

El Boosting se basa en un procedimiento que combina las salidas de muchos clasificadores "débiles" para generar modelos de predicción de alta performance

El Boosting como método aditivo

El Boosting es una forma de ajustar una expansión aditiva en un conjunto de funciones "base" elementales (los clasificadores débiles)

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m),$$

$\beta_m, m = 1, 2, \dots, M$ Coeficientes de expansión

$b(x; \gamma) \in \mathbb{R}$ Funciones base (clase déb)



$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m), \quad (1)$$

Y parametriza las variables de división y los puntos de división en los nodos internos, y las predicciones en los nodos terminales.

Optimización de la Función de Pérdida (o error/costo)

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^N L \left(y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m) \right). \quad (2)$$

Esta función a optimizar (minimizar en este caso) incluye funciones como parámetros (los árboles) y no se puede optimizarse utilizando métodos de optimización tradicionales en el espacio euclidiano. En su lugar, el modelo se entrena de forma aditiva, resolviendo cada función base individualmente:

$$\min_{\beta, \gamma} \sum_{i=1}^N L(y_i, \beta b(x_i; \gamma)). \quad (3)$$



Modelado Aditivo por Etapas Hacia Adelante (Forward Stagewise Additive Modeling)

La modelización por etapas hacia adelante aproxima la solución de (2) añadiendo secuencialmente nuevas funciones de base a la expansión (3) , **sin ajustar** los parámetros y coeficientes de las que **ya se han añadido** en (1). Veámoslo el algoritmo de este proceso en pseudo-código

Algorithm *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.

2. For $m = 1$ to M :

(a) Compute

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

Diagram annotations for the equation:

- Valor real de y (points to y_i)
- Solución función base anterior (points to $f_{m-1}(x_i)$)
- Solución función base actual (points to $b(x_i; \gamma)$)
- Coeficiente de expansión fc base actual (points to β)

Expansión aditiva

(b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.



Función de Pérdida Cuadrática (Error Cuadrático)

$$L(y, f(x)) = (y - f(x))^2$$

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= \overbrace{(y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2}^{\text{Residuo de la observación } i} \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned} \quad (4)$$

Residuo de la observación i de la función base (árbol) m

$$r_{im} = y_i - f_{m-1}(x_i) \quad (5)$$

Entonces si usamos el error cuadrático medio lo que estaremos ajustando son **los residuos**, o sea la diferencia entre el valor real de la variable objetivo y su predicción actual $f_m(x)$!



$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m).$$



Gradient Boosting Machines

Recordemos el algoritmo general de un modelo aditivo por etapas hacia adelante

Algorithm	<i>Forward Stagewise Additive Modeling.</i>
------------------	---

1. Initialize $f_0(x) = 0$.

2. For $m = 1$ to M :

(a) Compute

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

(b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Para el cálculo de la función de pérdida en cada nodo se aplican sofisticadas técnicas numéricas, en particular la de gradiente descendente. Al estar derivando respecto a una función que es un árbol de decisión, este proceso numérico (que ya veremos en profundidad cuando presentemos los modelos de redes neuronales de perceptrón multicapa) no termina teniendo buena capacidad de generalización. Afortunadamente en el caso del error cuadrático medio su aplicación es sumamente práctica, como la veremos en el algoritmo que sigue.



Algorithm*Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \bar{y}$
2. For $m = 1$ to M :
 - (a) For $i = 1, 2, \dots, N$ compute

$$\tilde{y}_{im} = y_i - f_{m-1}(x_i)$$

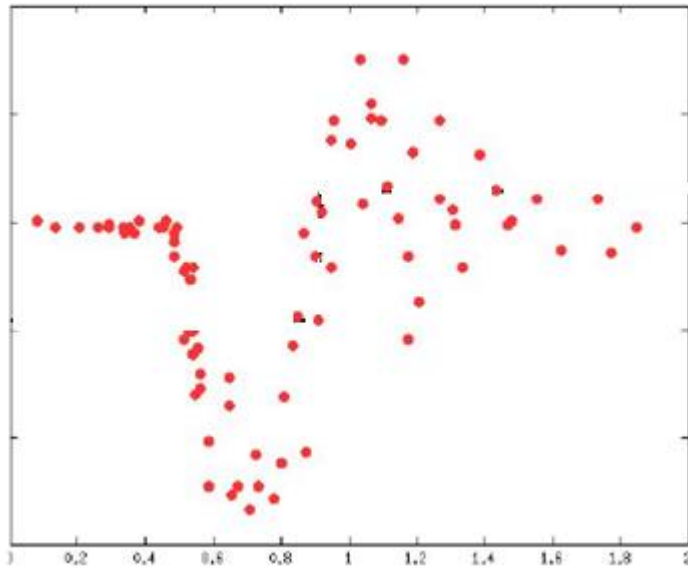
- (b) Fit a regression tree to the targets \tilde{y}_{im} giving terminal regions R_{jm} , $j = 1, 2, \dots, J_m$.
 - (c) For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \bar{y}_{jm} = \text{mean}_{\mathbf{x}_i \in R_{jm}}(\tilde{y}_{im})$$

