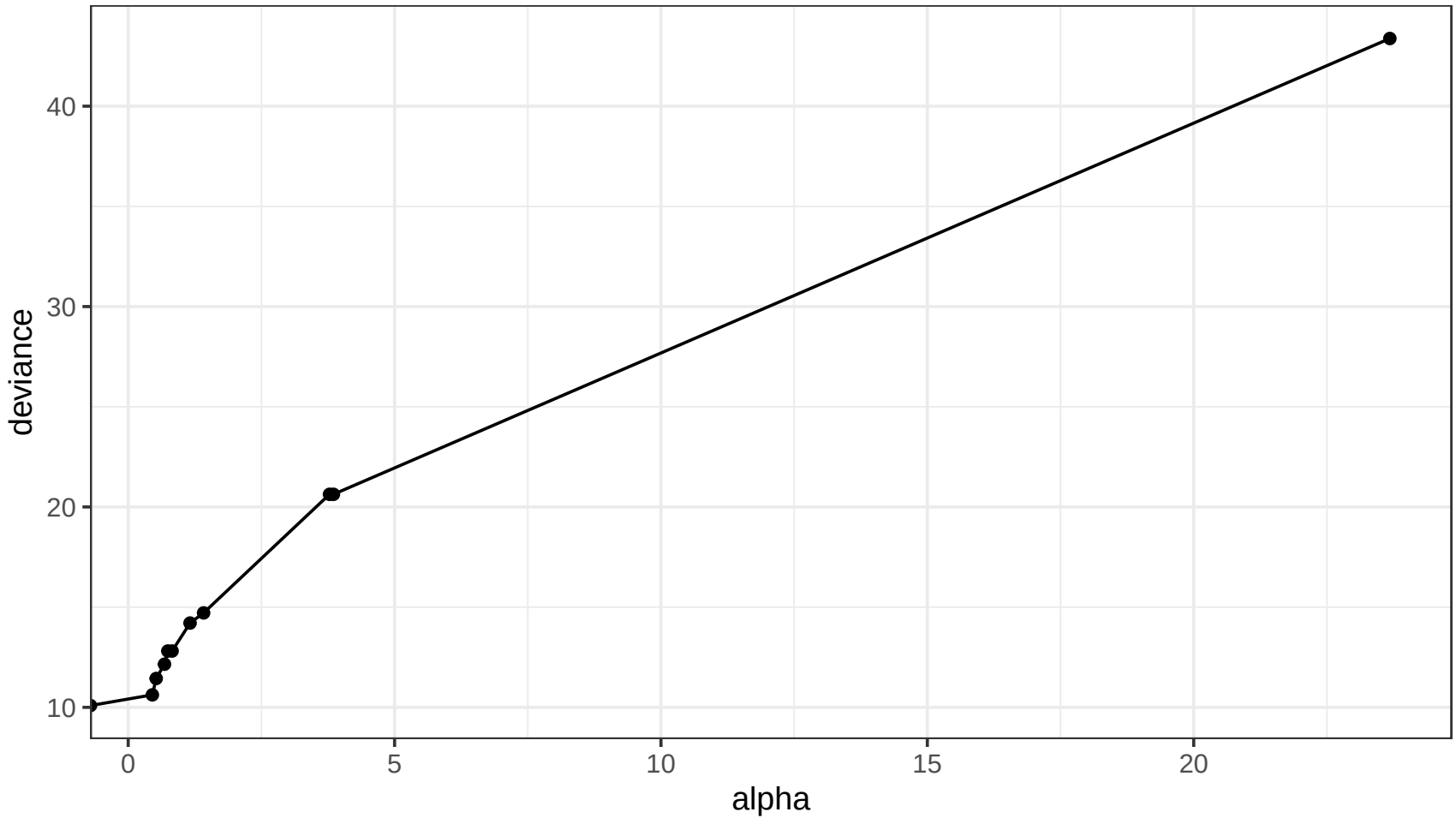
El resultado `size` es el numero de nodos terminales; `dev` corresponde al deviance CV; `k` $= \alpha$.

Vemos que con 10 nodos terminales, logramos la menor deviance.

# Ejemplo de CV tree



# Ejemplo de CV tree



# Árboles vs métodos lineales

Qué es mejor? Métodos lineales o árboles? Depende del problema.

Si el problema tiene altas no-linealidades, los árboles son mucho mejores. Esto es porque detectan las mismas de manera automática.

Si el problema es más aproximable de manera lineal, los algoritmos lineales lo harán mejor.

