uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
APRENDIZAJE AUTOMÁTICO PARA EL ANÁLISIS DE DATOS
GRADO EN ESTADÍSTICA Y EMPRESA
Ricardo Aler

CONJUNTOS (ENSEMBLES)

BOOSTING, BAGGING, …

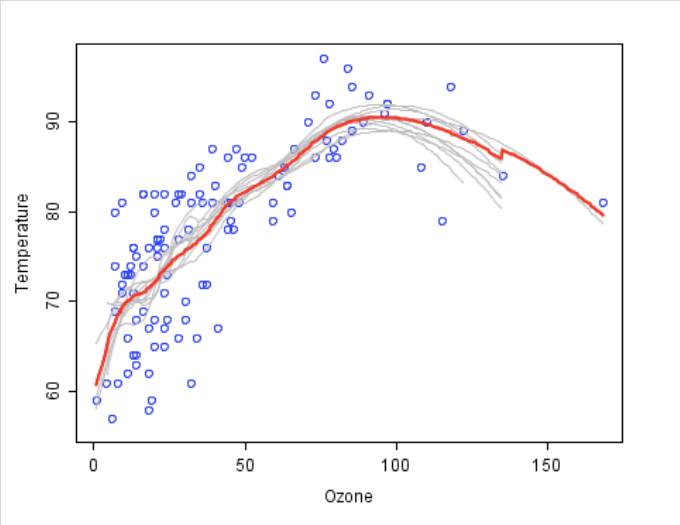
Conjuntos (ensembles)

- Construyen varios modelos (llamados modelos base) y después los usan de manera conjunta
- Suelen ser mas precisos que los modelos base
- Tipos principales:
  - Bagging: construye varios modelos base en paralelo con el mismo método. Clasifica datos por votación y regresión con media.
    - Variante: Random Forests: Crea un ensemble de varios árboles de decisión
  - Boosting: construye varios modelos base de manera secuencial, cada uno se centra en los datos difíciles para el modelo anterior.
    - Variante: Gradient tree boosting: Crea un ensemble de varios árboles de regresión

BAGGING (BOOSTRAP AGGREGATING)

Bagging (Bootstrap aggregating)

- Motivación: hay métodos de construcción de modelos que se ajustan excesivamente a la muestra de entrenamiento (métodos inestables, de alta varianza, overfitting)
  - Inestables: redes de neuronas, árboles (especialmente con profundidad alta), …
  - Estables: vecino más cercano (KNN), SVM's, …
- Estrategia Bagging: dado que los sobreajustes serán debidos a sesgos aleatorios presentes en el conjunto de entrenamiento, generar modelos con distintas muestras y calcular la media de los distintos modelos.

