



Boosting

Álvaro Barbero Jiménez
alvaro.barbero.jimenez@gmail.com

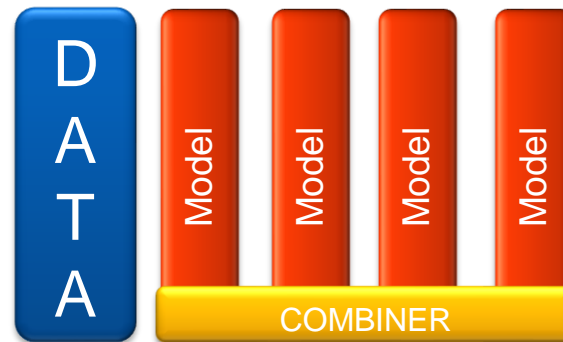
Ensembles secuenciales

En Random Forest el orden en el que se construyen los árboles es irrelevante

- **Ensemble paralelo**

Resultados empíricos han demostrado que en muchos problemas prácticos es más útil construir **ensembles secuenciales**

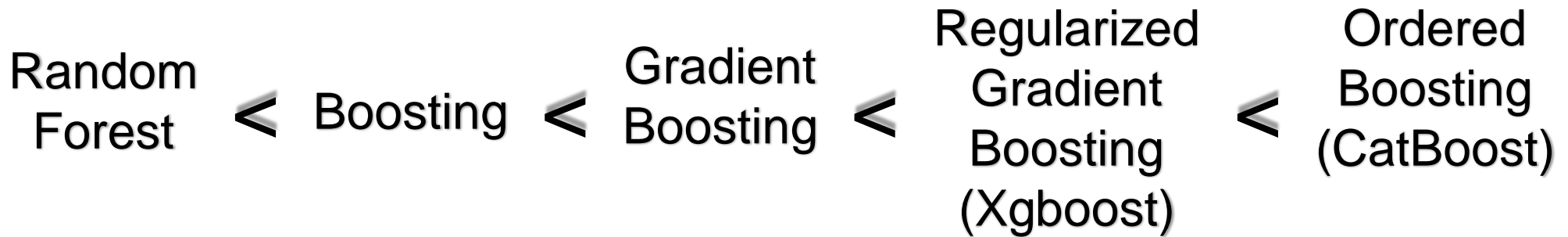
- Cada árbol se construye teniendo en cuenta cómo han funcionado los árboles anteriores



Ensembles secuenciales



Extreme Gradient Boosting (Xgboost) [10] is a very fast and effective machine learning model, which implements algorithms under a gradient-boosting framework, including a generalized linear model and gradient-boosted regression tree. Xgboost is first introduced by [19], it is widely used in Kaggle competitions and is utilized in many winning solutions.



1. Boosting

Boosting

La forma más simple de hacer un ensemble secuencial es mediante **boosting**

Entrenar un árbol de decisión estándar

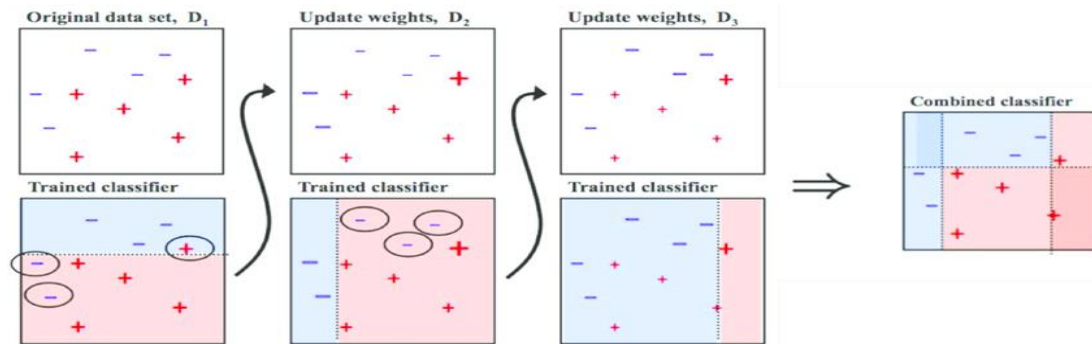
Repetir T veces:

*Calcular los **errores** del ensemble*

*Modificar la **distribución** de muestreo de los datos*

Más error \rightarrow más probabilidad

Entrenar un nuevo árbol con una muestra de los datos



Cada árbol puede tener un peso distinto en la combinación del ensemble

<https://medium.com/swlh/boosting-and-bagging-explained-with-examples-5353a36eb78d?>

Boosting - Adaboost

El algoritmo de boosting más popular es **AdaBoost**.

- Base: considerar una forma particular de medir el error y de combinar los estimadores del ensemble

Motivación: considerar que el error de la función de decisión h de un estimador se mide como:

$$l_{exp}(h|\mathcal{D}) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[e^{-yh(x)}] \quad (\text{con } y \in \{-1,1\})$$

Además considerar que los estimadores se combinan en el ensemble como:

$$H(x) = \sum_{t \in T} \alpha_t h(x)$$

Boosting - Adaboost

Input: Data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
Base learning algorithm \mathfrak{L} ;
Number of learning rounds T .

Process:

1. $\mathcal{D}_1(\boldsymbol{x}) = 1/m$. % Initialize the weight distribution
2. **for** $t = 1, \dots, T$:
3. $h_t = \mathfrak{L}(D, \mathcal{D}_t)$; % Train a classifier h_t from D under distribution \mathcal{D}_t
4. $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$; % Evaluate the error of h_t
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of h_t
7.
$$\begin{aligned} \mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases} \\ &= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t} \end{aligned}$$

% Update the distribution, where
 % Z_t is a normalization factor which
 % enables \mathcal{D}_{t+1} to be a distribution
8. **end**

Output: $\hat{H}(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$

Boosting - Adaboost

¿Por qué esta función de error? $l_{exp}(h|\mathcal{D}) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[e^{-yh(x)}]$

Veamos cómo se descompone l_{exp} para un cierto dato x

$$\begin{aligned} l_{exp}(h|x) &= \mathbb{E}_{y|x}[e^{-yh(x)}] = \int_y e^{-yh(x)} p(y|x) dy \\ &\stackrel{y = \{-1, +1\}}{=} e^{-H(x)} P(y = 1|x) + e^{H(x)} P(y = -1|x) \end{aligned}$$

Esta función es convexa respecto a $H(x)$, por lo que alcanza su mínimo cuando su derivada es 0:

$$\frac{\partial l_{exp}(h|x)}{\partial H(x)} = -e^{-H(x)} P(y = 1|x) + e^{H(x)} P(y = -1|x) = 0$$

Despejando para $H(x)$ obtenemos $H(x) = \frac{1}{2} \ln \frac{P(y = 1|x)}{P(y = -1|x)}$

Boosting - Adaboost

Si aplicamos la función signo

$$\begin{aligned}\text{sign}(H(x)) &= \text{sign}\left(\frac{1}{2} \ln \frac{P(y = 1|x)}{P(y = -1|x)}\right) \\ &= \begin{cases} 1 & P(y = 1|x) > P(y = -1|x) \\ -1 & P(y = 1|x) < P(y = -1|x) \end{cases} \\ &= \arg \max_{y \in \{-1, 1\}} P(y|x)\end{aligned}$$

- ✓ El ensemble $H(x)$ que minimiza la función de error $e^{-yH(x)}$ alcanza el error de Bayes (error óptimo) para cualquier dato x

Boosting - Adaboost

Input: Data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
Base learning algorithm \mathfrak{L} ;
Number of learning rounds T .