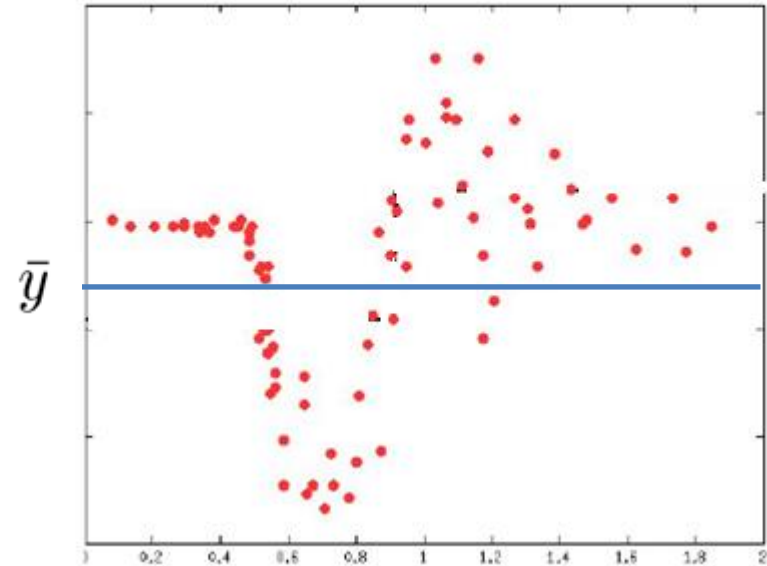
- (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

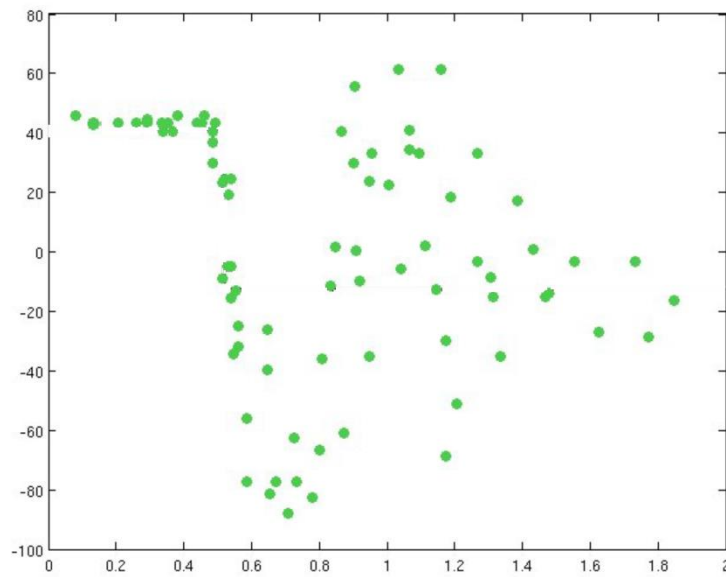


Pasos del Algoritmo en forma gráfica



1. Initialize $f_0(x) = \bar{y}$





2. For $m = 1$ to M :

(a) For $i = 1, 2, \dots, N$ compute

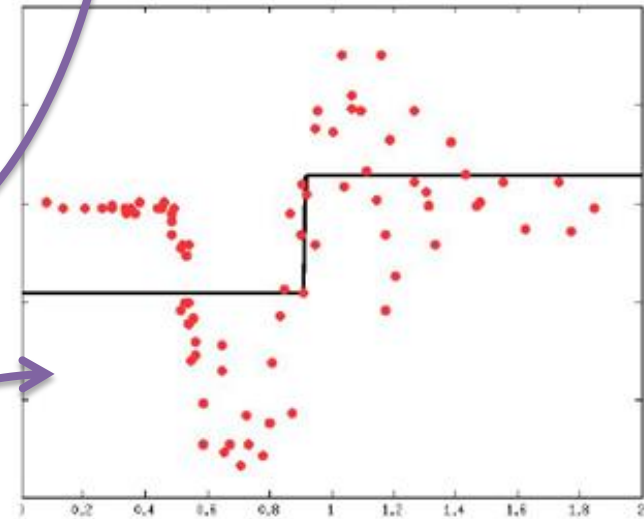
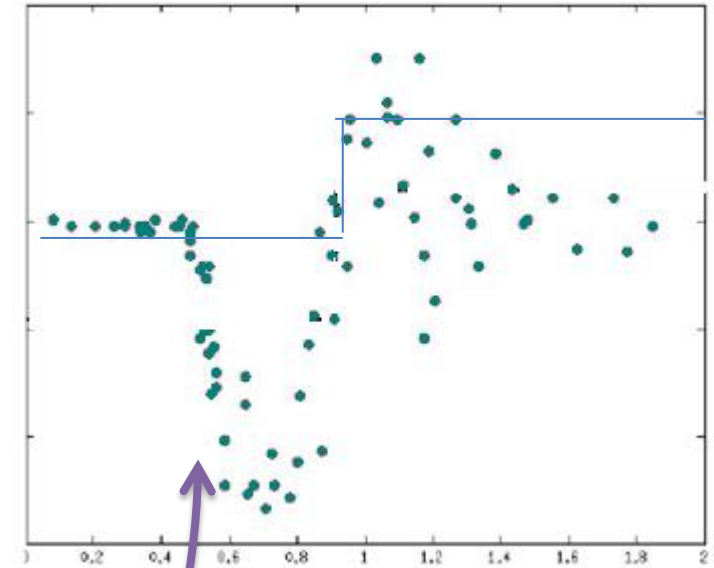
$$\tilde{y}_{im} = y_i - f_{m-1}(x_i)$$

