

# Tree-Based Methods

Big Data y Machine Learning para Economía Aplicada

Ignacio Sarmiento-Barbieri

Universidad de los Andes

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Recap

- Queremos predecir  $y$  en función de observables ( $\mathbf{x}_i$ )

$$y = f(\mathbf{x}_i) + u \quad (1)$$

- donde la estimación de  $f$  implica la que mejor generalize (prediga mejor fuera de muestra):

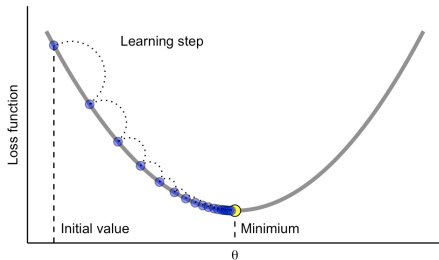
$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \sum_{i=1}^n L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \quad (2)$$

# Recap

- Por ejemplo si  $L$  es la perdida cuadrática y  $f(x_i) = \mathbf{x}_i\beta$  el problema se resume a encontrar los  $\beta$ .

$$\hat{\beta} = \underset{\tilde{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^n (y_i - \mathbf{x}_i\beta)^2 \right\} \quad (3)$$

- para estimar los  $\beta$  usamos en ML gradiente descendiente



Source: Boehmke, B., & Greenwell, B. (2019)

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

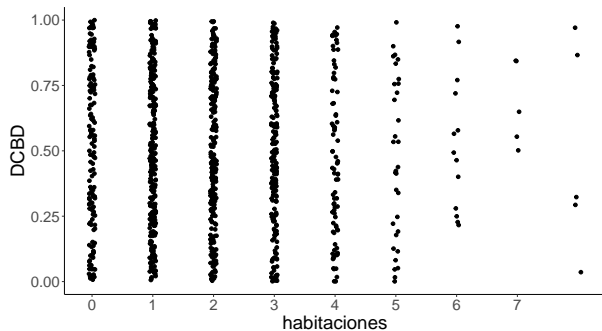
## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Motivación

- Queremos predecir:

$$\text{Precio} = f(\text{habitaciones}, \text{ubicación}) \quad (4)$$





# Árboles

- Podemos asumir  $f$  es lineal

$$f(\mathbf{x}_i) = \beta_0 + \beta_1 \text{Habitaciones}_i + \beta_2 \text{DCBD}_i + u_i \quad (5)$$

# Árboles

- Podemos asumir  $f$  es lineal

$$f(\mathbf{x}_i) = \beta_0 + \beta_1 \text{Habitaciones}_i + \beta_2 \text{DCBD}_i + u_i \quad (5)$$

- o ajustar un modelo flexible e interpretable como son los arboles

$$f(x_i; \{R_j, \gamma_j\}_1^J) = T(x_i; \{R_j, \gamma_j\}_1^J) \quad (6)$$

- donde

$$T(x_i; \{R_j, \gamma_j\}_1^J) = \sum_{j=1}^J \gamma_j I(x \in R_j) \quad (7)$$

# Árboles

- El problema de optimización

$$\hat{f} = \operatorname{argmin}_f \left\{ \sum_{i=1}^n L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \quad (8)$$

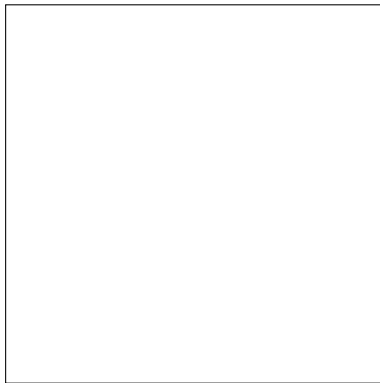
- es ahora

$$\{\hat{R}_j, \hat{\gamma}_j\}_1^J = \operatorname{argmin}_{\{R_j, \gamma_j\}_1^J} \left\{ \sum_{i=1}^n L(y_i, T(\mathbf{x}_i; \{R_j, \gamma_j\}_1^J)) \right\} \quad (9)$$

# Árboles

“Recursive binary splitting”