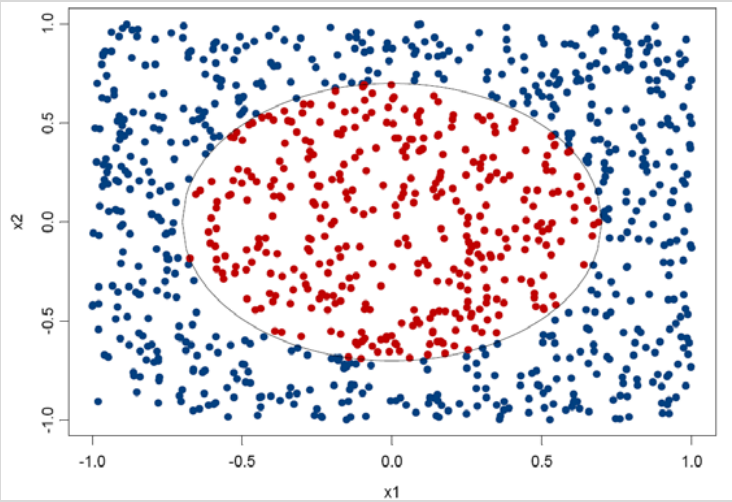


Source: https://cdn2.hubspot.net/hubfs/2575516/Imported_Blog_Media/skitch.png

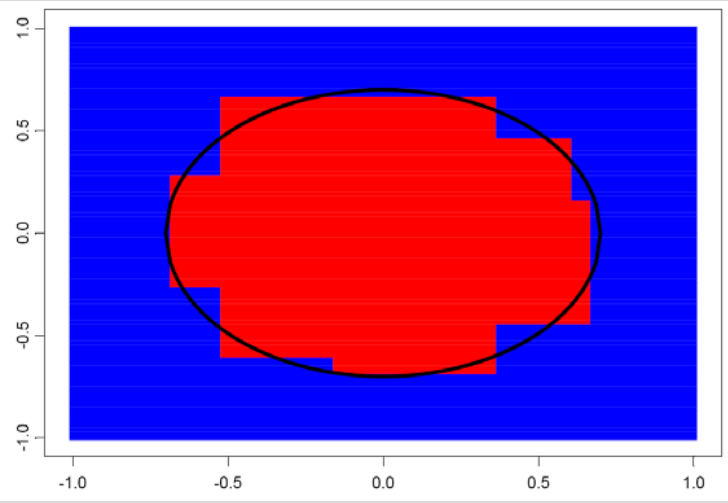
Una visión geométrica para clasificación

- La frontera “media” es mejor que una frontera concreta.