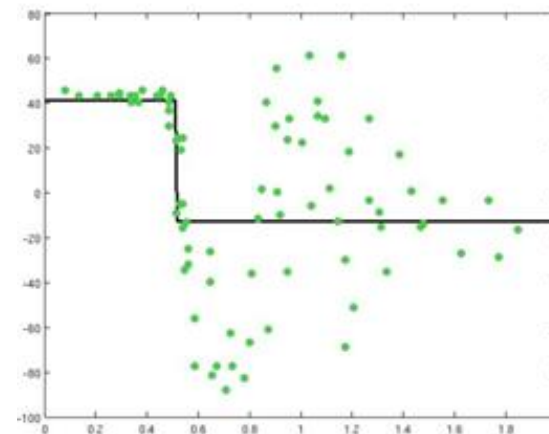
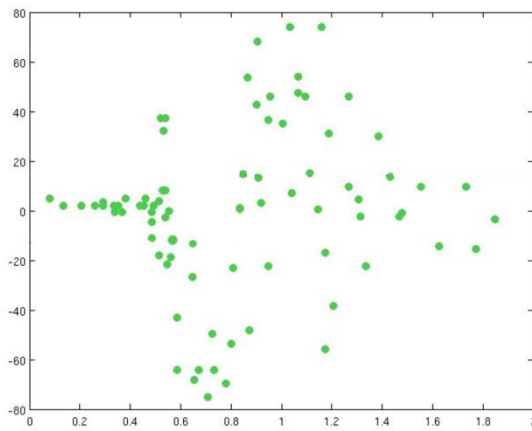
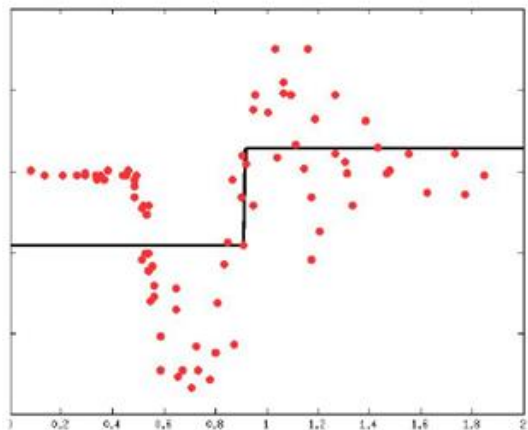
(b) Fit a regression tree to the targets \tilde{y}_{im} giving terminal regions R_{jm} , $j = 1, 2, \dots, J_m$.

(c) For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \bar{y}_{jm} = \text{mean}_{\mathbf{x}_i \in R_{jm}}(\tilde{y}_{im})$$

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





2. For $m = 1$ to M :

(a) For $i = 1, 2, \dots, N$ compute

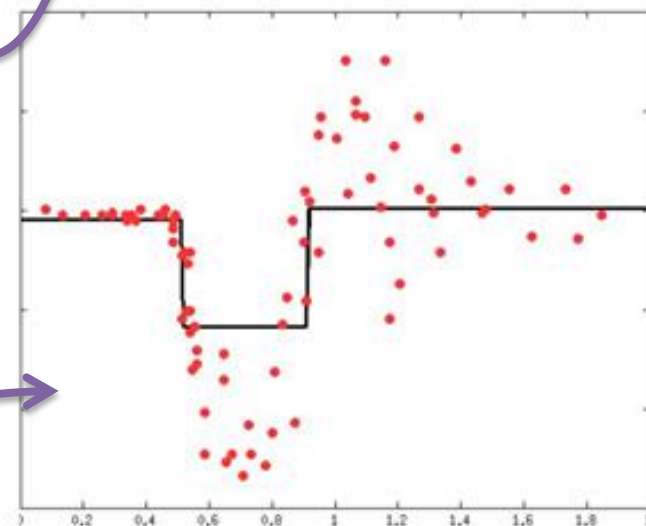
$$\tilde{y}_{im} = y_i - f_{m-1}(x_i)$$

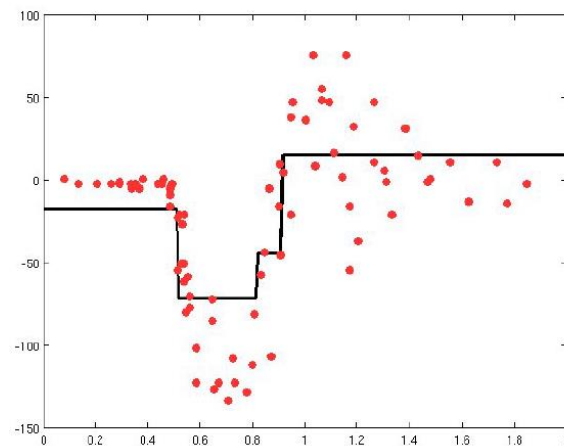
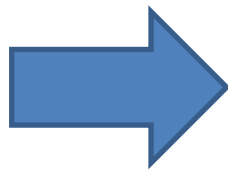
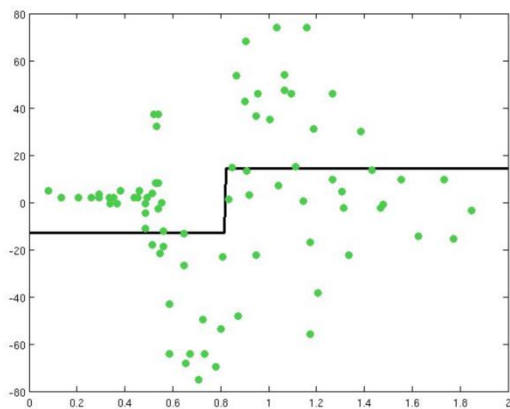
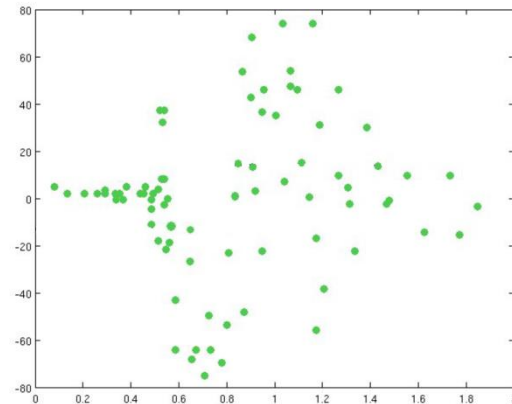
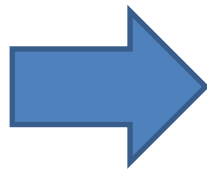
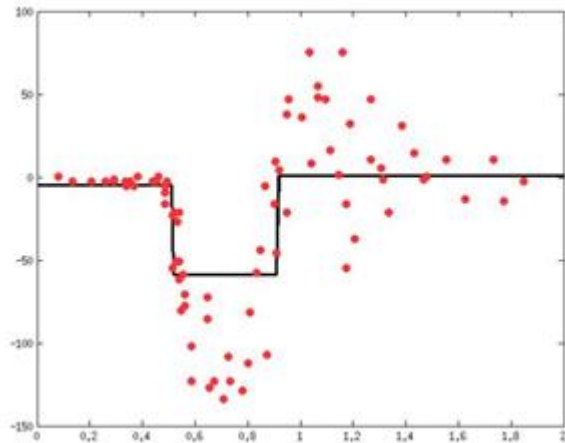
(b) Fit a regression tree to the targets \tilde{y}_{im} giving terminal regions R_{jm} , $j = 1, 2, \dots, J_m$.

