

Lec5: Árboles de Decisión

Isidoro Garcia Urquieta

2021

Agenda

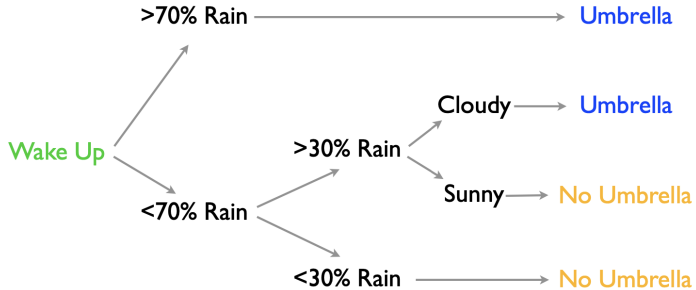
- ▶ Árboles de Clasificación
- ▶ Árboles de Regresión
- ▶ Recursive binary splitting: Decision split for each case
- ▶ Tree Pruning
- ▶ Linear vs Trees
- ▶ Bootstrapping & Bagging

Qué es un árbol 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



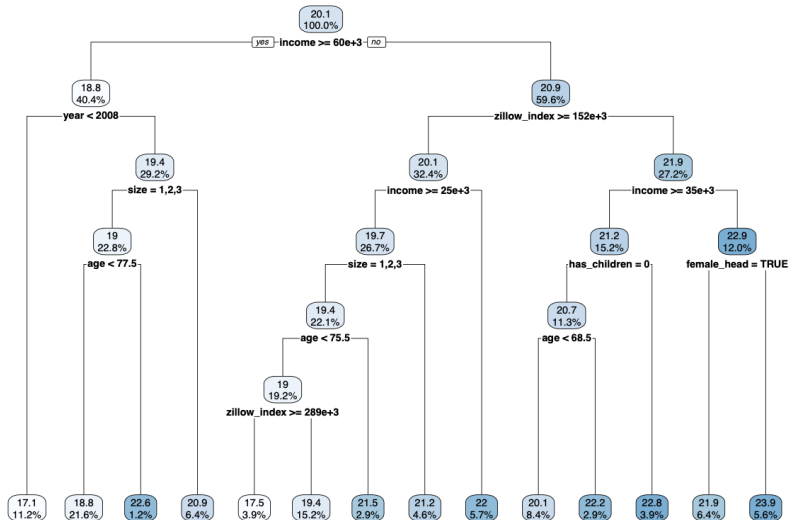
Elementos de un Árbol de decisión

Los árboles se componen de:

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

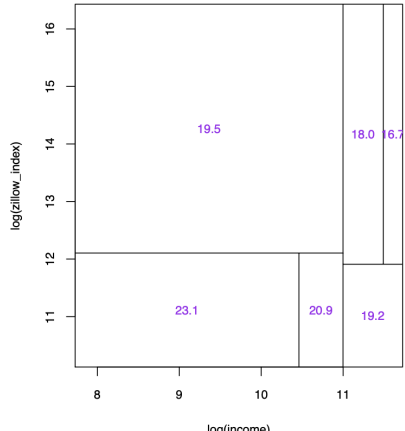
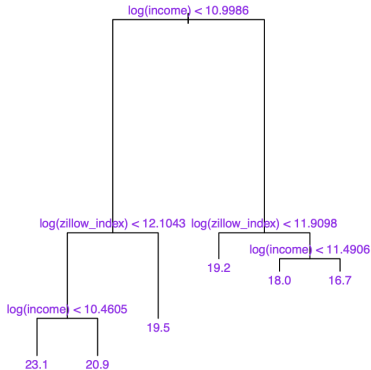
Las predicciones se marcan por algún estadístico (i.e. la media) de las observaciones de algún nodo terminal.

Ejemplo II



Arbol graficamente

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



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

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.

Árboles de Regresión

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:

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

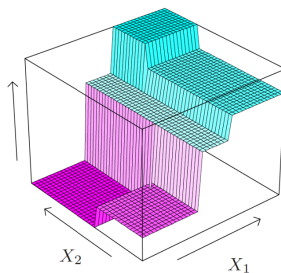
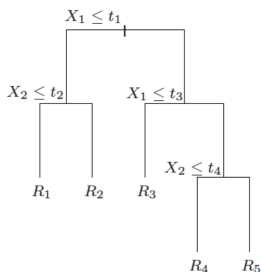
Se sigue que:

$$\hat{y}_i = \sum_{r=1}^R \hat{\beta}_R D_{iR} + \epsilon_i$$

Donde $\hat{\beta}_R = \frac{1}{N_r} \sum_{i \in r} y_i$

Árboles de Regresión

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



Árboles de Clasificación

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

1. $y_i \in \{1, 2, 3, \dots, K\}$
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

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

Sin tomar en cuenta esto, los árboles típicamente se estiman **top-down** y de forma **greedy**. Esto se llama **CART: Classification and Regression Tree** algorithm de Brieman (1984):

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

Estimación de árboles

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

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

El RSS resultante es ahora:

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

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

El proceso sigue hasta que se llega a un criterio de finalización.

