

# Árboles (CARTs), Bagging and Random Forests

## Big Data y Machine Learning para Economía Aplicada

Ignacio Sarmiento-Barbieri

Universidad de los Andes

# Motivación

- Queremos predecir:

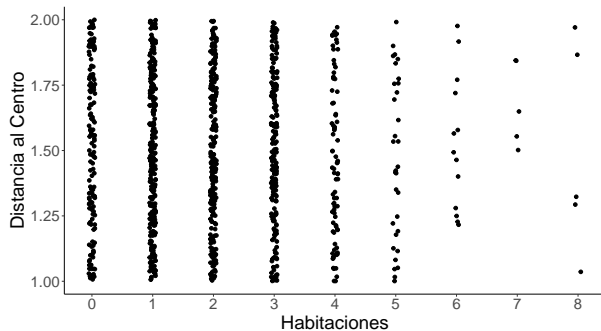
$$Price = f(\text{structural attributes}, \text{amenities}, \dots) \quad (1)$$

- Podemos aplicar linear regression,

$$Price = \beta_0 + \beta_1 \text{Habitaciones} + \beta_2 \text{DCBD} + u \quad (2)$$

- Aplicar OLS a este problema requiere tomar algunas decisiones.

# Motivación



# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

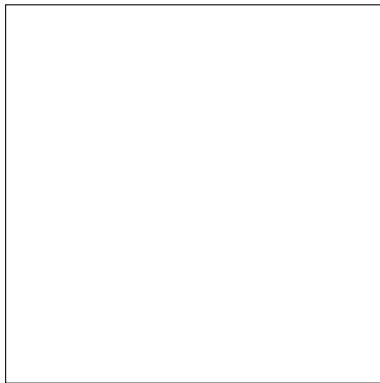
- Random Forests

## 3 Boosting

# ¿Qué hacen?

“Recursive binary splitting”

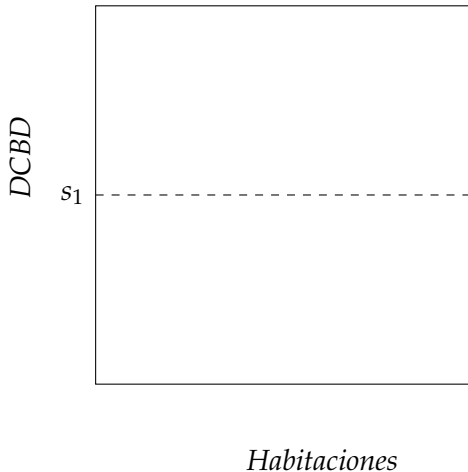
*DCBD*



*Habitaciones*

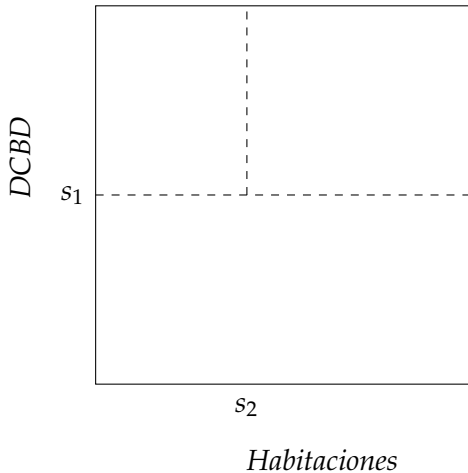
# ¿Qué hacen?

“Recursive binary splitting”



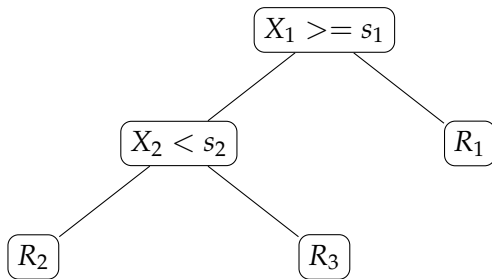
# ¿Qué hacen?

“Recursive binary splitting”

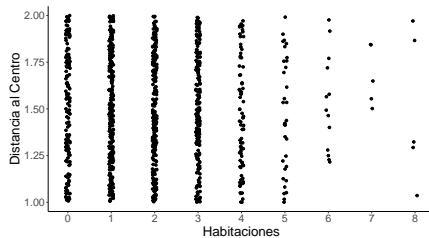
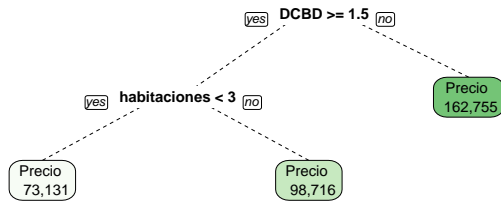




¿Qué hacen?



# ¿Qué hacen?



# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# ¿Cómo construimos un árbol de decisión?