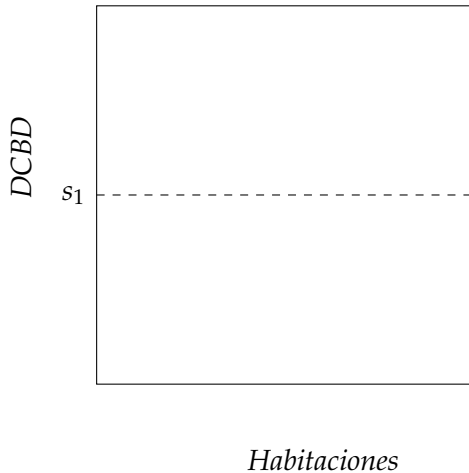
*DCBD*



*Habitaciones*

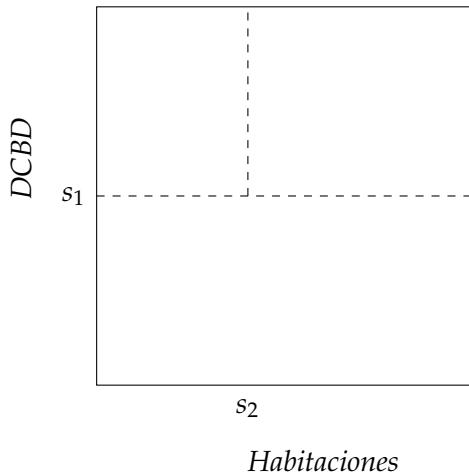
# Árboles

“Recursive binary splitting”

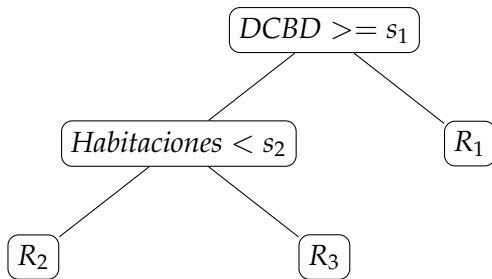


# Árboles

“Recursive binary splitting”



# Árboles



# Ejemplo



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



# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

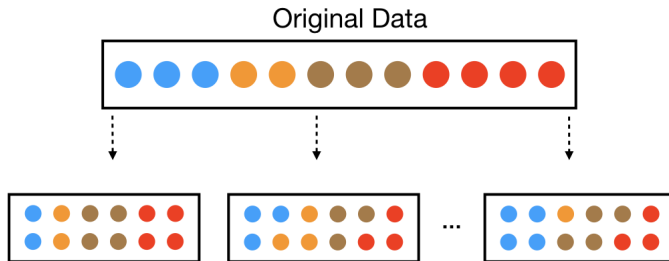
# Bagging

- ▶ Problema con CART: pocos robustos.
- ▶ Idea: podemos mejorar mucho el rendimiento mediante la agregación

# Bagging

- ▶ Bagging:
  - ▶ Obtenga muestras aleatorias  $(X_i^b, Y_i^b)_{i=1}^N$  de la muestra observada (bootstrap).

# Bagging



# Bagging

- ▶ Bagging:

- ▶ Obtenga muestras aleatorias  $(X_i^b, Y_i^b)_{i=1}^N$  de la muestra observada (bootstrap).