Process:

1. $\mathcal{D}_1(\boldsymbol{x}) = 1/m$. % Initialize the weight distribution
 2. **for** $t = 1, \dots, T$:
 3. $h_t = \mathfrak{L}(D, \mathcal{D}_t)$; % Train a classifier h_t from D under distribution \mathcal{D}_t
 4. $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$; % Evaluate the error of h_t
 5. **if** $\epsilon_t > 0.5$ **then break**
 6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of h_t
 7. $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
 $= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$ % Update the distribution, where
 % Z_t is a normalization factor which
 % enables \mathcal{D}_{t+1} to be a distribution
 8. **end**
- Output:** $\hat{H}(\boldsymbol{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\boldsymbol{x}) \right)$

Boosting - Adaboost

El peso óptimo de cada estimador, α_t , puede obtenerse minimizando esta función de error:

$$\begin{aligned} l_{exp}(\alpha_t h_t | \mathcal{D}_t) &= \mathbb{E}_{(x,y) \sim \mathcal{D}_t} [e^{-y\alpha_t h_t(x)}] \\ &= \mathbb{E}_{(x,y) \sim \mathcal{D}_t} [\underbrace{\delta_{y=h_t(x)} e^{-\alpha_t}}_{\text{Patrones acertados}} + \underbrace{\delta_{y \neq h_t(x)} e^{\alpha_t}}_{\text{Patrones fallados}}] \\ &= e^{-\alpha_t} P_{(x,y) \sim \mathcal{D}_t}(y = h_t(x)) + e^{\alpha_t} P_{(x,y) \sim \mathcal{D}_t}(y \neq h_t(x)) \\ &= e^{-\alpha_t} (1 - \epsilon_t) + e^{\alpha_t} \epsilon_t \end{aligned}$$

con $\epsilon = P_{(x,y) \sim \mathcal{D}_t}(h_t(x) \neq y)$. Derivando respecto a α_t e igualando a 0 se obtiene:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Boosting - Adaboost

Input: Data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
Base learning algorithm \mathfrak{L} ;
Number of learning rounds T .

Process:

1. $\mathcal{D}_1(\mathbf{x}) = 1/m.$ % Initialize the weight distribution

2. for $t = 1, \dots, T$:

3. $h_t = \mathcal{L}(D, \mathcal{D}_t)$; % Train a classifier h_t from D under distribution \mathcal{D}_t

4. $\epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}));$ % Evaluate the error of h_t

5. **if** $\epsilon_t > 0.5$ **then break**

6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of h_t

$$7. \quad \mathcal{D}_{t+1}(\mathbf{x}) = \frac{\mathcal{D}_t(\mathbf{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\mathbf{x}) = f(\mathbf{x}) \\ \exp(\alpha_t) & \text{if } h_t(\mathbf{x}) \neq f(\mathbf{x}) \end{cases}$$

$$= \frac{\mathcal{D}_t(\mathbf{x}) \exp(-\alpha_t f(\mathbf{x}) h_t(\mathbf{x}))}{Z_t} \quad \begin{array}{l} \% \text{ Update the distribution, where} \\ \% Z_t \text{ is a normalization factor which} \\ \% \text{ enables } \mathcal{D}_{t+1} \text{ to be a distribution} \end{array}$$

8. end

Output: $\hat{H}(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$

Boosting - Adaboost

Considerar ahora el error al añadir un clasificador al conjunto:

$$\begin{aligned}l_{exp}(H_{t-1} + h_t|\mathcal{D}) &= \mathbb{E}_{(x,y) \sim \mathcal{D}}[e^{-y(H_{t-1}(x) + h_t(x))}] \\&= \mathbb{E}_{(x,y) \sim \mathcal{D}}[e^{-yH_{t-1}(x)} e^{-yh_t(x)}]\end{aligned}$$

Usando la aproximación de Taylor de segundo orden de una exponencial ($e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$) para el término $e^{-yh_t(x)}$

$$\begin{aligned}l_{exp}(H_{t-1} + h_t|\mathcal{D}) &\simeq \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[e^{-yH_{t-1}(x)} \left(1 - yh_t(x) + \frac{y^2 h_t(x)^2}{2} \right) \right] \\&\stackrel{y^2 = 1, h_t^2(x) = 1}{\simeq} \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[e^{-yH_{t-1}(x)} \left(1 - yh_t(x) + \frac{1}{2} \right) \right]\end{aligned}$$

Boosting - Adaboost

Con esto podemos encontrar el mejor clasificador a añadir:

$$h_t(x) = \arg \min_h l_{exp}(H_{t-1} + h | \mathcal{D})$$

$$\simeq \arg \min_h \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[e^{-yH_{t-1}(x)} \left(1 - yh(x) + \frac{1}{2} \right) \right]$$

$e_t^{-yH_{t-1}(x)}$ es constante

$$\equiv \arg \max_h \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[e^{-yH_{t-1}(x)} yh(x) \right]$$

Añadir constante $E_{x,y \sim \mathcal{D}} [e_t^{-yH_{t-1}(x)}]$

$$\equiv \arg \max_h \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\frac{e^{-yH_{t-1}(x)}}{\mathbb{E}_{(x,y) \sim \mathcal{D}} [e^{-yH_{t-1}(x)}]} yh(x) \right]$$

Boosting - Adaboost

Ahora definimos la distribución modificada

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x)e^{-yH_{t-1}(x)}}{\mathbb{E}_{(x,y) \sim \mathcal{D}} [e^{-yH_{t-1}(x)}]}$$

que podemos usar en el resultado anterior para obtener

$$\begin{aligned} h_t(x) &= \arg \max_h \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\frac{e^{-yH_{t-1}(x)}}{\mathbb{E}_{(x,y) \sim \mathcal{D}} [e^{-yH_{t-1}(x)}]} y h(x) \right] \\ &= \arg \max_h \mathbb{E}_{(x,y) \sim \mathcal{D}_t} [y h(x)] \\ &\equiv \arg \max_h \mathbb{E}_{(x,y) \sim \mathcal{D}_t} [\delta_{y=h(x)}] \end{aligned}$$

- ✓ El mejor clasificador a añadir al ensemble es el que maximiza los aciertos sobre la distribución modificada

Boosting - Adaboost

Input: Data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
Base learning algorithm \mathfrak{L} ;
Number of learning rounds T .