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 node size / min cut: Estima el árbol hasta que haya z o más observaciones en cada nodo terminal (típicamente 10).
- ▶ Minimum loss improvement / min dev: Estima el árbol hasta que la disminución de cada split llegue a un corte de disminución.

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

Árboles en R

La librería para estimar árboles es `tree`. Los otros dos argumentos relevantes son: `mincut`: observaciones mínimas en los nodos terminales y `mindev`: es la ganancia en deviance mínima (en proporción) para que haga un split.

```
library(MASS)
library(tidyverse)
library(tree)
library(broom)
data(cpus, package = 'MASS')

cpus.ltr <- tree(log10(perf) ~ syct+mmin+mmax+cach
                 +chmin+chmax, cpus)
```

Árboles en R

```
summary(cpus.ltr)
```

```
##
```

```
## Regression tree:
```

```
## tree(formula = log10(perf) ~ syct + mmin + mmax + cach + chmi
```

```
##      chmax, data = cpus)
```

```
## Variables actually used in tree construction:
```

```
## [1] "cach" "mmax" "syct" "chmin"
```

```
## Number of terminal nodes: 10
```

```
## Residual mean deviance: 0.03187 = 6.342 / 199
```

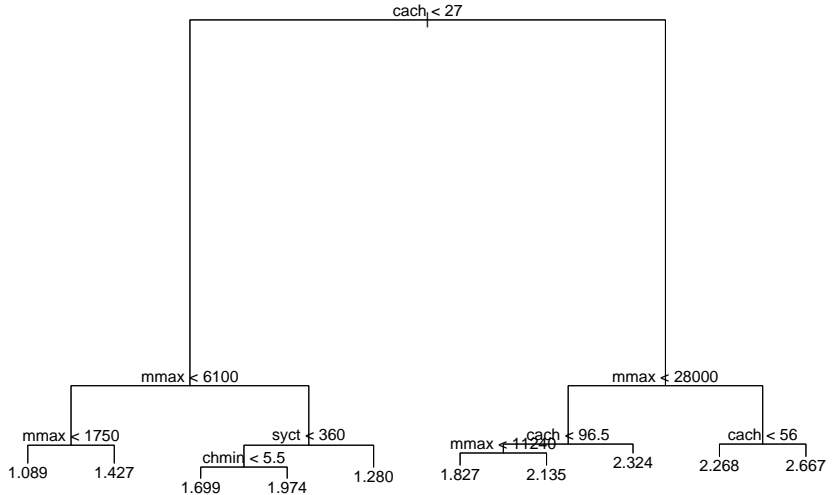
```
## Distribution of residuals:
```

```
##      Min.      1st Qu.      Median      Mean      3rd Qu.
```

```
## -0.4945000 -0.1191000  0.0003571  0.0000000  0.1141000  0.468
```


Árboles en R

```
plot(cpus.ltr); text(cpus.ltr)
```



Tree pruning

Hasta ahora vimos que estimamos el árbol hasta que mincut o min dev lo marquen. Esto genera dos problemas: Es difícil no caer en overfitting y no consideramos la posibilidad de split mediocres seguidos de grandes splits.

Por lo anterior, es mejor estrategia estimar un árbol grande y despues 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 difícil reducir el problema de **modelos candidatos** vs LASSO.

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 α que castigue la complejidad del árbol. Esto te dará una secuencia de subárboles como función de α .
3. Aplica k-fold CV para escoger α

Cost complexity pruning

Veamos la formula de cost complexity pruning:

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

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

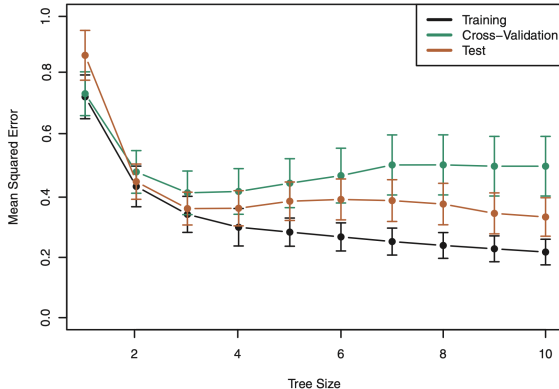
Resulta que si un α muy grande, tendremos un árbol con $T_0 \subset T$.

Mientras α tiende a cero $T_0 \rightarrow T$.