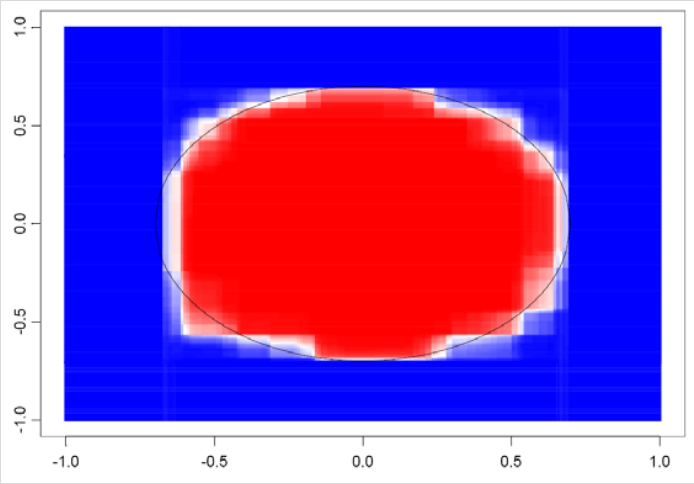
Datos originales



Frontera de un único árbol

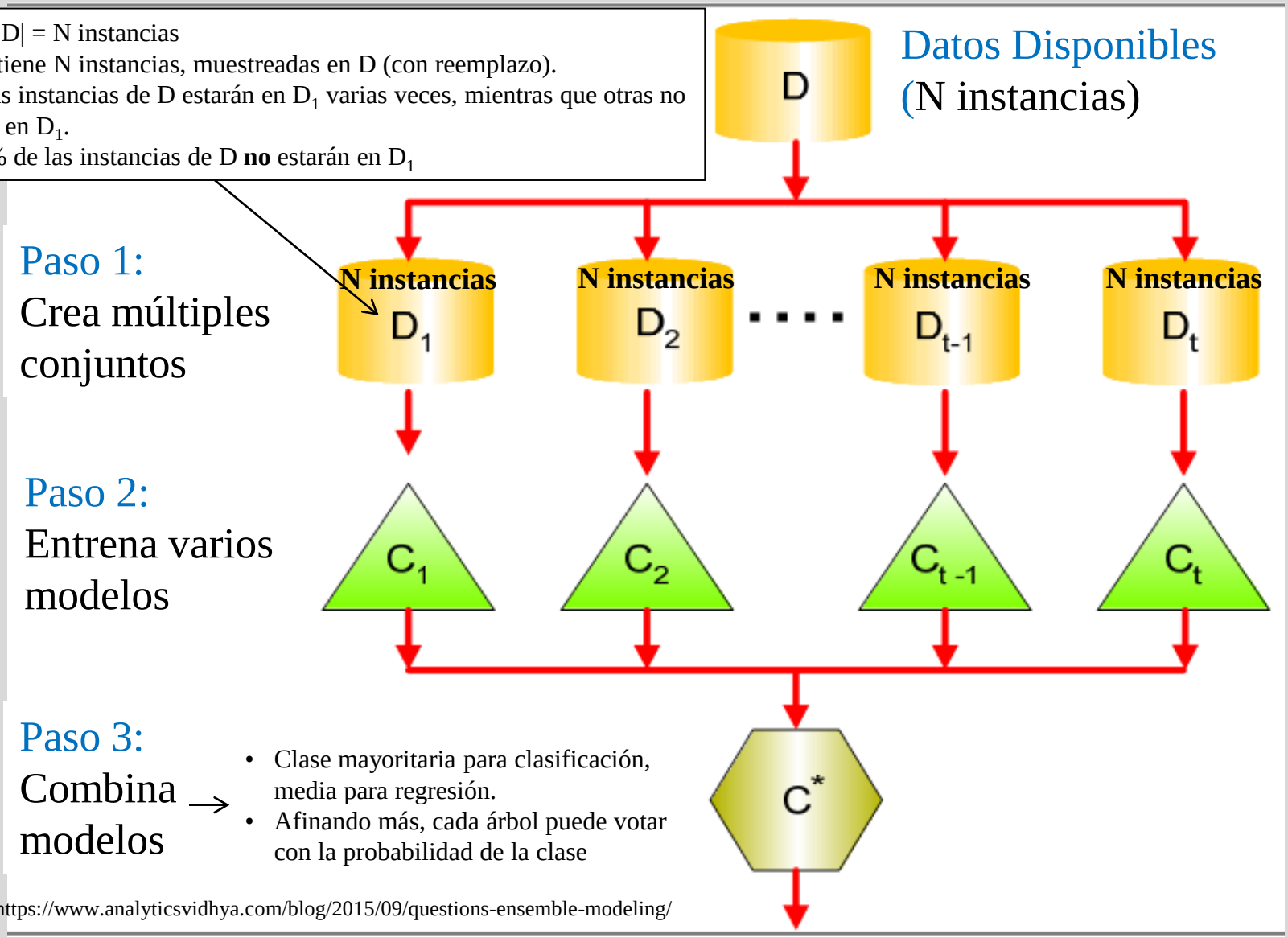


Frontera de un bagging con 100 árboles



Bagging (Bootstrap aggregating)

- $|D_1| = |D| = N$ instancias
- D_1 contiene N instancias, muestreadas en D (con reemplazo).
- Algunas instancias de D estarán en D_1 varias veces, mientras que otras no estarán en D_1.
- 33-36% de las instancias de D **no** estarán en D_1