# Bagging

- ▶ Bagging:
  - ▶ Obtenga 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

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x) \quad (10)$$

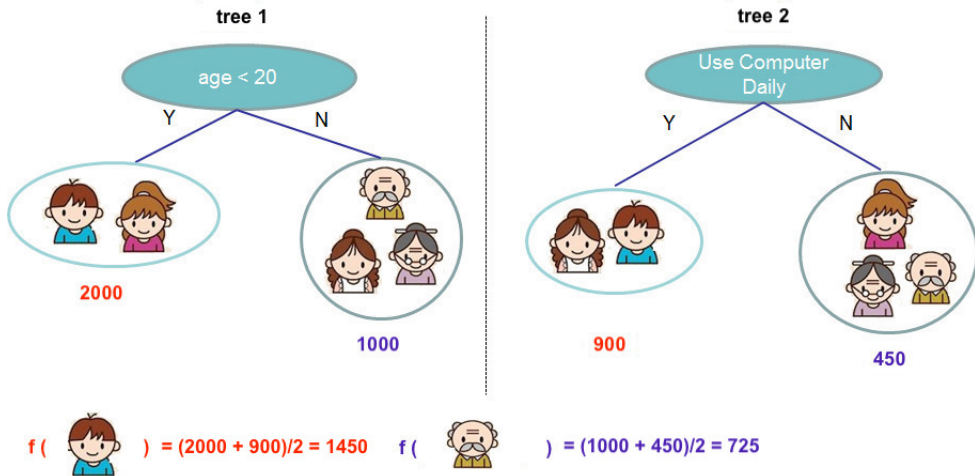
# Bagging

- ▶ Bagging:
  - ▶ Obtenga 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

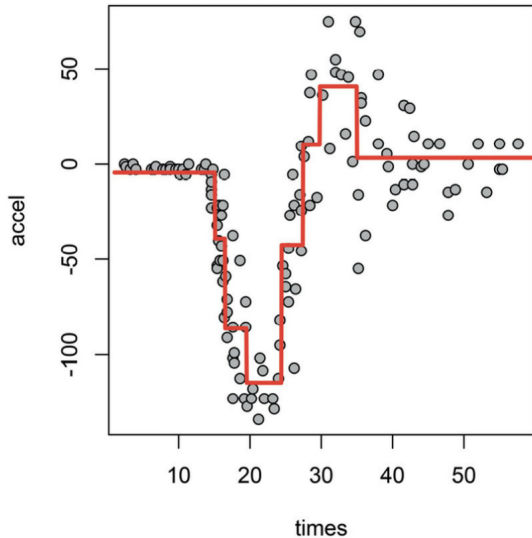
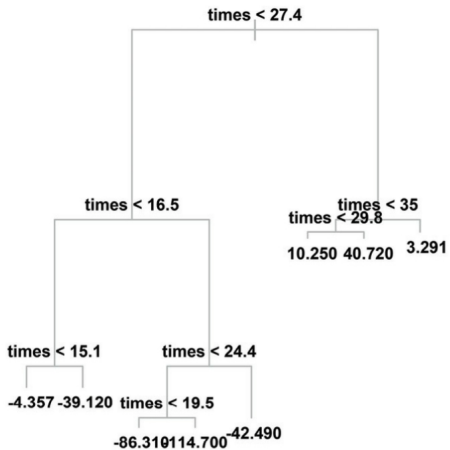
$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x) \quad (10)$$



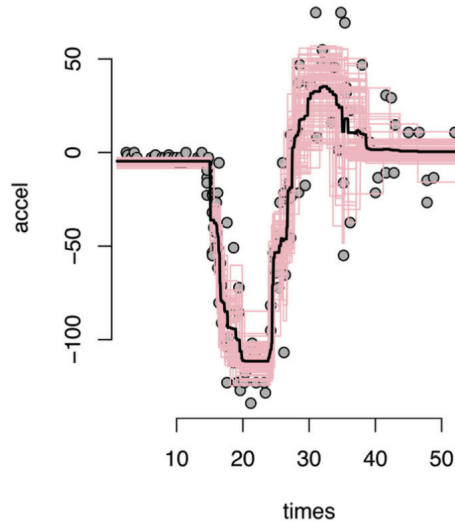
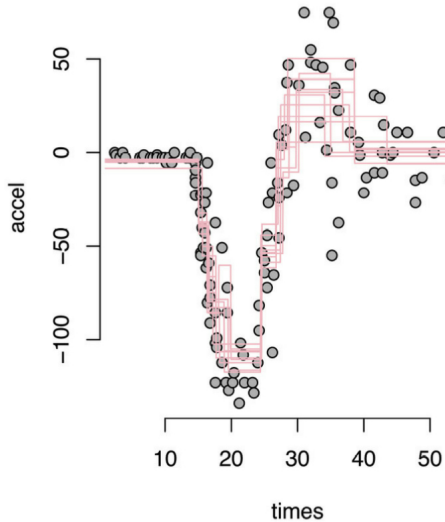
# Bagging