(c) For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \bar{y}_{jm} = \text{mean}_{\mathbf{x}_i \in R_{jm}}(\tilde{y}_{im})$$

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





Función de Pérdida Exponencial (Para Clasificación Binaria)

Presentaremos el algoritmo d ADABOOST (Adaptative Boosting), desarrollado por Freund y Shapire, 1997) . Es análogo al algoritmo del modelo aditivo por etapas hacia adelante, salvo que la función de pérdida (al ser de clasificación) es:

$$L(y, f(x)) = \exp(-y f(x))$$

En AdaBoost las funciones base, lo clasificadores débiles, son árboles sumamente simples que contienen una sola bifurcación. En inglés es se los denomina “stumps”. La salida de estos stumps , al ser un clasificador binario, se expresa de la siguiente manera $\{-1,1\}$. La utilización del -1 para la clase negativa (o fracaso, o falso) es para mejorar la performance numérica y matemática del algoritmo.



AdaBoost

El algoritmo AdaBoost aprende **lentamente**. Cada árbol se entrena con los **residuos del anterior**, o sea, con los errores del otro. O sea que en vez de evaluar una función de error con los datos observados y utilizamos los **residuos** del árbol anterior $(y - \hat{y})$, enfocándonos especialmente en las observaciones en las que el árbol anterior **predijo mal**.

Este método se enfoca en las observaciones que son difíciles de predecir correctamente y va encadenando clasificadores simples para al final, ponderándolos, obtener un modelo ensamblado.

Cada uno de estos modelos simples se llama **clasificadores débiles** o “**weak learners**”. Estos clasificadores débiles garantizan la obtención de un resultado de clasificación binaria mejor que el de arrojar una moneda al aire, que es la definición máxima de incertidumbre en este tipo de experimentos binomiales.



Algoritmo AdaBoost (Adaptative Boosting)

Freund and Schapire (1997)

$$Y \in \{-1, 1\} \quad \epsilon_t \doteq \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

Generaremos sucesivos modelos h_1, h_2, \dots, h_t para luego “votar” en un formato de “comité” y obtener la predicción final:

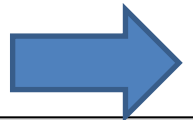
$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

Los α_t serán calculados por el algoritmo y su valor dependerá de la capacidad discriminante del clasificador débil correspondiente.

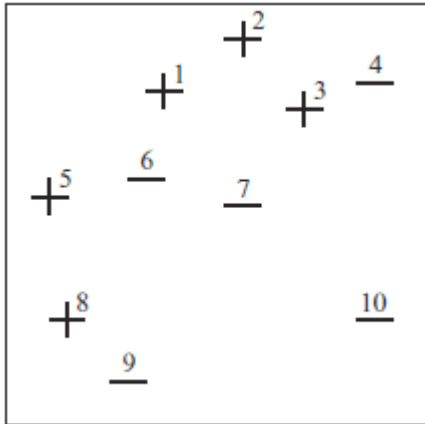
Los pesos iniciales son $D_1(i) = 1/m$

Para cada iteración (o clasificación débil) $t = 1, \dots, T$ se recalculan los pesos en función del resultado de la clasificación_t, incrementando su valor si ésta fue errónea, y reduciéndolo si fue correcta.

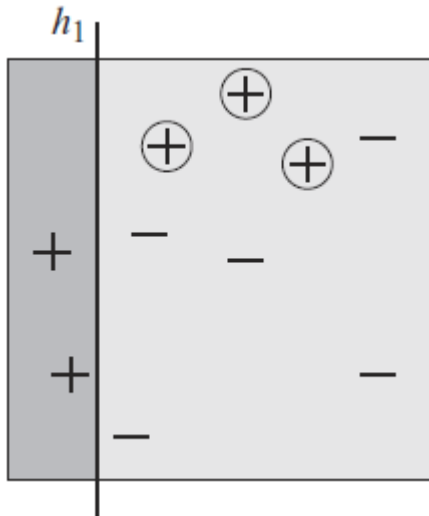
Analicemos el algoritmo de Boosting gráficamente