Paso 1:
Crea múltiples conjuntos

Paso 2:
Entrena varios modelos

Paso 3:
Combina modelos →

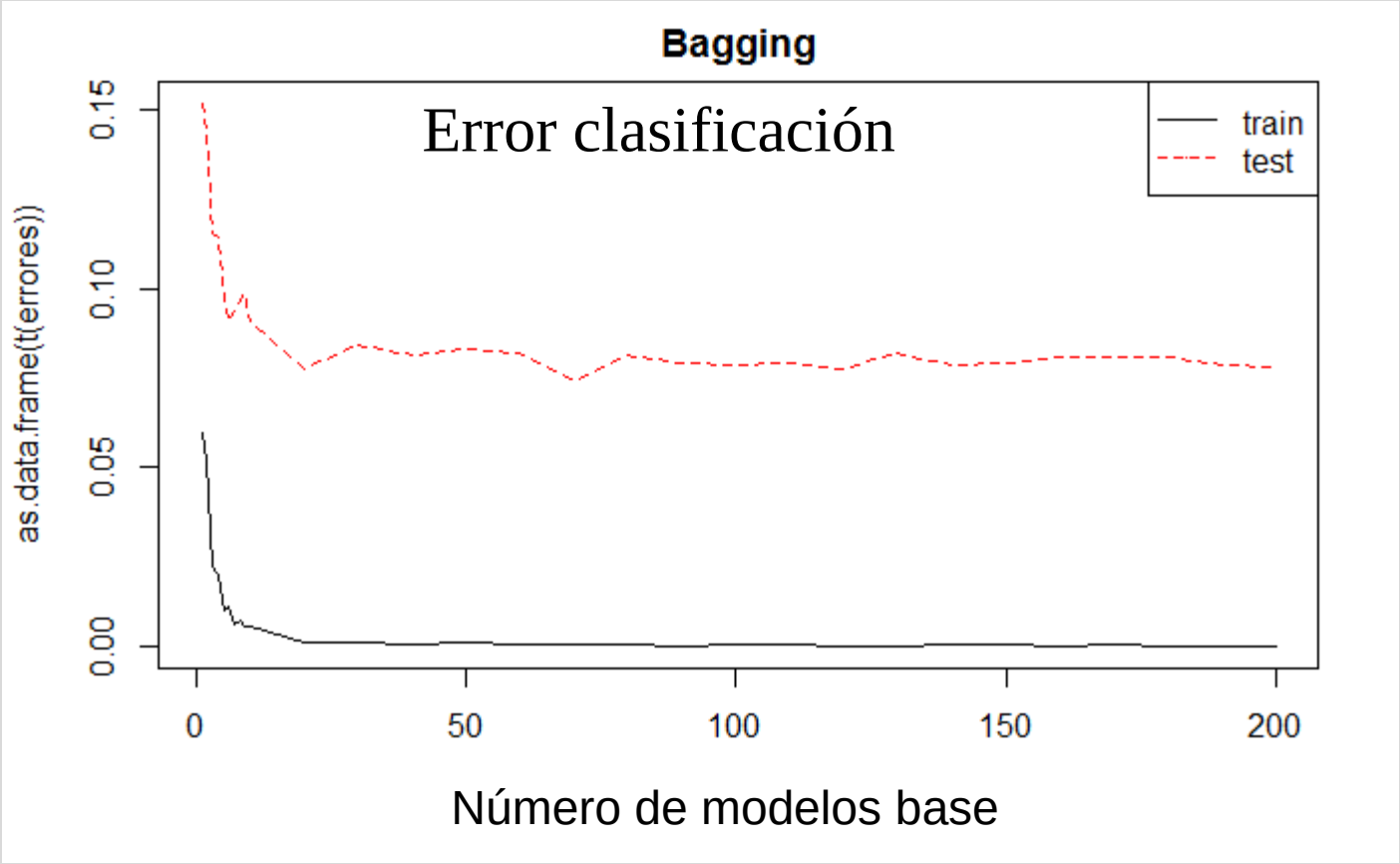
- Clase mayoritaria para clasificación, media para regresión.
- Afinando más, cada árbol puede votar con la probabilidad de la clase

Source: https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/

Randomización

- La randomización es otra manera de crear una variedad de modelos (distinta a la creación de varios conjuntos de entrenamiento.
- Algunos algoritmos de aprendizaje son estocásticos: cada vez que se usan, generan un modelo (ligeramente) diferente, aunque se comience con exactamente el mismo conjunto de entrenamiento
  - Ej: redes de neuronas, ya que los pesos iniciales son aleatorios.
- Pero también podemos modificar algoritmos no estocásticos para que lo sean.
- Los Random Forests usan tanto muestreo con reemplazo como randomización.