Process:

1. $\mathcal{D}_1(\mathbf{x}) = 1/m$. % Initialize the weight distribution
2. **for** $t = 1, \dots, T$:
3. $h_t = \mathfrak{L}(D, \mathcal{D}_t)$; % Train a classifier h_t from D under distribution \mathcal{D}_t
4. $\epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$; % Evaluate the error of h_t
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of h_t
7.
$$\mathcal{D}_{t+1}(\mathbf{x}) = \frac{\mathcal{D}_t(\mathbf{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\mathbf{x}) = f(\mathbf{x}) \\ \exp(\alpha_t) & \text{if } h_t(\mathbf{x}) \neq f(\mathbf{x}) \end{cases}$$
$$= \frac{\mathcal{D}_t(\mathbf{x}) \exp(-\alpha_t f(\mathbf{x}) h_t(\mathbf{x}))}{Z_t}$$
 % Update the distribution, where
% Z_t is a normalization factor which
% enables \mathcal{D}_{t+1} to be a distribution
8. **end**

Output: $\hat{H}(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

Boosting - Adaboost

Por último, la distribución de muestreo puede actualizarse como:

$$\begin{aligned}\mathcal{D}_{t+1}(x) &= \frac{\mathcal{D}(x)e^{-yH_t(x)}}{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_t(x)}]} \\ \text{Separar último modelo añadido } h_t \\ &= \frac{\mathcal{D}(x)e^{-yH_{t-1}(x)}e^{-y\alpha_t h_t(x)}}{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_t(x)}]} \\ \text{Multiplicar/dividir por } \mathbb{E}_{x,y\sim\mathcal{D}}[e^{-yH_{t-1}(x)}] \\ &= \frac{\mathcal{D}(x)e^{-yH_{t-1}(x)}e^{-y\alpha_t h_t(x)}}{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_t(x)}]} \frac{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_{t-1}(x)}]}{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_{t-1}(x)}]} \\ &= \mathcal{D}_t(x)e^{-y\alpha_t h_t(x)} \underbrace{\frac{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_{t-1}(x)}]}{\mathbb{E}_{(x,y)\sim\mathcal{D}}[e^{-yH_t(x)}]}}_{\text{Renormalización}}\end{aligned}$$

lo que produce la regla de actualización de AdaBoost.

2. Gradient Boosting

Gradient Boosting

Gradient Boosting es una generalización de Boosting que trata de encontrar el mejor modelo F^* para una función de error dada L

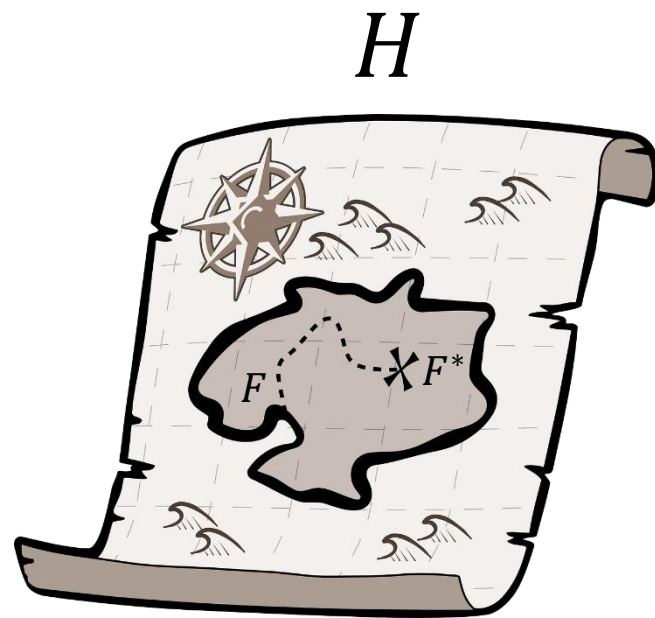
$$F^* = \arg \min_{F \in \mathcal{H}} \Phi(F) = \arg \min_{F \in \mathcal{H}} \mathbb{E}_{(x,y)} [L(y, F(x))]$$

Φ = esperanza de la función de error L
sobre la distribución de los datos

H = familia de funciones (modelos) a la que
limitar la búsqueda

☹ Encontrar el mínimo anterior es MUY difícil

→ Se puede utilizar una estrategia iterativa



Gradient Boosting

Descenso por gradiente para encontrar el mejor modelo:

F_0 = cualquier función con la que empezar (modelo simple)

for $m = 1 \dots M$:

$$F_m = F_{m-1} - \rho_{m-1} \nabla_F \Phi(F_{m-1})$$

donde los pasos de avance ρ_{m-1} pueden encontrarse por búsqueda lineal

$$\nabla_F \Phi(F_{m-1})$$

Gradiente de
un funcional

Respecto a
una función

Evaluado en
otra función

Funcional





KEEP
CALM
AND
CARRY
ON



Gradient Boosting

En la realidad solo tenemos una cantidad limitada de datos (N) para entrenar el modelo.

$$F^* = \arg \min_{F \in \mathcal{H}} \Phi(F) = \arg \min_{F \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N [L(y_i, F(x_i))]$$

con Φ = media de la función de error L sobre los datos

Puede aproximarse la derivada $\nabla_F \Phi(F)$ como el vector de **derivadas parciales** sobre cada uno de los datos

$$\nabla_F \Phi(F) \simeq \begin{bmatrix} \frac{\partial L(y_1, F(x_1))}{\partial F(x_1)} \\ \vdots \\ \frac{\partial L(y_N, F(x_N))}{\partial F(x_N)} \end{bmatrix} = \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_n) \end{bmatrix}$$

donde $g(x_i) = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$ son los **pseudo-residuos** de los datos de entrenamiento $(x_1, y_1), \dots, (x_n, y_n)$

Gradient Boosting

Recordemos: nuestro objetivo es encontrar el mejor modelo

$$F^* = \arg \min_{F \in \mathcal{H}} \Phi(F) = \arg \min_{F \in \mathcal{H}} \mathbb{E}_{(x,y)} [L(y, F(x))]$$

donde H es una cierta **familia de modelos** dada, ej. redes neuronales.

Obs: al realizar actualizaciones del modelo en la forma

$$F_m = F_{m-1} - \rho_{m-1} \nabla_F \Phi(F_{m-1})$$

¿Cómo aseguramos que al sumar un modelo (función gradiente) a otro (F_{m-1}) **nos mantenemos en la misma familia** de funciones?

Gradient Boosting

Una familia que encaja bien en el marco de Gradient Boosting son los **ensembles**:

$$\mathcal{H} = \left\{ \sum_{m=1}^M \beta_m h(a_m) \right\}$$

para cualquier combinación de número de estimadores M , pesos de los estimadores en el ensemble β_m y parámetros de cada estimador a_m .

Si usamos esta familia estaremos construyendo un ensemble paso a paso, **cada paso de gradiente también es un estimador**, $\beta_m h(a_m)$, **que añadimos al conjunto**.