Dataset original D_1

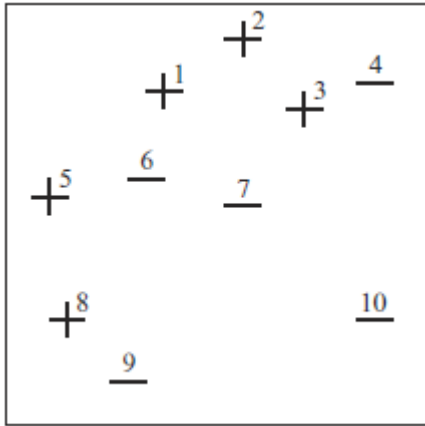


1º Clasificador

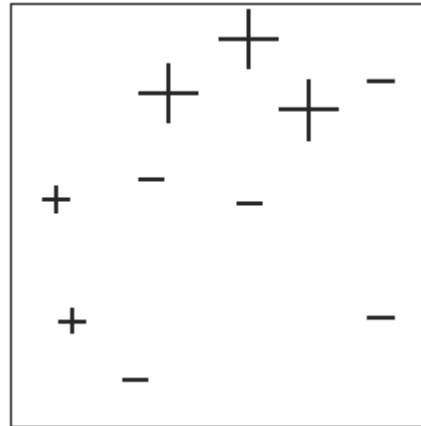


Ahora le asignaremos a cada observación un PESO en función a cómo la clasificó el “weak learner” (mayor a los que clasificó MAL, menor a los que clasificó BIEN)

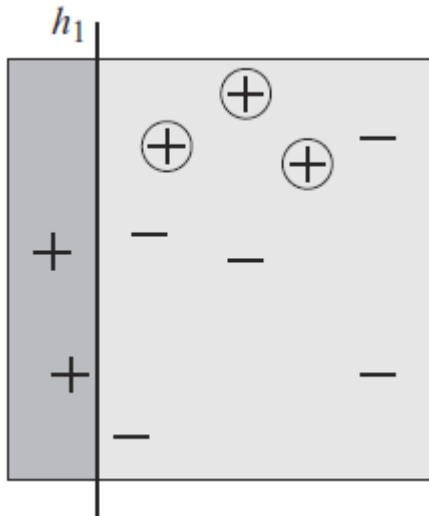
Dataset original D_1



Dataset Modificado (pesos) D_2

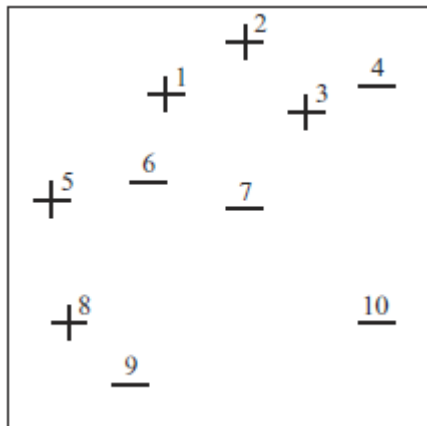


1º Clasificador

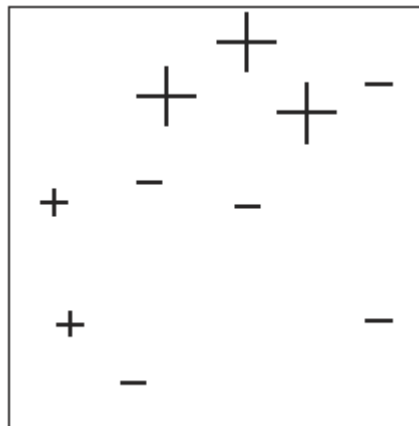


Obtengo es este dataset modificado de pesos D_2
Aumentando el peso de las instancias en las que predijo mal y reduciéndolo en las que predijo bien. Ahora entreno un nuevo clasificador débil (*weak learner*) con este nuevo dataset D_2

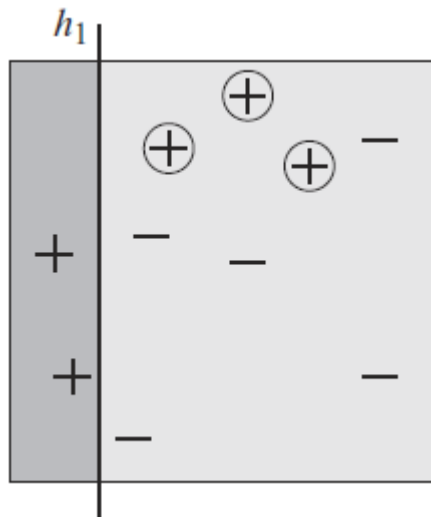
Dataset original D_1



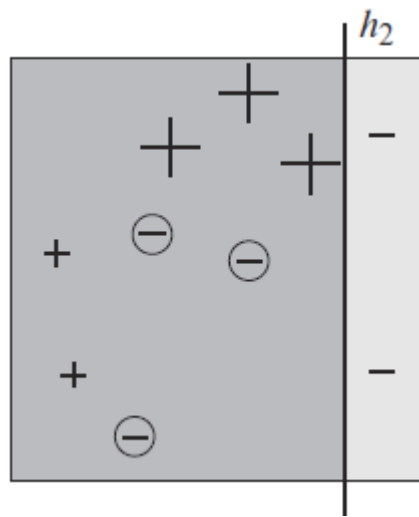
Dataset Modificado (pesos) D_2



1º Clasificador

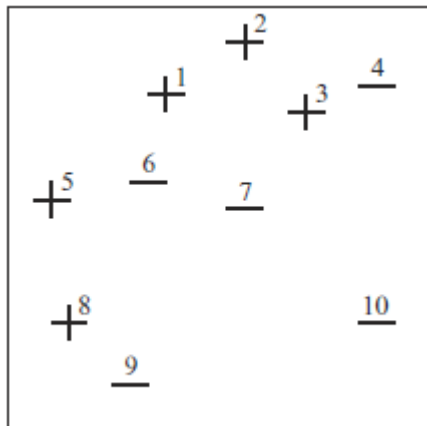


2º Clasificador

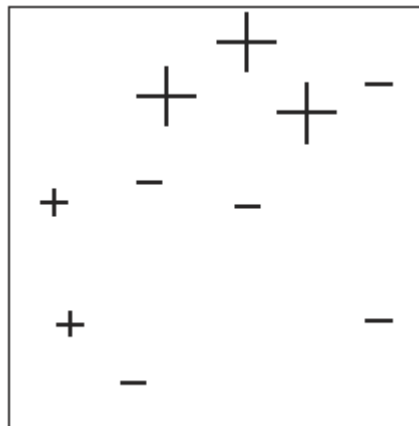


Identificamos los que clasificó mal y generamos un nuevo Dataset Modificado D_3 , en el que nuevamente incrementamos los pesos de las que predijo mal y reducimos las que predijo bien