Pros árboles:

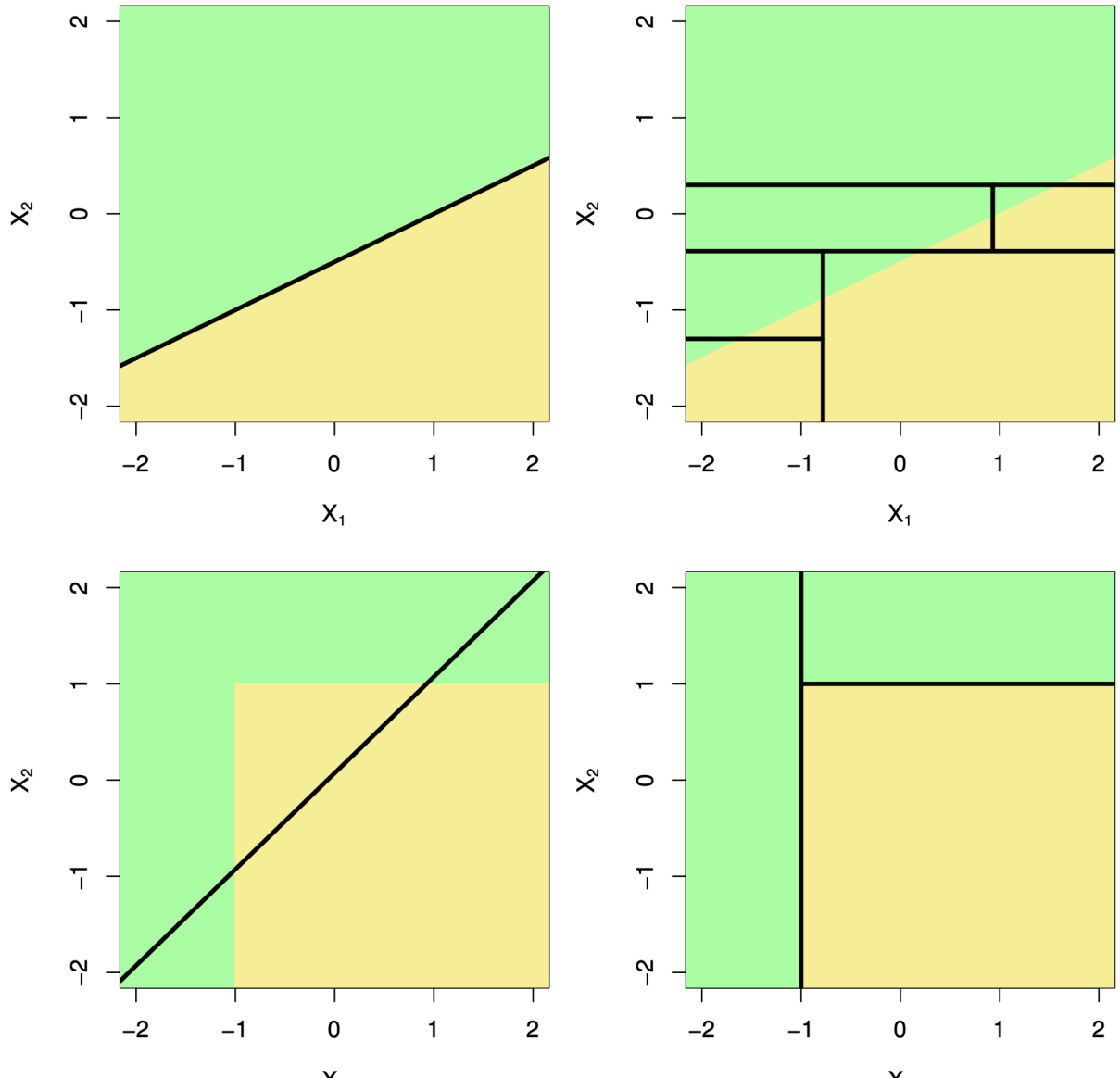
- Muy intuitivos. Esto los hace muy fáciles de explicar.
- Parecen ser más intuitivos porque se asemejan más al razonamiento humano (ifelse nodes).
- Los árboles pueden manejar fácilmente variables categóricas sin necesidad de crear dummys.

# Árboles vs métodos lineales

Cons:

- Al ser un método no paramétrico. Tienden a no ser estimadores muy estables antes nuevos datos.
- Incluso con CV, es díficil no caer en overfitting.