Gradient Boosting

Todo esto significa que lo que estamos haciendo realmente en cada paso de gradient boosting es **actualizar** el ensemble:

$$\sum_{i=1}^m \beta_i h(a_i) = \sum_{i=1}^{m-1} \beta_i h(a_i) - \rho_{m-1} \hat{\beta} h(\hat{a})$$

Donde el nuevo estimador que añadimos, $\hat{\beta} h(\hat{a})$, se calcula como el estimador que para los datos de entrenamiento genera **salidas similares al negativo de los pseudo-residuos** (reducción del error del ensemble):

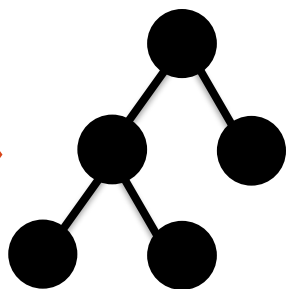
$$(\hat{\beta}, \hat{a}) = \arg \min_{\beta, a} \sum_{i=1}^N (-g_i - \beta h(x_i; a))^2$$

Gradient Boosting

Ejemplo con función de error cuadrático $L(y, \hat{y}) = 1/2 (y - \hat{y})^2$.

Los pseudo-residuos se calcularían como $g = \frac{\partial L(y, F(x))}{\partial F} = y - F(x)$

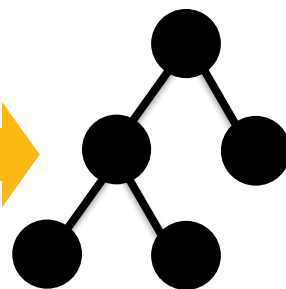
x_1	x_2	y
3	-2	1
7	3	-1
4	2	-1
-1	0	1



x_1	x_2	g
3	-2	0.5
7	3	-1
4	2	0.3
-1	0	-0.2



y	F_1	g
1	0.5	0.5
-1	0	-1
-1	-1.3	0.3
1	1.2	-0.2



x_1	x_2	g
3	-2	0.2
7	3	-0.5
4	2	0.2
-1	0	-0.1



y	F_2	g
1	0.8	0.2
-1	-0.5	-0.5
-1	-1.2	0.2
1	1.1	-0.1

...

Gradient Boosting

Algorithm 1: Gradient_Boost

```
1  $F_0(\mathbf{x}) = \arg \min_{\rho} \sum_{i=1}^N L(y_i, \rho)$ 
2 For  $m = 1$  to  $M$  do:
3    $\tilde{y}_i = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, N$ 
4    $\mathbf{a}_m = \arg \min_{\mathbf{a}, \beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$ 
5    $\rho_m = \arg \min_{\rho} \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$ 
6    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$ 
7 endFor
end Algorithm
```

La búsqueda del paso de avance ρ_m se hace mediante **optimización lineal**.

Gradient Boosting

Si se elige como función de error $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$

Algorithm 2: LS_Boost

$$F_0(\mathbf{x}) = \bar{y}$$

For $m = 1$ to M do:

$$\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$$

$$(\rho_m, \mathbf{a}_m) = \arg \min_{\mathbf{a}, \rho} \sum_{i=1}^N [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$$

endFor

end Algorithm

Cada nuevo estimador tiene como **target** los **errores del ensemble** hasta el momento.

Gradient Boosting

Otras elecciones de funciones de error:

- **Error absoluto**

$$L(y, F(x)) = |F(x) - y| \quad \frac{\partial L(y, F(x))}{\partial F(x)} = \text{sign}(F(x) - y)$$

- **Huber loss**

$$L(y, F(x)) = \begin{cases} \frac{1}{2}(F(x) - y)^2 & |F(x) - y| \leq \delta \\ \delta(|F(x) - y| - \frac{\delta}{2}) & |F(x) - y| > \delta \end{cases}$$

$$\frac{\partial L(y, F(x))}{\partial F(x)} = \begin{cases} F(x) - y & |F(x) - y| \leq \delta \\ \delta \text{sign}(F(x) - y) & |F(x) - y| > \delta \end{cases}$$

3. Extreme Gradient Boosting (XGB)

Extreme Gradient Boosting

Extreme Gradient Boosting es un caso particular de Gradient Boosting con **regularización** y muy especializado para construir **ensembles de árboles**.

Buscamos el mejor modelo que minimiza una función de **error + regularización**:

$$F^* = \arg \min_{F \in \mathcal{H}} \Phi(F) = \arg \min_{F \in \mathcal{H}} \mathbb{E}_{(x,y)} [L(y, F(x))] + \Omega(F)$$

con Ω alguna función que penaliza la complejidad del modelo F .

Considerando la familia H como la de los **ensembles**:

$$\min_a \mathbb{E}_{(x,y)} \left[L \left(y, \sum_{m=1}^M f(x; a_m) \right) \right] + \sum_{m=1}^M \Omega(f(a_m))$$

Extreme Gradient Boosting

Considerar ahora un **conjunto de datos limitado**:

$$\min_a \sum_{i=1}^N \left[L \left(y_i, \sum_{m=1}^M f(x_i; a_m) \right) \right] + \sum_{m=1}^M \Omega(f(a_m))$$

Considerar también que hasta el momento se han construido $t-1$ árboles. La mejor elección para el **siguiente árbol** será:

$$\min_{a_t} \sum_{i=1}^N \left[L \left(y_i, \sum_{m=1}^{t-1} f(x_i; a_m) + f(x_i; a_t) \right) \right] + \sum_{m=1}^{t-1} \Omega(f(a_m)) + \Omega(f(a_t))$$

que eliminando constantes y definiendo \hat{y}^{t-1} como la predicción del ensemble hasta ahora se convierte en:

$$\min_{a_t} \sum_{i=1}^N \left[L \left(y_i, \hat{y}_i^{t-1} + f(x_i; a_t) \right) \right] + \Omega(f(a_t))$$


Extreme Gradient Boosting

$$\min_{a_t} \sum_{i=1}^N [L(y_i, \hat{y}_i^{t-1} + f(x_i; a_t))] + \Omega(f(a_t))$$

Para facilitar la minimización cambiamos el objetivo por su **aproximación de Taylor de segundo orden** en torno a \hat{y}^{t-1}

$$\min_{a_t} \sum_{i=1}^N \left[L(y_i, \hat{y}_i^{t-1}) + g_i f(x_i; a_t) + \frac{1}{2} h_i f^2(x_i; a_t) \right] + \Omega(f(a_t))$$

donde se han definido las primeras y segundas derivadas

 $g_i = \frac{\partial L(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} \quad h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{t-1})}{\partial^2 \hat{y}_i^{t-1}}$

¡Pseudo-residuos!

Eliminando constantes nos queda:

$$\min_{a_t} \sum_{i=1}^N \left[g_i f(x_i; a_t) + \frac{1}{2} h_i f^2(x_i; a_t) \right] + \Omega(f(a_t))$$

Extreme Gradient Boosting

Como **función de error** podemos usar algunas de las vistas hasta el momento.