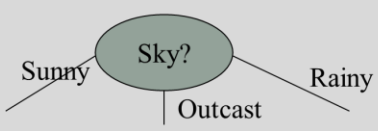
Bagging y descenso del error

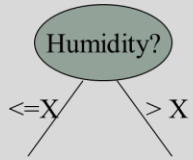


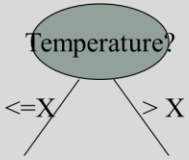
Random Forests

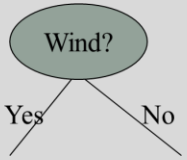
Random Forests

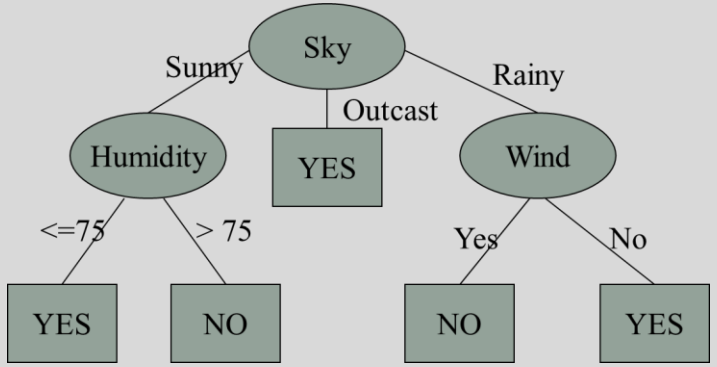
- RF = **Bagging** con árboles:
  - Muestreo aleatorio con reemplazo.
  - Randomización: al elegir un atributo para un nodo, no se elige al mejor de todos ellos, sino que se elige el mejor de un subconjunto con m atributos. Por ejemplo, si hay M=16 atributos y m=4, entonces 4 se cogen 4 aleatoriamente, y el mejor de esos 4 es seleccionado.
    - Normalmente, m=sqrt(M) para clasificación y m = M/3 para regresión.
    - Si m == M, entonces RF == Bagging
    - Al hiper-parámetro *m* se lo suele denominar *mtry*











Resultados de Random Forests

- Normalmente, RF es mejor que árboles individuales

Test set misclassification error (%)

Data set	**Forest**	**Single tree**
Breast cancer	2.9	5.9
Ionosphere	5.5	11.2
Diabetes	24.2	25.3
Glass	22.0	30.4
Soybean	5.7	8.6
Letters	3.4	12.4
Satellite	8.6	14.8
Shuttle $\times 10^3$	7.0	62.0
DNA	3.9	6.2
Digit	6.2	17.1

Random Forests. Evaluación"Out of bag"

- Sabemos que para evaluar un modelo es necesario hacer el test usando datos distintos a los de entrenamiento.
- Típicamente se usa train/test (hold-out) o crossvalidation.
- Los RF son capaces de evaluar el modelo sin separar una parte de los datos (la partición de test)
- Se aprovecha de que no todos los datos de D están en todas las muestras D_i. Las que no están, se pueden usar para evaluar el árbol T_i
- El error de cada dato se calcula usando únicamente aquellos árboles en los que no se utilizó dicho dato para construirlo