# Árboles vs métodos lineales

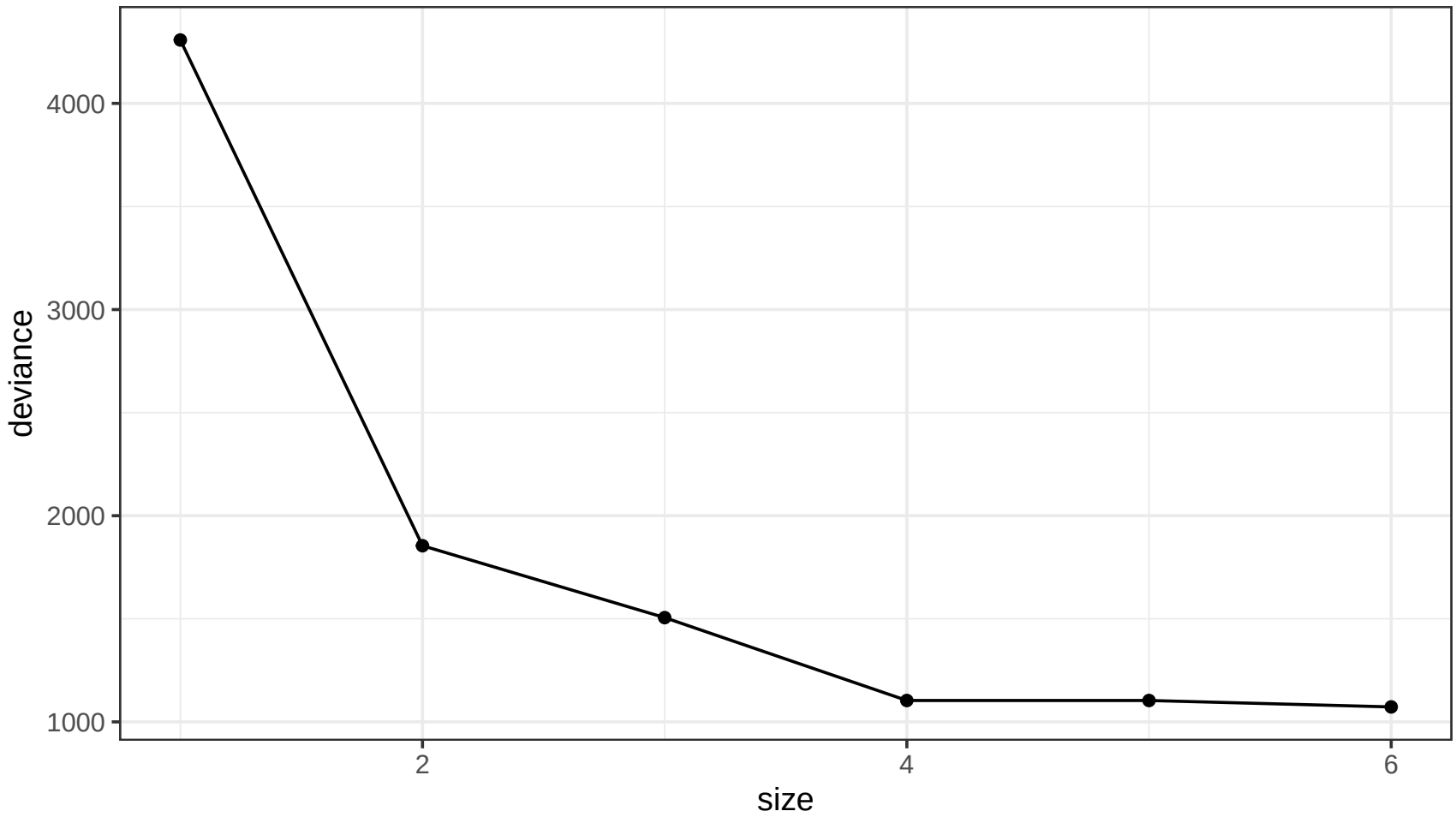


# Ejemplo II

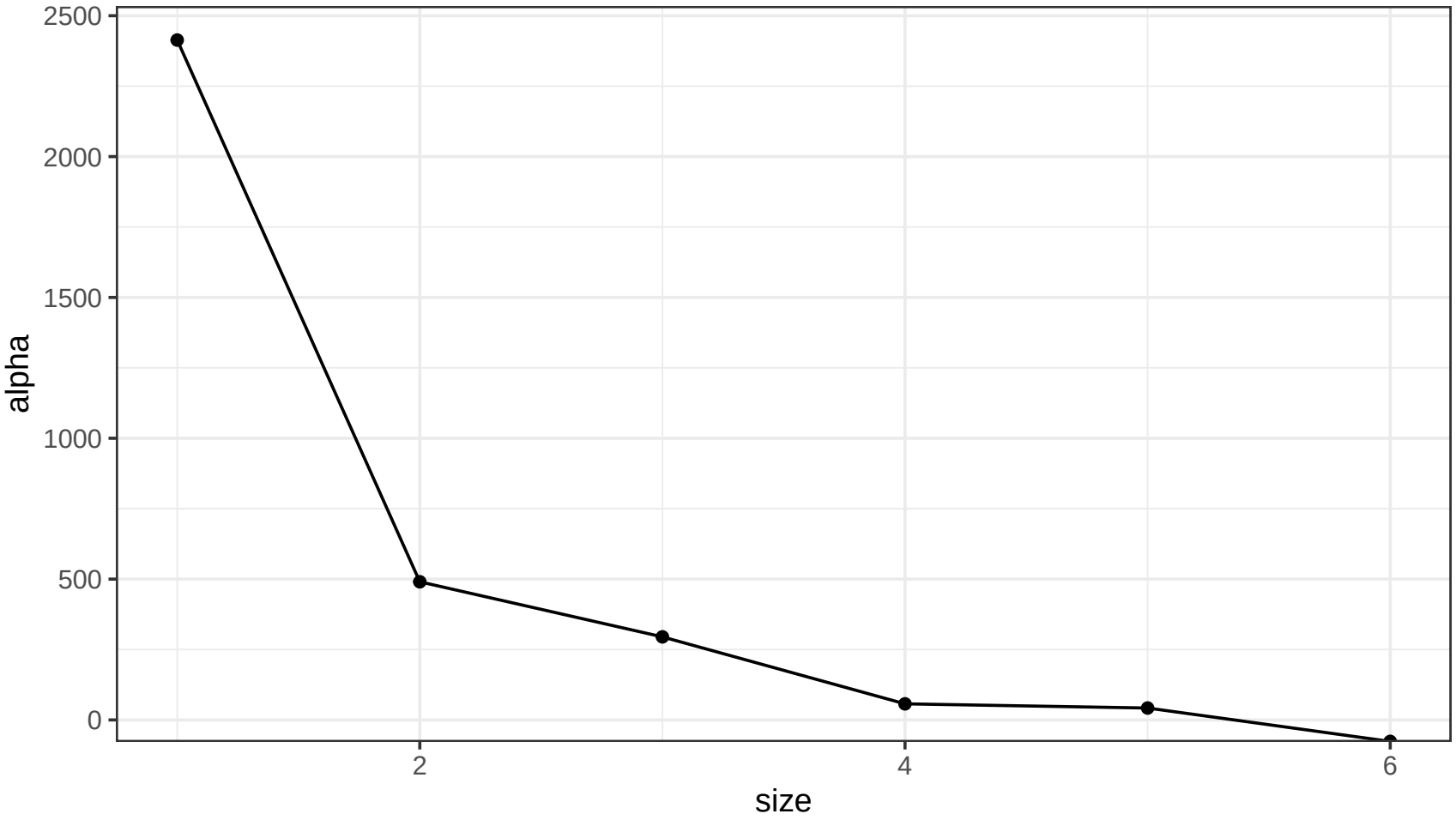
```
t3 <- cv.tree( tree(cty ~ displ + year + cyl + drv, data=mpg), K
t3
```

```
## $size
## [1] 6 5 4 3 2 1
##
## $dev
## [1] 1072.543 1103.718 