Para el **regularizador** conviene primero observar que un árbol $f(a)$ puede definirse alternativamente como:

$$f(x; a) = f(x; w, q) = w_{q(x)}, w \in \mathbb{R}^l, q : \mathbb{R}^d \rightarrow \{1, 2, \dots, l\}$$

donde w son los valores emitidos en las hojas, l es el número de hojas del árbol y q es el mapeo que hace recorrer un dato x por el árbol y devuelve la hoja en la que termina.

Con esto el regularizador se define como:

$$\Omega(f(a)) = \underbrace{\gamma l}_{\text{Penalizar muchas hojas}} + \underbrace{\frac{1}{2} \lambda \sum_{j=1}^l w_j^2}_{\text{Penalizar predicciones con valores grandes}}$$

Extreme Gradient Boosting

Uniendo todo tenemos como objetivo:

$$\min_{a_t} \sum_{i=1}^N \left[g_i f(x_i; a_t) + \frac{1}{2} h_i f^2(x_i; a_t) \right] + \gamma l + \frac{1}{2} \lambda \sum_{j=1}^l w_j^2$$

Representación alternativa de árboles

$$= \min_{w, q} \sum_{i=1}^N \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma l + \frac{1}{2} \lambda \sum_{j=1}^l w_j^2$$

Sumatorio sobre las hojas: I_j índices de los datos que caen en la hoja j

$$= \min_{w, q} \sum_{j=1}^l \left[w_j \sum_{i \in I_j} g_i + \frac{1}{2} w_j^2 \left(\sum_{i \in I_j} h_i + \lambda \right) \right] + \gamma l$$

Agrupar sumatorios de pseudo-residuos

$$= \min_{w, q} \sum_{j=1}^l \left[w_j G_j + \frac{1}{2} w_j^2 (H_j + \lambda) \right] + \gamma l$$

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

Q: ¿cómo encontramos los valores óptimos de w y q ? (Árbol)

Extreme Gradient Boosting

Supone un árbol con **estructura dada** (q). El valor óptimo de sus predicciones en las hojas (w) puede calcularse analíticamente:

$$\min_w \sum_{j=1}^l \left[w_j G_j + \frac{1}{2} w_j^2 (H_j + \lambda) \right] + \gamma l$$

Analíticamente se demuestra que los w óptimos son $w_j^* = -\frac{G_j}{H_j + \lambda}$

Poniendo de vuelta los valores óptimos tenemos que para un árbol con estructura fija su **valor objetivo** es:

$$-\frac{1}{2} \sum_{j=1}^l \frac{G_j^2}{H_j + \lambda} + \gamma l$$

Con esto w está resuelto. ¿Cómo encontramos el árbol cuya estructura q minimiza esto?

Extreme Gradient Boosting

$$-\frac{1}{2} \sum_{j=1}^l \frac{G_j^2}{H_j + \lambda} + \gamma l$$






Dado que nos interesa construir un árbol que minimice esta función, podemos tratarla como una **función de impureza** y realizar el procedimiento habitual de árboles de decisión.

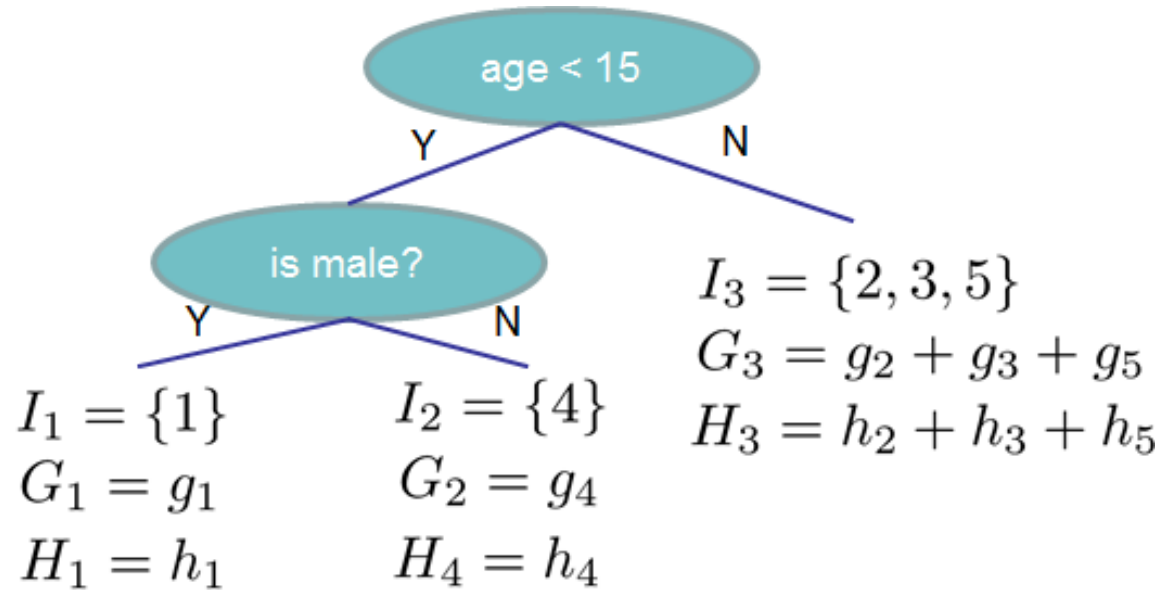
Se trata de una función de impureza que:

- ✓ Minimiza la función de error L del ensemble (ya demostrado)
- ✓ Tiene poda integrada (γl)
- ✓ Proporciona los valores de las hojas (w)

Extreme Gradient Boosting

Instance index gradient statistics

1		g_1, h_1
2		g_2, h_2
3		g_3, h_3
4		g_4, h_4
5		g_5, h_5



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Extreme Gradient Boosting

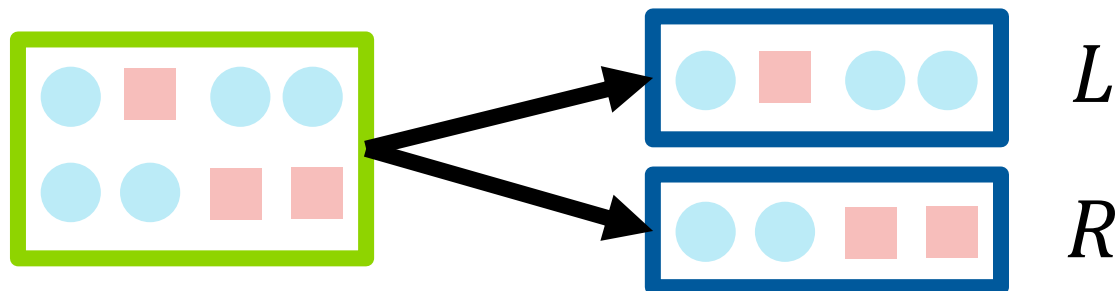
Con la función de impureza presentada, la mejora en impureza en un corte se calcula como