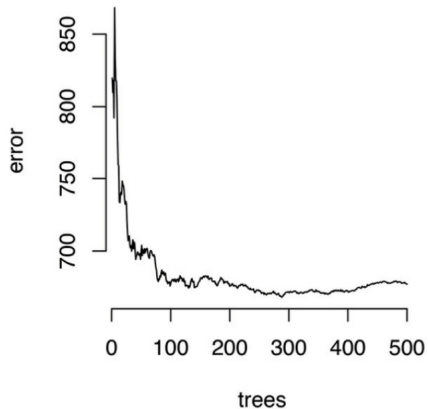
# Bagging



# Bagging



# Bagging



# Ejemplo



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

# Bagging

## Out-of-Bag Error: Intuición

- ▶ En Bagging, entrenamos múltiples modelos sobre muestras bootstrap.
- ▶ Cada muestra contiene  $\sim 63\%$  de los datos originales. (Quiz 1)
- ▶ Estos ejemplos no usados se llaman Out-of-Bag (OOB).

# Bagging

Out-of-Bag Error: Matemáticamente

Sean:

- ▶  $\{(x_i, y_i)\}_{i=1}^n$ : conjunto de entrenamiento.
- ▶  $T_1, \dots, T_B$ : arboles entrenados en cada muestra bootstrap.
- ▶  $\mathcal{B}_i \subset \{1, \dots, B\}$ : índices de modelos donde  $x_i$  no fue usado (OOB).

Entonces, el error OOB es:

$$E_{OOB} = \frac{1}{n} \sum_{i=1}^n L \left( y_i, \frac{1}{|\mathcal{B}_i|} \sum_{b \in \mathcal{B}_i} T_b(x_i) \right)$$

- ▶  $L$  función de pérdida (error cuadrático, 0 – 1, etc.).
- ▶ Estima el **error de generalización, fuera de muestra** sin necesidad de un set de validación.

# Bagging

## Importancia de Variables por Permutación: Intuición

- ▶ En modelos tipo Random Forests, queremos saber:

¿Qué tan importante es cada variable para las predicciones?

- ▶ La idea:

- ▶ Medimos cuánto empeora el desempeño del modelo si rompemos la relación entre una variable y el objetivo.

- ▶ ¿Cómo rompemos esa relación?

- ▶ Permutando los valores de la variable, dejando las otras igual.

- ▶ Si la variable es importante, el error debería aumentar.



# Bagging

## Importancia por Permutación: Definición

- ▶ Sea  $E_{\text{OOB}}$ : error del modelo usando los datos OOB.
- ▶ Para cada variable  $j$ :
  - ▶ Permutamos la columna  $x_j$  en los datos OOB.
  - ▶ Calculamos el nuevo error:  $E_{\text{perm}(j)}$ .
- ▶ La importancia de la variable  $j$  es:

$$\text{VI}(j) = E_{\text{perm}(j)} - E_{\text{OOB}}$$

- ▶ Si  $x_j$  es irrelevante, el error no debería cambiar.
- ▶ Si  $x_j$  es clave, el error aumentará.

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# Random Forests

- ▶ Problema con el bagging: si hay un predictor dominante en la muestra de entrenamiento, diferentes árboles son muy similares entre sí.
- ▶ 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).
- ▶  $m$  es un hiper-parámetro,  $m = \sqrt{p}$  es un benchmark

# Ejemplo



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

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

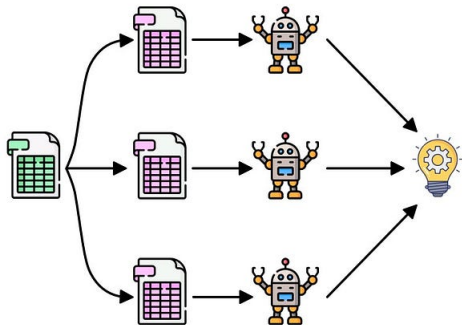
- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# 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 “aprende” secuencialmente

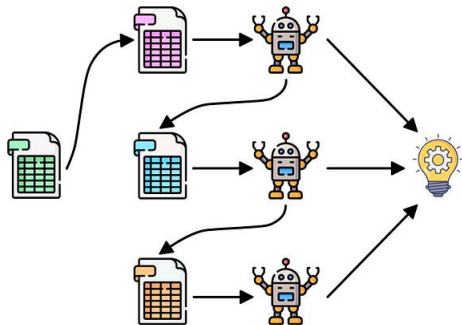
# Boosting: Motivation

## Bagging



## Parallel

## Boosting



## Sequential

# Boosting: Motivation

- ▶ Boosting es una de las ideas de aprendizaje más poderosas introducidas en los últimos veinte años.
- ▶ Originalmente fue diseñado para problemas de clasificación,
- ▶ Pero también puede extenderse a la regresión.