α 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

```
(cv_tree<-cv.tree(cpus.ltr, K= 5))
```

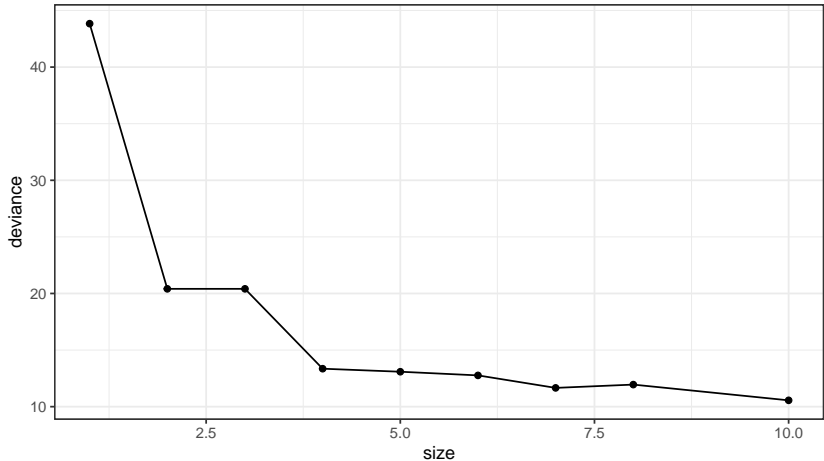
```
## $size
## [1] 10  8  7  6  5  4  3  2  1
##
## $dev
## [1] 10.56032 11.94992 11.65978 12.76386 13.08661 13.35461 20.
## [9] 43.83733
##
## $k
## [1]      -Inf  0.6808309  0.7243056  0.8000558  1.1607588 1
## [8]  3.8519002 23.6820624
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"          "tree.sequence"
```

Ejemplo de CV tree

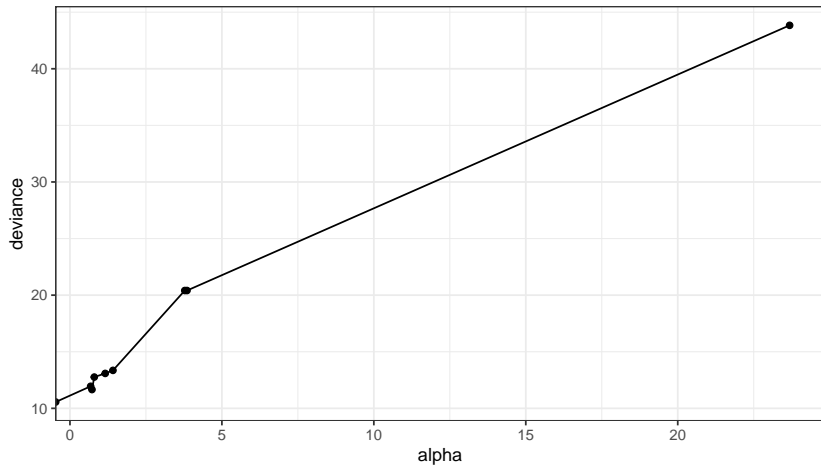
El resultado `size` es el numero de nodos terminales; `dev` corresponde al deviance CV; $k = \alpha$.

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 dummies.

Árboles vs métodos lineales

Cons:

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

Boostrapping

Llegamos al segundo método de **remuestreo** (El primero fue CV)!

El bootstrapping es muy útil para conseguir errores estándares e intervalos de confianza cuando la distribución de algún estimador es desconocida, tenemos pocas observaciones o es muy difícil de estimar.

Por ejemplo, en el caso de la econometría, podemos inferir la distribución de nuestro estimador $\hat{\beta}$ al remuestrear la base con reemplazo y estimar β con cada muestra.

Lo bonito del Bootstrapping es que tiene otras aplicaciones además de la generación de errores estándar. Se puede aplicar a mejorar el poder predictivo de nuestros modelos estadísticos.

Boostrapping

Pasos:

Dada una base de datos $\{z_i\}_{i=1}^n$, para $b \in \{1, 2, 3, \dots, B\}$ muestras:

1. Tomas una muestra **con reemplazo** $\{z_i^b\}_{i=1}^n$
2. Estimás β_b usando la base de (1)

Luego, $\{\beta_b\}_{b=1}^B$ es una aproximación de la distribución de $\hat{\beta}$

El vector de tamaño B $\{\beta_b\}_{b=1}^B$ puede ser usado como la distribución teorica que aprendimos en la clase 2 (inferencia!) que usaba el Teorema del Límite Central.

$$\beta \in \hat{\beta} \pm 2sd(\hat{\beta}_b)$$

$$se(\hat{\beta}) \simeq sd(\beta_b) = \sqrt{\frac{1}{B} \sum_b (\hat{\beta}_b - \hat{\beta})^2}$$

Donde $\hat{\beta}$ es el estimador original usando toda la base de datos.

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 difícil 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 Bootstrapping y otros métodos para lograr esto! (Próxima clase!)