$$-\frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} \right) + 2\gamma$$

$$- \left(\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma \right)$$

Impureza tras corte

Impureza antes del corte

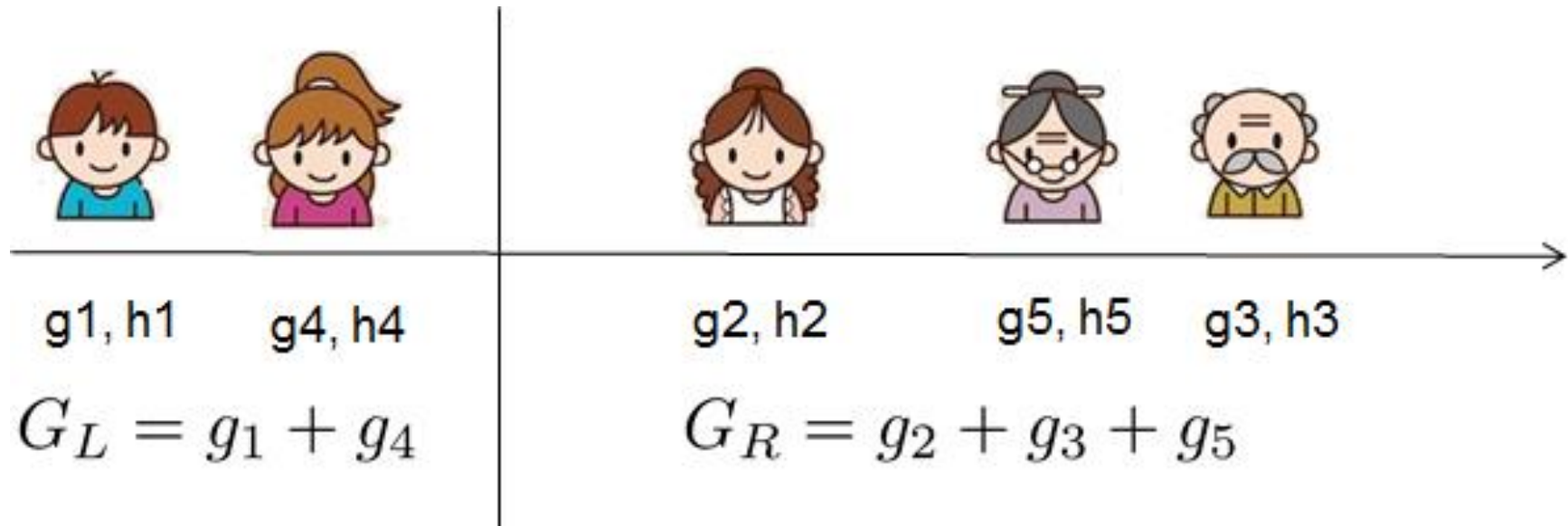


Simplificando la expresión:

$$-\frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) + \gamma$$

Extreme Gradient Boosting

$$-\frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) + \gamma$$



Para calcular de forma eficiente el corte por una variable, se ordenan de menor a mayor todos los datos que hayan llegado al nodo, siguiendo el valor de esa variable, y se consideran sus pseudo-residuos (g_1, h_1) , (g_2, h_2) , \dots , (g_N, h_N) .

Se inicializa $G_L = g_1, H_L = h_1, G_R = \sum_{i=2}^N g_i, H_R = \sum_{i=2}^N h_i$, y se evalúa la mejora en impureza. Para considerar el siguiente corte posible, basta con actualizar $G_L += g_2, H_L += h_2, G_R -= g_2, H_R -= h_2$, y reevaluar la mejora en impureza. El proceso se repite iterativamente con todos los posibles cortes.

Extreme Gradient Boosting

Algoritmo resumido

アルゴリズム10.3 勾配ブースティング木

1. $f_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ となるように初期化する。 $f_0(x) = \gamma$ である。

2. $m = 1$ から M に対して、以下を行う。

(a) 添え字をランダムに入れ替えた上で、 $i = 1, 2, \dots, N$ に対して次を計算する。

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

(b) 回帰木を目的変数 r_{im} に対して回帰木を推定し、その終端領域を $R_{jm} (j = 1, 2, \dots, J_m)$ とする。

(c) $j = 1, 2, \dots, J_m$ に対して次を計算する。

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

(d) $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ のように更新する。

3. $\hat{f}(x) = f_M(x)$ を出力する。

(※筆者注：ここでは回帰木を想定、 $f(x)$ は予測関数、 $L(y, f(x))$ は学習データ上の y に対する損失関数、 $I(x \in R_{jm})$ は指示関数で x が R_{jm} に含まれる場合1を返す)

Extreme Gradient Boosting

Algoritmo resumido

Elegir función de error L , regularizadores λ, γ

$F = f_0 = \text{DecisionTree}(x, y)$

for $t = 1 \dots M$:

for $i = 1 \dots N$:

$$g_i = \frac{\partial L(y_i, \hat{y}^{t-1})}{\partial \hat{y}^{t-1}}, \quad h_i = \frac{\partial^2 L(y_i, \hat{y}^{t-1})}{\partial^2 \hat{y}^{t-1}}$$

$f_t = \text{DecisionTree}(x, y)$ con impureza $-\frac{1}{2} \sum_{j=1}^l \frac{G_j^2}{H_j + \lambda} + \gamma l$

$F += f_t$



4. Ordered Boosting y otras formas de Gradient Boosting

Otros modelos de Gradient Boosting

A comparative analysis of gradient boosting algorithms

Candice Bentéjac¹ · Anna Csörgő² · Gonzalo Martínez-Muñoz³ 

© Springer Nature B.V. 2020

Abstract

The family of gradient boosting algorithms has been recently extended with several interesting proposals (i.e. XGBoost, LightGBM and CatBoost) that focus on both speed and accuracy. XGBoost is a scalable ensemble technique that has demonstrated to be a reliable and efficient machine learning challenge solver. LightGBM is an accurate model focused on providing extremely fast training performance using selective sampling of high gradient instances. CatBoost modifies the computation of gradients to avoid the prediction shift in order to improve the accuracy of the model. This work proposes a practical analysis of how these novel variants of gradient boosting work in terms of training speed, generalization performance and hyper-parameter setup. In addition, a comprehensive comparison between XGBoost, LightGBM, CatBoost, random forests and gradient boosting has been performed using carefully tuned models as well as using their default settings. The results of this comparison indicate that CatBoost obtains the best results in generalization accuracy and AUC in the studied datasets although the differences are small. LightGBM is the fastest of all methods but not the most accurate. Finally, XGBoost places second both in accuracy and in training speed. Finally an extensive analysis of the effect of hyper-parameter tuning in XGBoost, LightGBM and CatBoost is carried out using two novel proposed tools.

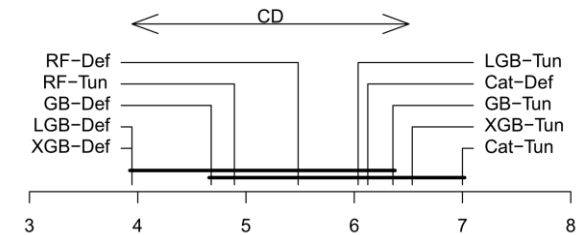


Fig. 1 Average ranks (a higher rank is better) for the tested methods across 28 datasets (Critical difference CD= 2.56)

Otros modelos de Gradient Boosting



Discretización de variables continuas mediante métodos de histogramas
→ mayor velocidad de entrenamiento

Crecimiento de árboles modo “leaf-wise”: hacer crecer la hoja que más reduce la impureza.