# Boosting: Motivation

- ▶ Intuitivamente lo que hacen es “aprender” de los errores lentamente.
- ▶ Esto lo hace “ajustando” árboles pequeños.
- ▶ Esto permite mejorar lentamente aprendiendo  $f(\cdot)$  en áreas donde no funciona bien.
- ▶ OJO: a diferencia de *bagging*, la construcción de cada árbol depende en gran medida de los árboles anteriores

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# AdaBoost.M1

- ▶ El AdaBoost.M1 es una implementación para clasificación

# AdaBoost.M1

- ▶ El AdaBoost.M1 es una implementación para clasificación
- ▶ Como viene de la computación tiene un lenguaje ligeramente diferente.
- ▶ Vocabulario:
  - ▶  $y \in -1, 1$ ,  $X$  vector de predictores.
  - ▶  $\hat{y} = G(X)$  (clasificador)
  - ▶  $err = \frac{1}{N} \sum_i^N I(y_i \neq G(x_i))$

# AdaBoost.M1

- 1 Comenzamos con ponderadores  $w_i = 1/N$
- 2 Para  $m = 1$  hasta  $M$ :
  - 1 Estimar  $G_m(x)$  usando ponderadores  $w_i$ .
  - 2 Computar el error de predicción

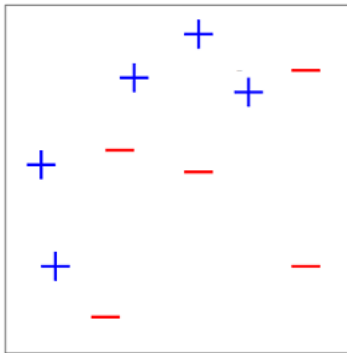
$$err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i} \quad (11)$$

- 3 Obtener  $\alpha_m = \ln \left[ \frac{(1-err_m)}{err_m} \right]$
- 4 Actualizar los ponderadores :  $w_i \leftarrow w_i c_i$

$$c_i = \exp [\alpha_m I(y_i \neq G_m(x_i))] \quad (12)$$

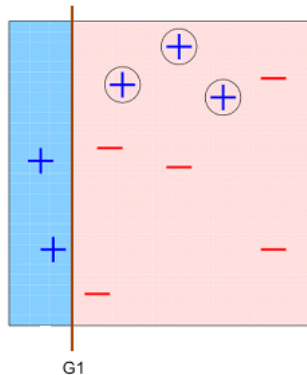
- 3 Resultado:  $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m G_m(x)]$

# AdaBoost.M1



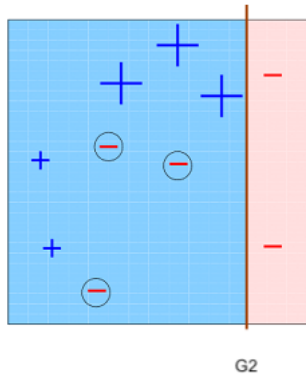
# AdaBoost.M1

## ► Iteración 1



# AdaBoost.M1

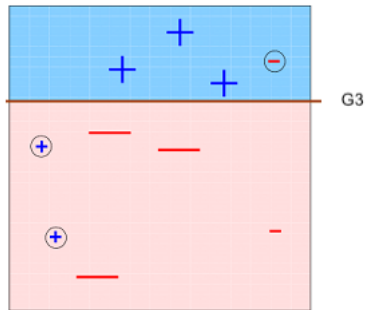
## ► Iteración 2





# AdaBoost.M1

## ► Iteración 3



# AdaBoost.M1

## ► Resultado Final

$$G_{\text{final}} = \text{sign} \left( \alpha_1 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} + \alpha_2 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} + \alpha_3 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} \right) = \begin{array}{|c|c|c|} \hline \text{blue} & + & + \\ \hline \text{blue} & + & + \\ \hline \text{red} & - & - \\ \hline \end{array}$$

The diagram illustrates the final result of the AdaBoost.M1 algorithm. It shows the combination of three weak classifiers, each represented by a square divided into a blue region and a red region by a vertical line. The weights  $\alpha_1$ ,  $\alpha_2$ , and  $\alpha_3$  are applied to these classifiers. The final result is a square divided into a blue region and a red region by a vertical line, with the blue region containing three '+' signs and the red region containing three '-' signs.