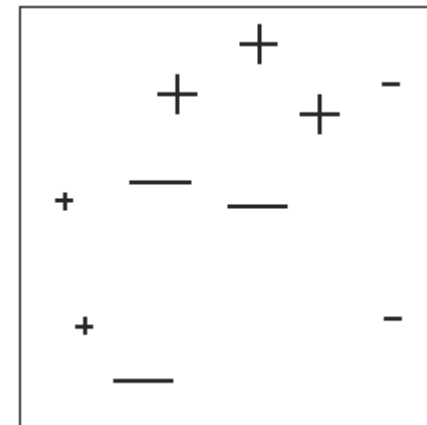
Dataset original D_1



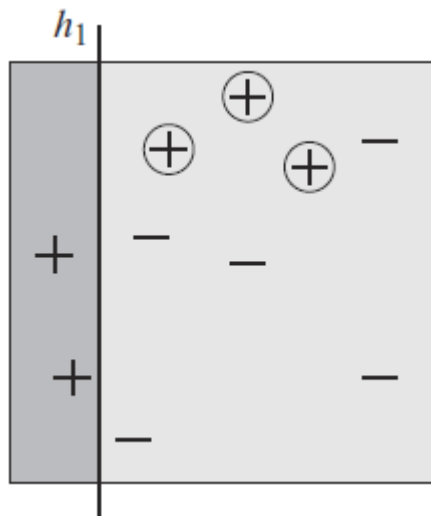
Dataset Modificado (pesos) D_2



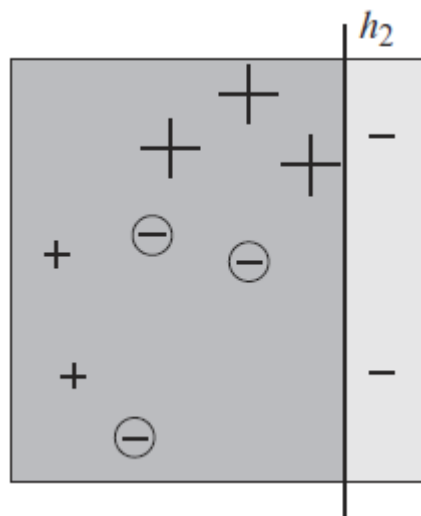
Dataset Modificado (pesos) D_3



1º Clasificador

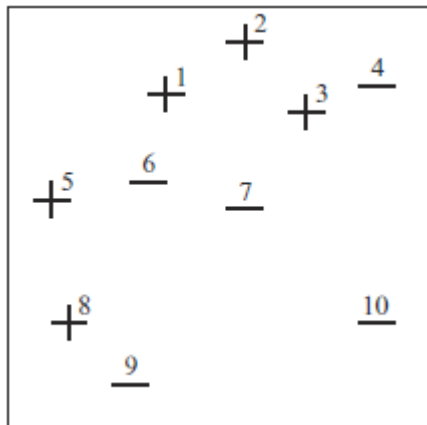


2º Clasificador

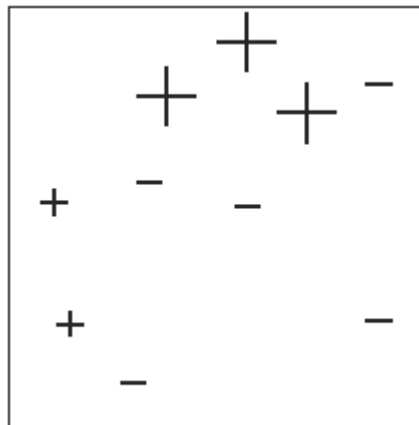


Y ahora entrenaremos un tercer clasificador débil sobre este dataset D_3 , que será:

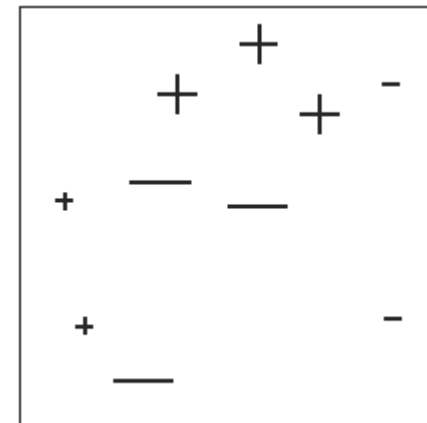
Dataset original D_1



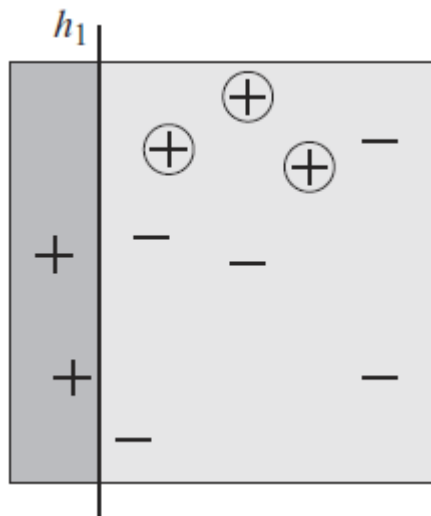
Dataset Modificado (pesos) D_2



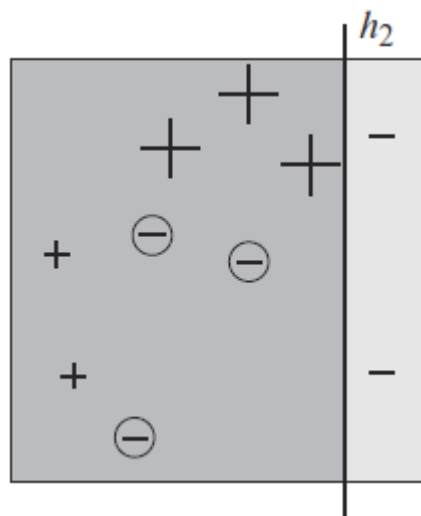
Dataset Modificado (pesos) D_3



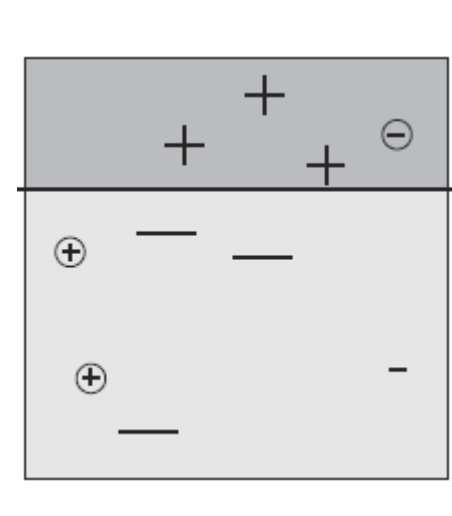
1º Clasificador



2º Clasificador



3º Clasificador



Los pasos y las fórmulas que utiliza el algoritmo AdaBoost son:

Dados $(x_1, y_1), \dots, (x_m, y_m)$ donde $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$

1) Inicializar los pesos $D_i = 1/m$ para $i = 1, 2, \dots, m$

2) De $t = 1$ a T repetir {

i. Entrenar un clase. débil sobre los pesos D_t y calcular hip. débil $h_t : \mathcal{X} \rightarrow \{-1, +1\}$

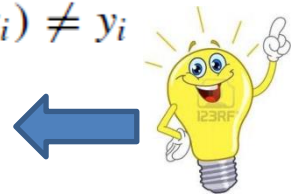
ii. Objetivo: Seleccionar h_t que minimice el error ponderado $\epsilon_t \doteq \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$

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

iv. Actualizar, para $i = 1, 2, \dots, m$ $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Donde Z_t es un factor de normalización
(elegido para que D_{t+1} sea una distribución)



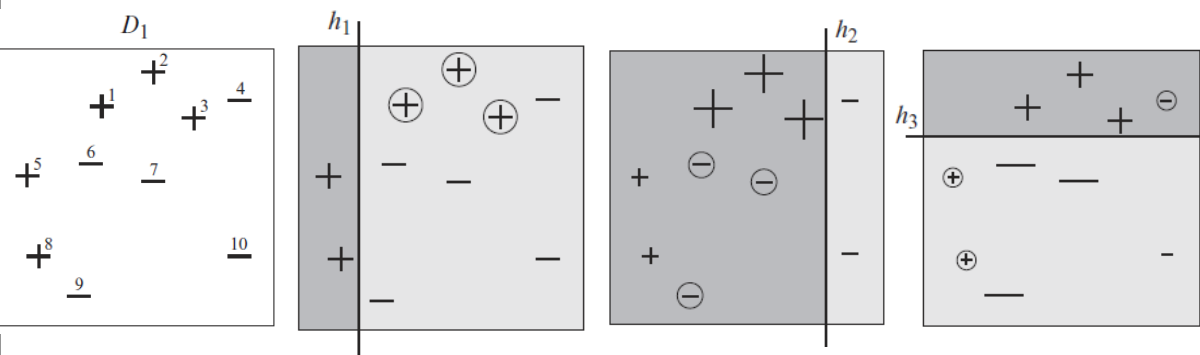
}

3) Clasificar a la observación x como:

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

Veamos un EJEMPLO





$$D_1(i) = 1/m$$

$$\epsilon_t \doteq \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

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

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

	1	2	3	4	5	6	7	8	9	10	
$D_1(i)$	<u>0.10</u>	<u>0.10</u>	<u>0.10</u>	0.10	0.10	0.10	0.10	0.10	0.10	0.10	$\epsilon_1 = 0.30$
$e^{-\alpha_1 y_i h_1(x_i)}$	1.53	1.53	1.53	0.65	0.65	0.65	0.65	0.65	0.65	0.65	$\alpha_1 \approx 0.42$
$D_1(i) e^{-\alpha_1 y_i h_1(x_i)}$	0.15	0.15	0.15	0.07	0.07	0.07	0.07	0.07	0.07	0.07	$Z_1 \approx 0.92$
$D_2(i)$	0.17	0.17	0.17	0.07	0.07	<u>0.07</u>	<u>0.07</u>	0.07	<u>0.07</u>	0.07	$\epsilon_2 \approx 0.21$
$e^{-\alpha_2 y_i h_2(x_i)}$	0.52	0.52	0.52	0.52	0.52	1.91	1.91	0.52	1.91	0.52	$\alpha_2 \approx 0.65$
$D_2(i) e^{-\alpha_2 y_i h_2(x_i)}$	0.09	0.09	0.09	0.04	0.04	0.14	0.14	0.04	0.14	0.04	$Z_2 \approx 0.82$
$D_3(i)$	0.11	0.11	0.11	<u>0.05</u>	<u>0.05</u>	0.17	0.17	<u>0.05</u>	0.17	0.05	$\epsilon_3 \approx 0.14$
$e^{-\alpha_3 y_i h_3(x_i)}$	0.40	0.40	0.40	2.52	2.52	0.40	0.40	2.52	0.40	0.40	$\alpha_3 \approx 0.92$
$D_3(i) e^{-\alpha_3 y_i h_3(x_i)}$	0.04	0.04	0.04	0.11	0.11	0.07	0.07	0.11	0.07	0.02	$Z_3 \approx 0.69$