1103.718 1505.690 1854.455 4307.611
##
## $k
## [1]       -Inf   42.29365   57.10906  294.92332  490.40806 24
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

# Ejemplo II



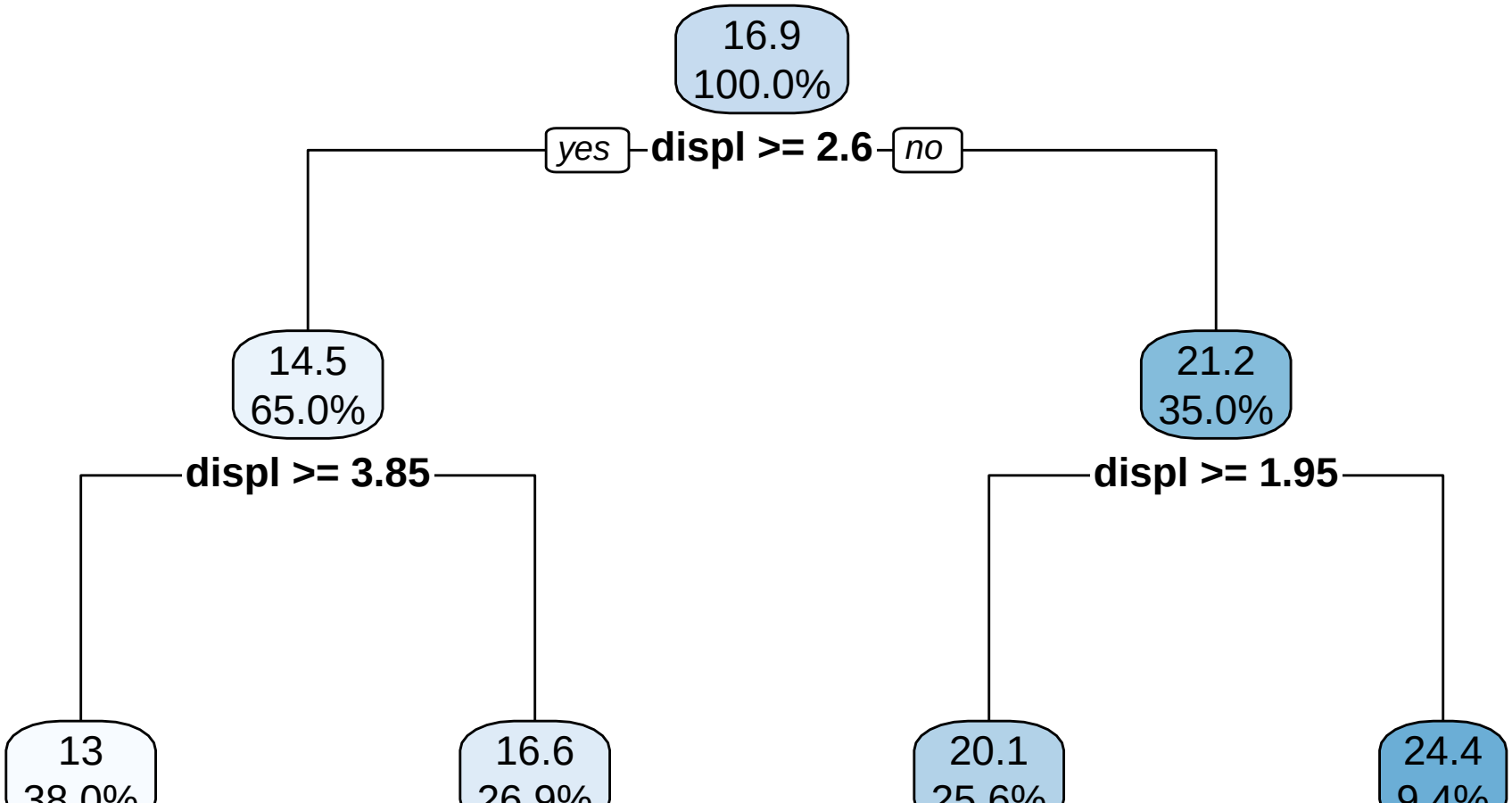
# Ejemplo II



# Prunes tree

Escogemos el