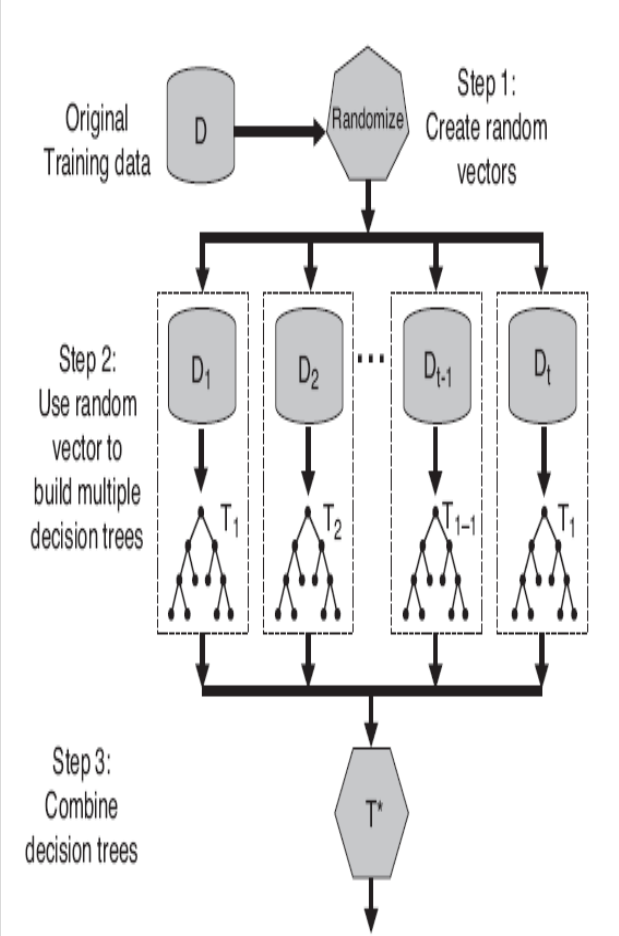


Figure 5.40. Random forests.

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Random Forests. Importancia de los atributos

- Se calcula el error out-of-bag $\hat{e}$
- Recordemos que un atributo a_k no es mas que una columna de valores en la tabla de datos
- Reordenamos aleatoriamente los valores de a_k en la tabla de datos de entrenamiento y calculamos el nuevo error out-of-bag $\hat{e}_{ak}$
- Ordenamos a las variables por diferencia de error: $\hat{e}_{ak}$- $\hat{e}$. Aquellas que tengan mayores diferencias son mas importantes para la predicción

RF y distancias

- RF proporciona también una función de similaridad: la similaridad $W(\boldsymbol{x}_i,\boldsymbol{x}_j)$ entre $\boldsymbol{x}_i$ y $\boldsymbol{x}_j$ es la proporción de veces que ambas terminan en la misma hoja (recordar que tenemos M árboles)
- Se puede calcular una "dissimilaridad" (pseudo-distancia) como:

  $sqrt(1- W(\boldsymbol{x}_i,\boldsymbol{x}_j))$
- Podríamos usar esta dissimilaridad como distancias para KNN, por ejemplo

Random Forests

- Sólo dos hiper-parámetros: número *k* de árboles en el ensemble y tamaño *mtry* del subconjunto de atributos
- Típicamente mejor que los árboles individuales
- Más rápido que Bagging (menos atributos a evaluar para cada nodo: *mtry* << *M*)
- Evaluación: proporciona una evaluación del modelo (out-of-bag)
- Ranking: ordena los atributos según su importancia
- Estimación de probabilidades: puede cuantificar el grado de creencia en una predicción: si 90 árboles dicen “si” y “10”, el grado de creencia es 0.9

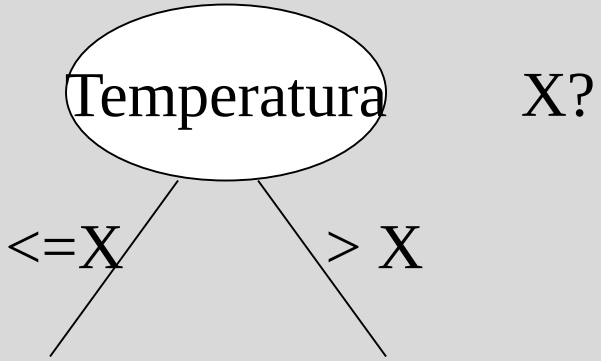
Extremely randomized tres (Extra-trees)