Algoritmo GOSS para acelerar aprendizaje sobre datos con error alto.



Compresión de variables categóricas usando estadísticos.

Uso de Oblivious Trees → mayor velocidad de predicción.

Ordered Boosting para reducción del filtrado de target.

CatBoost - <https://tech.yandex.com/catboost/>
Dorogush et al – CatBoost: gradient boosting with categorical features support
LightGBM - <https://github.com/Microsoft/LightGBM>

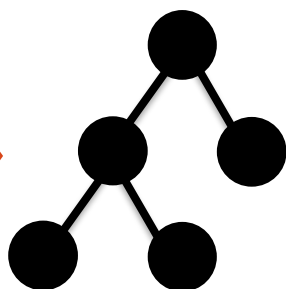
CatBoost

Una de las innovaciones clave de CatBoost es abordar un tipo de **filtrado de target** que otros métodos de boosting ignoran.

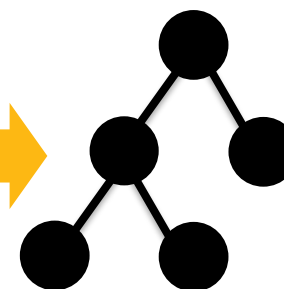


Yandex
CatBoost

x_1	x_2	y
3	-2	1
7	3	-1
4	2	-1
-1	0	1



x_1	x_2	g
3	-2	0.5
7	3	-1
4	2	0.3
-1	0	-0.2

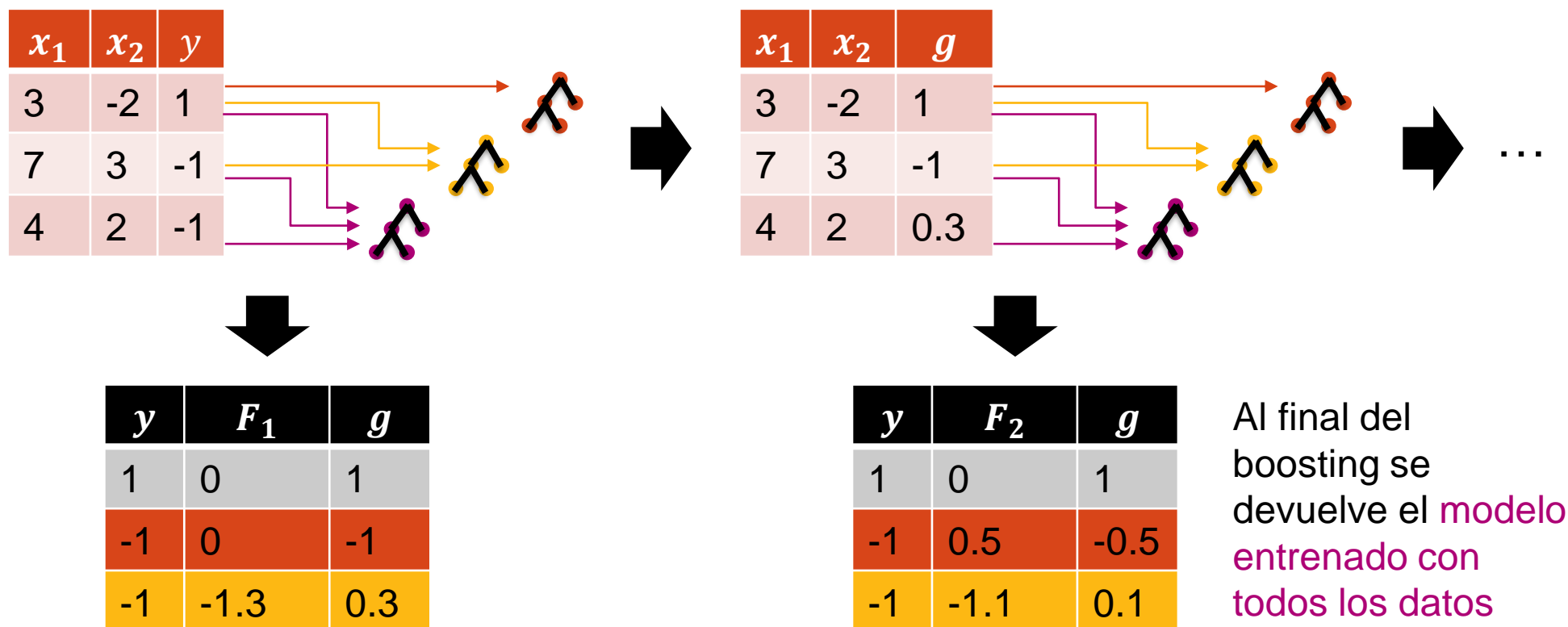


El segundo modelo aprende a corregir los errores (pseudo-residuos g) del primero. Pero estos errores se calculan comparando las predicciones del primer modelo con el target, por lo que **indirectamente contienen información del target**.

Además, si el primer modelo sobreajusta, el segundo modelo aprende a **corregir errores más pequeños** de los que se encontrará realmente en test.

CatBoost

La solución propuesta por Catboost es realizar un **boosting ordenado**: calcular los pseudo-residuos con una serie de modelos de soporte, de forma que el pseudo-residuo para el dato i se calcule con un modelo que solo se ha entrenado con los datos $x[1:i - 1]$

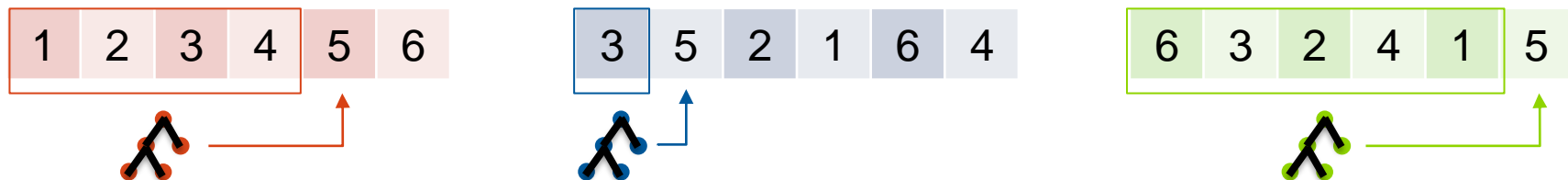


CatBoost – Ordered boosting

El boosting ordenado presenta **dificultades** de implementación práctica:

- Entrenar **N modelos** en cada iteración de boosting es muy **costoso**
- Los pseudo-residuos de los primeros datos se han calculado con modelos que entrenaron con pocos datos, y por tanto son **muy imprecisos** (alta varianza)

El segundo problema puede resolverse realizando **s permutaciones aleatorias** de los datos, $\sigma_1, \sigma_2, \dots, \sigma_s$, y entrenando los N modelos de soporte para cada una de las s permutaciones. Los pseudo-residuos del dato $x[i]$ se calculan promediando los calculados por los s modelos que han entrenado con todos los datos hasta el previo a i en su permutación: $\sigma_k(x)[1:i-1]$



Efectivo para mejorar el cálculo de los pseudo-residuos, ¡pero requiere entrenar Ns modelos!