- ▶ Datos:  $y_{n \times 1}$  y  $X_{n \times k}$
- ▶ Definiciones
  - ▶  $j$  es la variable que parte el espacio y  $s$  es el punto de partición
  - ▶ Defina los siguientes semiplanos

$$R_1(j, s) = \{X | X_j \leq s\} \ \& \ R_2(j, s) = \{X | X_j > s\} \quad (3)$$

- ▶ El problema: usando una “perdida cuadrática” buscar la variable de partición  $X_j$  y el punto  $s$  de forma tal que:

$$\min_{j,s} \left[ \min_{y_{R_1}} \sum_{x_i \in R_1(j,s)} (y - y_{R_1})^2 + \min_{y_{R_2}} \sum_{x_i \in R_2(j,s)} (y - y_{R_2})^2 \right] \quad (4)$$

# ¿Cómo construimos un árbol de decisión?

- ▶ Datos:  $y_{n \times 1}$  y  $X_{n \times k}$
- ▶ Definiciones
  - ▶  $j$  es la variable que parte el espacio y  $s$  es el punto de partición
  - ▶ Defina los siguientes semiplanos

$$R_1(j, s) = \{X | X_j \leq s\} \ \& \ R_2(j, s) = \{X | X_j > s\} \quad (3)$$

- ▶ El problema: usando una “perdida cuadrática” buscar la variable de partición  $X_j$  y el punto  $s$  de forma tal que:

$$\min_{j,s} \left[ \min_{y_{R_1}} \sum_{x_i \in R_1(j,s)} (y - y_{R_1})^2 + \min_{y_{R_2}} \sum_{x_i \in R_2(j,s)} (y - y_{R_2})^2 \right] \quad (4)$$

- ▶ ¿Cuál es la solución?

# ¿Cómo construimos un árbol de decisión?



photo from <https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/>

# Agenda

## 1 Árboles

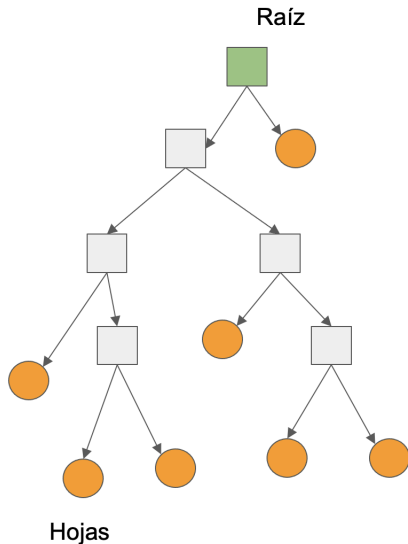
- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# Sobreajuste





# Sobreajuste. Algunas soluciones

- ▶ Fijar la profundidad del árbol.
- ▶ Fijar la mínima cantidad de datos que están contenidos dentro de cada hoja.
- ▶ Pruning (poda).
  - ▶ Dejar crecer un árbol muy grande  $T_0$
  - ▶ Luego cortarlo obteniendo sub-árbol (*subtree*)
  - ▶ Como cortarlo?

# Pruning (poda)

- ▶ No es posible calcular el error de predicción usando cross-validation para cada sub-árbol posible
- ▶ Solución: *Cost complexity pruning* (cortar las ramas mas débiles)
  - ▶ Indexamos los arboles con  $T$ .
  - ▶ Un sub-árbol  $T \in T_0$  es un árbol que se obtuvo colapsando los nodos terminales de otro árbol (cortando ramas).
  - ▶  $[T]$  = número de nodos terminales del árbol  $T$

# Pruning (poda)

- Cost complexity del árbol  $T$

$$C_{\alpha}(T) = \sum_{m=1}^{[T]} n_m Q_m(T) + \alpha [T] \quad (5)$$

- donde  $Q_m(T) = \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2$  para los árboles de regresión
- $Q_m(T)$  penaliza la heterogeneidad dentro de la regresión y el número de regiones
- Objetivo: para un dado  $\alpha$ , encontrar el pruning óptimo que minimice  $C_{\alpha}(T)$

# Pruning (poda)

- Mecanismo de búsqueda para  $T_\alpha$  ( pruning óptimo dado  $\alpha$ ).

*Resultado: para cada  $\alpha$  hay un sub-árbol único  $T_\alpha$  que minimiza  $C_\alpha(T)$ .*

- Eliminar sucesivamente las ramas que producen un aumento mínimo en  $\sum_{m=1}^{[T]} n_m Q_m(T)$
- Se colapsa hasta el nodo inicial pero va a través de una sucesión de árboles
- $T_\alpha$  pertenece a esta secuencia. (Breiman et al., 1984)

# Pruning (poda)

## Algoritmo Completo

- 1 Utilizamos particiones recursivas binarias para hacer crecer el árbol
- 2 Para un dado  $\alpha$ , aplicamos *cost complexity pruning* al árbol para obtener la secuencia de los subárboles como  $\alpha$ .
- 3 Utilizamos K-fold cross-validation para elegir  $\alpha$ .
- 4 Tenemos entonces una secuencia de subárboles para distintos valores de  $\alpha$
- 5 Elegimos el  $\alpha$  y el subárbol que tienen el menor error de predicción.

# Ejemplo



photo from <https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/>

# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# Bagging

- ▶ Problema con CART: pocos robustos.
- ▶ Podemos mejorar mucho el rendimiento mediante la agregación
- ▶ Idea: la varianza del promedio es menor que la de una sola predicción.