Los ejemplos en los el clasificador débil comete error es se indican subrayados



Ponderamos cada clasificador débil, los sumamos y en función del signo clasificamos:

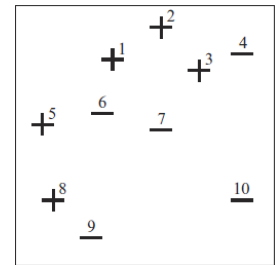
$$H = \text{sign} \left(0.42 \left[\begin{array}{|c|} \hline \text{shaded} \\ \hline \end{array} \right] + 0.65 \left[\begin{array}{|c|} \hline \text{shaded} \\ \hline \end{array} \right] + 0.92 \left[\begin{array}{|c|} \hline \text{shaded} \\ \hline \end{array} \right] \right)$$

Clasificador combinado

=

		+	
	+		+
			-
+	-	-	
+			-
	-		

Ejemplo, obs #4



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

$$\text{sign}(\overset{,42}{\alpha_1} - \overset{,65}{\alpha_2} + \overset{,92}{\alpha_3}) = \text{sign}(-0.15) = -1.$$

-1

↑

-1

↑

1

↑



Extra Gradient Boosting (XGBoost)

XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

ABSTRACT

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, we provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems.

problems. Besides being used as a stand-alone predictor, it is also incorporated into real-world production pipelines for ad click through rate prediction [15]. Finally, it is the de-facto choice of ensemble method and is used in challenges such as the Netflix prize [3].

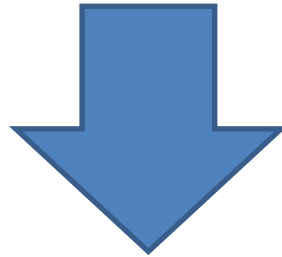
In this paper, we describe XGBoost, a scalable machine learning system for tree boosting. The system is available as an open source package². The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions³ published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model



XGBoost usa aproximaciones más precisas al emplear gradientes de segundo orden y regularización avanzada.

XGBoost ofrece características como:

- Computación distribuida.
- Paralelización.
- Optimización de caché.



Big Data + Alta Performance




Desarrollo del modelo de XGBoost

Utilizaremos la nomenclatura de Tianqi Chen en su paper.

Primero presentemos el desarrollo de un modelo de expansión aditivo. Es intratable aprender todos los árboles a la vez. En su lugar, utilizamos una estrategia aditiva: fijar lo que hemos aprendido, y añadir un nuevo árbol a la vez.

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\dots\end{aligned}$$

Predicción en árbol (t) 
$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$



Queda por preguntarse: ¿qué árbol queremos en cada paso? Lo natural es añadir el que optimiza nuestro objetivo.

$$\begin{aligned}\text{obj}^{(t)} &= \sum_{i=1}^n (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant}\end{aligned}$$

La forma de la función de pérdida cuadrática se impone de un término de primer orden (el residuo) y un término cuadrático. Si hacemos una expansión de Taylor de la función de pérdida hasta el segundo orden, nos queda:

$$\text{obj}^{(t)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

donde:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$



Después de eliminar todas las constantes, el objetivo específico en el paso t se convierte en

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Regularización en XGBoost

Para presentar los términos que definen la complejidad del árbol y su consiguiente regularización primero veamos la definición del árbol:

$$f_t(x) = w_{q(x)}, w \in R^T, q: R^d \rightarrow \{1, 2, \dots, T\}.$$

Aquí w es el vector de valores que generan las hojas, q es la función que asigna cada punto de datos a la hoja correspondiente, y T es el número de hojas. En XGBoost, definimos la complejidad como

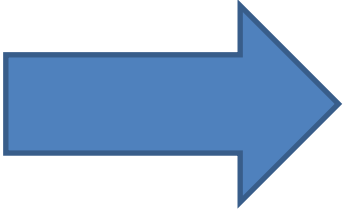
$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$



Ahora podemos reformular la función objetivo con los hiperparámetros de regularización

$$\begin{aligned}\text{obj}^{(t)} &\approx \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T\end{aligned}$$

Si definimos $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$:


$$\text{obj}^{(t)} = \sum_{j=1}^T [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T$$



Armando la estructura del árbol

Ahora que tenemos una forma de medir la calidad de un árbol, lo ideal sería enumerar todos los árboles posibles y elegir el mejor. En la práctica esto es intratable, así que intentaremos optimizar un nivel del árbol cada vez. En concreto, tratamos de dividir una hoja en dos hojas, y la ganancia que obtiene es

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

esta fórmula puede descomponerse en:

- 1) la ganancia en la nueva hoja izquierda
- 2) la ganancia en la nueva hoja derecha
- 3) la ganancia en la hoja original
- 4) la regularización en la hoja adicional.

Podemos ver un hecho importante aquí: si la ganancia es menor que γ , haríamos mejor en no añadir esa rama. Esto es exactamente las técnicas de poda en los modelos basados en árboles.