# AdaBoost.M1

- 1 Comenzamos con ponderadores  $w_i = 1/N$
- 2 Para  $m = 1$  hasta  $M$ :
  - 1 Estimar  $G_m(x)$  usando ponderadores  $w_i$ .
  - 2 Computar el error de predicción

$$err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i} \quad (13)$$

- 3 Obtener  $\alpha_m = \ln \left[ \frac{(1-err_m)}{err_m} \right]$
- 4 Actualizar los ponderadores :  $w_i \leftarrow w_i c_i$

$$c_i = \exp [\alpha_m I(y_i \neq G_m(x_i))] \quad (14)$$

- 3 Resultado:  $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m G_m(x)]$

# AdaBoost.M1

- ▶  $c_i = \exp[\alpha_m I(y_i \neq G_m(x_i))]$
- ▶ Si fue correctamente predicho,  $c_i = 1$ .
- ▶ En caso contrario,  $c_i = \exp(\alpha_m) = \frac{(1 - \text{err}_m)}{\text{err}_m} > 1$
- ▶ En cada paso el algoritmo da mas importancia relativa a las predicciones incorrectas.
- ▶ Paso final: promedio ponderado de estos pasos

$$G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right] \quad (15)$$

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- **Boosting Regression Trees**
- Gradient Boosting Trees
  - XGBoost

# Boosting Trees

## Intuition

- ▶ Boosting combina muchos modelos débiles (por ejemplo, árboles pequeños), entrenados **secuencialmente**, donde cada nuevo modelo se ajusta para **corregir los errores del modelo anterior**.
  - ▶ Los modelos débiles por sí solos tienen bajo desempeño (alto sesgo).
  - ▶ Boosting reduce el error al combinar varios de estos modelos.
  - ▶ Cada nuevo modelo se entrena sobre los errores (residuos) del conjunto anterior.
  - ▶ El resultado final es un modelo fuerte con mejor capacidad predictiva.

*“No empezamos de cero, ajustamos lo que ya hay.”*

# 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  $\hat{f}^m(x) = 0$  con  $J$  bifurcaciones
  - 2 Actualizamos  $\hat{f}(x)$  con una versión "shrunk" del nuevo árbol

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}^m(x) \quad (16)$$

- 3 Actualizamos los residuales

$$r_i \leftarrow r_i - \hat{f}^m(x) \quad (17)$$

- 3 El modelo final es

$$\hat{f}_{boost} = \sum_{m=1}^M \hat{f}^m(x) \quad (18)$$

# Boosting Trees: Hiperparámetros

- ▶ Cuántas 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



# Boosting Trees: Hiperparámetros

- Podemos utilizar “shrinkage”?  $\Rightarrow$  Yes!
- Se escala la contribución de cada árbol por un factor  $\lambda \in (0, 1)$

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  $\hat{f}^m(x) = 0$  con  $J$  bifurcaciones

2 Actualizamos  $\hat{f}(x)$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^m(x) \quad (19)$$

3 Actualizamos los residuales

$$r_i \leftarrow r_i - \lambda \hat{f}^m(x) \quad (20)$$

3 El modelo final es

$$\hat{f}_{boost} = \sum_{m=1}^M \lambda \hat{f}^m(x) \quad (21)$$

# Boosting Trees: Iteraciones

- ▶ Los otros hiperparámetros a fijar son
  - ▶  $\lambda$  la tasa a la que aprende, los valores típicos son 0.01 o 0.001
  - ▶ El tamaño del árbol.
    - ▶  $4 \leq J \leq 8$  funciona bien
    - ▶  $J = 2$  suele ser insuficiente
    - ▶  $J > 10$  rara vez se utiliza

# Agenda

## 1 Recap

- Problem at hand
- CARTs

## 2 Bagging and Forests

- Bagging
- Random Forests

## 3 Boosting

- AdaBoost.M1
- Boosting Regression Trees
- Gradient Boosting Trees
  - XGBoost

# De Boosting a Gradient Boosting

- ▶ Boosting: modelos secuenciales que corrigen errores del anterior.
- ▶ AdaBoost: ajusta pesos a las observaciones mal clasificadas.
- ▶ ¿Y si en lugar de ajustar pesos, minimizamos directamente una función de pérdida?