CatBoost – Optimizaciones

CatBoost implementa varias aproximaciones para calcular de forma eficiente el boosting ordenado incluyendo permutaciones.



Optimización 1: entrenar $\log_2 N$ modelos de soporte en lugar de N

Modelo	Aprende de	Calcula pseudo-residuos para
M_1	$x[1:1]$	$x[2:2]$
M_2	$x[1:2]$	$x[3:4]$
M_3	$x[1:4]$	$x[5:8]$
M_4	$x[1:8]$	$x[9:16]$
M_5	$x[1:16]$	$x[17:32]$
...
M_k	$x[1:2^{k-1}]$	$x[2^{k-1} + 1:2^k]$

Cuando usamos s permutaciones, pasamos de sN modelos a $s \log_2 N$ modelos.

CatBoost – Optimizaciones

Optimización 2: en cada ronda de boosting t seleccionar aleatoriamente una única permutación σ_r , y entrenar un único árbol T_t usando un criterio de impureza que tiene en cuenta el funcionamiento de todos los modelos de soporte de esta permutación.

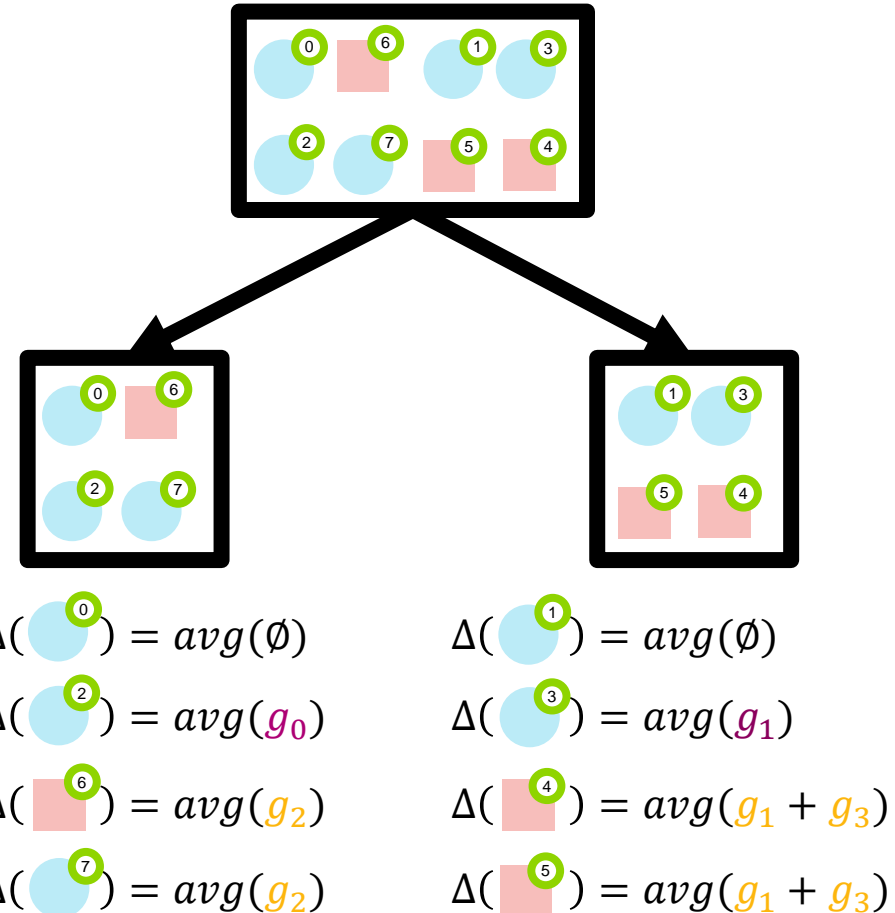
Ejemplo: datos tras permutación σ_r : 
Modelos de soporte { 

Criterio: generar un árbol T_t que maximice la similitud entre los pseudo-residuos actuales de todos los modelos de soporte ($g_{1:1}$, $g_{2:3}$, $g_{4:7}$) y las predicciones de cada modelo de soporte que usará este árbol.

$$\cos(\Delta, [g_{1:1}, g_{2:3}, g_{4:7}])$$

$$\text{con similitud coseno } \cos(x, z) = \frac{x \cdot z}{||x|| ||z||}$$

El árbol T_t generado se añade a todos los modelos de soporte, tanto de la permutación escogida σ_r como de todas las demás permutaciones, adaptando las predicciones de las hojas a los datos que cada modelo tiene permitido utilizar.



CatBoost – Algoritmo final

Definimos $g_{r,i} = \frac{\partial L(y_i, M_{r, \lfloor \log_2(\sigma_r(i)-1) \rfloor}(x_i))}{M_{r, \lfloor \log_2(\sigma_r(i)-1) \rfloor}(x_i)}$ (pseudo-residuo calculado para el dato i , usando el modelo de soporte adecuado según la permutación r)

Inicializar ensemble, $H = \emptyset$

Inicializar modelos de soporte, $M_{r,k}(x_i) = 0 \forall i = 1 \dots N, r = 1 \dots s, k = 1 \dots \log_2 N$

Durante $t = 1, \dots, I$ iteraciones de boosting:

$\sigma_r = \text{random_choice}([\sigma_1, \dots, \sigma_s])$

Calcular pseudo-residuos $g_{r,i} \forall i$

$T_t \leftarrow$ entrenar árbol de CatBoost con datos $(x_{\sigma(i)}, g_{r,i}) \forall i$

$M_{p,k}(x_i) \leftarrow$ añadir a todos los modelos de soporte una copia de T_t con valores de hoja calculados según los datos que ese modelo tiene permitidos para entrenar $(\sigma_p(x)[1:2^{k-1}])$

$H \leftarrow$ añadir T_t al ensemble final, con valores de hoja calculados usando todos los datos de entrenamiento

5. Cierre

Cierre

- ✓ En la actualidad los algoritmos de **ensembles** y en particular **Gradient Boosting y derivados** son los que consiguen **mejores resultados** en la mayoría de los problemas **con estructura de tabla**
- ✓ **Alta paralelización** en CPU y en sistemas distribuidos
- ✓ **Alto rendimiento** en datasets **muy grandes**
- ✓ Explicación de **variables relevantes**
- ✓ En general, son la **mejor elección** a la hora de aproximarse a un problema en formato DataFrame



Bibliografía

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

Ensemble Methods

Foundations and Algorithms



Zhi-Hua Zhou

 CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK



Afi Escuela

© 2021 Afi Escuela. Todos los derechos reservados.