```
t4 <- prune(t, cp = 0.05)
rpart.plot(t4, digits =3)
```



# Resumen Trees

- Los árboles de decisión (Regresión y Clasificación) son algoritmos no paramétricos
- Son fáciles de interpretar!
- Detectan no linealidades de manera automática
- Se estiman de manera **greedy**, donde se buscan splits que maximicen la varianza entre nodos (inter-varianza) y se minimice la varianze dentro de cada nodo (intra-varianza)
- Hay dos maneras de parar la estimación de los arboles: observaciones mínimas en los nodos terminales o disminución en el deviance mínima.
- El greediness es miope. Esto es, un split malo puede seguirle un gran split.
- Es díficil evitar el overfitting en los árboles.

# Resumen Tree

- Para ayudar al overfitting, podemos hacer Tree pruning con Cross Validation
- Esto hara que estimemos greedy y luego tengamos un set de árboles candidatos dependiendo de cuánto castigamos la complejidad de los mismos.
- Los prune.trees son más poderosos (predicción) que los árboles normales. Por ende, preferibles.
- Aún así, los árboles son estimadores de alta varianza. Esto es, varían mucho dependiendo de la muestra en la que nos encontremos (no paramétrico). Por ende, necesitamos alguna manera de estabilizar las predicciones de los árboles para lograr mejor poder predictivo OOS.
- Los Random Forests y los Gradient Boosting Machines usan Boostrapping y otros métodos para lograr esto! (Próxima clase!)

# Bias and variance ejemplo