*Del ajuste heurístico a la optimización formal.*

# Gradient Boosting Trees

- El objetivo es

$$\hat{f} = \operatorname{argmin}_f \left\{ \sum_{i=1}^n L(y_i, f(\mathbf{x}_i; \Theta)) \right\} \quad (22)$$

- Boosting

$$f(x_i; \{R_j, \gamma_j\}_1^J, M) = \sum_{m=1}^M T(x_i; \Theta_m) \quad (23)$$

- el problema es

$$\hat{f} = \operatorname{argmin}_f \left\{ \sum_{i=1}^n L(y_i, \sum_{m=1}^M T(x_i; \Theta_m)) \right\} \quad (24)$$

# Gradient Boosting Trees

- El problema

$$\hat{f} = \operatorname{argmin}_f \left\{ \sum_{i=1}^n L(y_i, \sum_{m=1}^M T(x_i; \Theta_m)) \right\} \quad (25)$$

- Se puede aproximar por

$$\hat{\Theta}_m = \operatorname{argmin}_{\Theta_m} \left\{ \sum_{i=1}^n L(y_i, f_{m-1} + T(x_i; \Theta_m)) \right\} \quad (26)$$

- Ejemplo para una función de pérdida cuadrática

$$\hat{\Theta}_m = \operatorname{argmin}_{\Theta_m} \left\{ \sum_{i=1}^n (y_i - f_{m-1} - T(x_i; \Theta_m))^2 \right\} \quad (27)$$

# Gradient Boosting Trees

- For **arbitrary** loss functions this “simplified” problem

$$\hat{\Theta}_m = \operatorname{argmin}_{\Theta_m} \left\{ \sum_{i=1}^n L(y_i, f_{m-1} + T(x_i; \Theta_m)) \right\} \quad (28)$$

- is still a tricky problem
- We can solve it by using an implementation of gradient descent

$$f_m = f_{m-1} - \epsilon \sum_{i=1}^n \nabla_{f_{m-1}} L(y_i, f_{m-1}(x)) \quad (29)$$

- How do we implement this search?

# Gradient Boosting Trees

## Algoritmo

1 Inicializar:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

2 Para  $m = 1$  hasta  $M$ :

1 Calcular los pseudo-residuos:

$$r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

2 Ajustar un árbol de regresión a los  $r_{im}$ , obteniendo regiones terminales  $R_{jm}$ ,  $j = 1, \dots, J_m$ .

3 Para cada región  $R_{jm}$ , calcular:

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

4 Actualizar el modelo:

$$f_m(x) = f_{m-1}(x) + \lambda \sum_{j=1}^{J_m} \gamma_{jm} \cdot I(x \in R_{jm})$$

3 El modelo final es:

$$\hat{f}(x) = f_M(x)$$



# XGBoost: Motivation

- ▶ XGBoost is an implementation was specifically engineered for optimal performance and speed.
- ▶ Why talk about XGBoost?
  - ▶ Among the 29 challenge winning solutions published in Kaggle's blog during 2015, 17 solutions used XGBoost.
  - ▶ Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. (The second most popular method, deep neural nets, was used in 11 solutions)
  - ▶ The success of the system was also witnessed in 2015 Data Mining and Knowledge Discovery competition organized by ACM (KDD Cup) , where XGBoost was used by every winning team in the top-10.
  - ▶ Historically, XGBoost has performed quite well for structured, tabular data. But, if you are dealing with non-structured data such as images, neural networks are usually a better option (more on this later)

# XGBoost

- Cual es la innovación? Penaliza la optimización, en cada paso la función objetivo para construir el árbol es

$$\mathcal{L} = \sum_{i=1}^N L(y_i, \sum_{m=1}^M T(x_i; \Theta_m)) + \sum_{m=1}^M \Omega(T(x_i; \Theta_m)) \quad (30)$$

- El segundo termino penaliza la complejidad del modelo,

$$\Omega(.) = \zeta J + \frac{1}{2} \phi \sum_{j=1}^J \gamma_j^2 \quad (31)$$

# Example



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