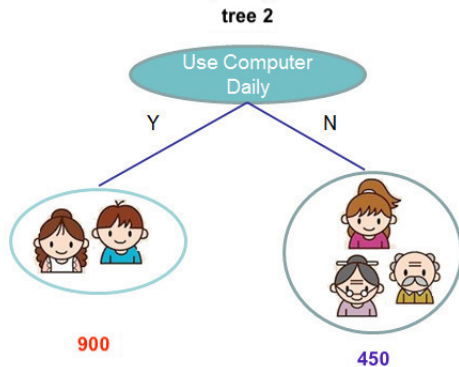
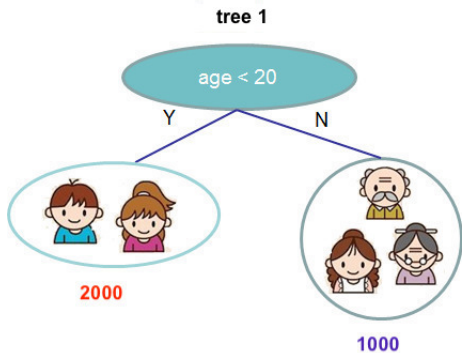
# Bagging

- ▶ Bagging:
  - ▶ Obtenga repetidamente muestras aleatorias  $(X_i^b, Y_i^b)_{i=1}^N$  de la muestra observada (bootstrap).
  - ▶ Para cada muestra, ajuste un árbol de regresión  $\hat{f}^b(x)$
  - ▶ Promedie las muestras de bootstrap

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x) \quad (6)$$

- ▶ Básicamente estamos suavizando las predicciones.

# Bagging



$f(\text{boy}) = (2000 + 900)/2 = 1450$       $f(\text{man}) = (1000 + 450)/2 = 725$

# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# Random Forests

- ▶ Problema con el bagging: si hay un predictor fuerte, diferentes árboles son muy similares entre sí. Si hay alta correlación, ¿está realmente reduciendo la varianza?
- ▶ Bosques (forests): reduce la correlación entre los árboles en el bootstrap.
- ▶ Si hay  $p$  predictores, en cada partición use solo  $m < p$  predictores, elegidos al azar.
- ▶ Bagging es forests con  $m = p$  (usando todo los predictores en cada partición).
- ▶ Tipicamente  $m = \sqrt{p}$

# Ejemplo



photo from <https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/>

# Agenda

## 1 Árboles

- ¿Qué hacen?
- ¿Cómo lo hacen?
- Sobreajuste

## 2 Bagging

- Random Forests

## 3 Boosting

# Boosting: Motivation

- ▶ Problema con CART: varianza alta.
- ▶ Podemos mejorar mucho el rendimiento mediante la agregación
- ▶ El boosting toma esta idea pero lo "encara" de una manera diferente → viene de la computación
- ▶ Va a usar arboles pequeños y a aprender de los errores

# Boosting Trees

- ▶ La idea es aprender de los errores lentamente.
- ▶ Ajustamos un árbol utilizando los errores del modelo.
- ▶ Cada uno de estos árboles puede ser bastante pequeño.
- ▶ Esto permite mejorar lentamente aprendiendo  $f(.)$  en áreas donde no funciona bien.
- ▶ OJO: a diferencia de *bagging*, la construcción de cada árbol depende en gran medida de los árboles que ya han crecido.



# Boosting Trees: Algoritmo

1 Iniciamos fijando  $\hat{f}(x) = 0$  y  $r_i = y_i$  para todos los  $i$  del training set

2 Para  $m = 1, 2, \dots, M$

1 Ajustamos un árbol  $\hat{f}^m$  con  $d$  bifurcaciones ( $d + 1$  hojas)

2 Actualizamos  $\hat{f}(x)$  con una versión "shrunk" del nuevo árbol

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^m(x) \quad (7)$$

3 Actualizamos los residuales

$$r_i \leftarrow r_i - \lambda \hat{f}^m(x) \quad (8)$$

3 El modelo final es

$$\hat{f}_{boost} = \sum_{m=1}^M \lambda \hat{f}^m(x) \quad (9)$$

# Boosting Trees: Iteraciones

- ▶ Los hiperparámetros a fijar son
  - ▶  $\lambda$  la tasa a la que aprende, los valores típicos son 0.01 o 0.001
  - ▶ El tamaño del árbol. Árboles pocos profundos funcionan bien.
  - ▶ El número de iteraciones (M) a usar?

# Boosting Trees: Iteraciones

- ▶ Cuantas iteraciones ( $M$ ) usar?
  - ▶ Cada iteración generalmente reduce el error de ajuste, de modo que para  $M$  lo suficientemente grande este error puede hacerse arbitrariamente pequeño (sesgo se va a cero).
  - ▶ Sin embargo, ajustar demasiado bien los datos de entrenamiento puede llevar a overfit (sobreajuste)
  - ▶ Por lo tanto, hay un número óptimo  $M^*$  que minimiza el error fuera de muestra
  - ▶ Una forma conveniente de encontrar  $M^*$  con validación cruzada

# Example



photo from <https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/>