- El RF estándar usa dos métodos de aleatorización para construir los distintos modelos del ensemble:
  - Creación de varias muestras con reemplazo
  - Selección aleatoria de *m* atributos de entre los *M*, antes de elegir el mejor
- Los extra-trees funcionan de manera similar, excepto que:
  - Se usa el conjunto de entrenamiento original para todos los miembros del ensemble
  - Se eligen *m* atributos aleatoriamente:
    - Si el atributo es numérico, el punto de corte se elige aleatoriamente.
    - Si el atributo es categórico, se construye una decisión binaria aleatoriamente:
      - Ej: si los valores del atributo A son {a,b,c,d,e}, una decisión binaria aleatoria podría ser: if(A ∈ {c,e}) left else right.
  - Una primera ventaja es que va a ser más rápido

Recordatorio: ¿Y si el atributo es contínuo?

Hay que partir por el valor X, donde sea mas conveniente, minimizando la entropía

Nota: solo hemos probado algunas de los posibles puntos de corte (thresholds), siendo el mejor X=84 con entropia media ponderada = 0.83



64 – Si, 65-No, 68 – Si, 69 – Si, 70 – Si, 71 – No, 72 – SiNo, 75 – SiSi, 80 – No, 81 – Si, 83 – Si, 85 - No

HP = 0.83

4 No, 9 Si

X=84

1 No, 0 Si

64 – Si, 65-No, 68 – Si, 69 – Si, 70 – Si, 71 – No, 72 – SiNo, 75 – SiSi, 80 – No, 81 – Si, 83 – Si, 85 - No

HP = 0.93

2 No, 4 Si

X=71.5

3 No, 5 Si

64 – Si, 65-No, 68 – Si, 69 – Si, 70 – Si, 71 – No, 72 – SiNo, 75 – SiSi, 80 – No, 81 – Si, 83 – Si, 85 - No

HP = 0.89

1 No, 4 Si

X=70.5

4 No, 5 Si

Resultados de Extra-trees

- En 12 problemas de clasificación y 12 de regresión

Table 2 Win/Draw/Loss records (corrected *t*-tests) comparing the algorithm in the column versus the algorithm in the row

	Classification problems					Regression problems				
	PST	TB	RS*	RF*	ET^d	PST	TB	RS*	RF*	ET^d
PST	–	8/4/0	11/1/0	11/1/0	10/2/0	–	10/2/0	8/4/0	10/2/0	10/2/0
TB	0/4/8	–	7/5/0	7/5/0	7/5/0	0/2/10	–	0/9/3	1/11/0	4/8/0
RS*	0/1/11	0/5/7	–	0/8/4	2/10/0	0/4/8	3/9/0	–	4/8/0	4/7/1
RF*	0/1/11	0/5/7	4/8/0	–	5/7/0	0/2/10	0/11/1	0/8/4	–	3/7/2
ET^d	0/2/10	0/5/7	0/10/2	0/7/5	–	0/2/10	0/8/4	1/7/4	2/7/3	–

- PST = árboles individuales (construidos con CART)
- TB = Tree Bagging
- RF = Random Forests

Resultados de Extra-trees

- ET tarda menos habitualmente

Table 4 Computing times (msec) of training (ensembles of $M = 100$ trees)

Classification problems					Regression problems			
Dataset	ST	TB	RF^d	ET^d	Dataset	ST	TB/RF^d	ET^d
Waveform	68	4022	1106	277	Friedman1	7	372	284
Two-Norm	66	3680	830	196	Housing	12	685	601
Ring-Norm	101	4977	1219	251	Hwang-f5	16	917	742
Vehicle	80	5126	1500	685	Hwang-f5n	15	948	748
Vowel	236	14445	4904	694	Pumadyn-32fh	251	13734	9046
Segment	291	18793	5099	1053	Pumadyn-32nm	221	11850	9318
Spambase	822	55604	9887	8484	Abalone	73	4237	3961
Satellite	687	45096	11035	5021	Ailerons	495	29677	26572
Pendigits	516	34449	12080	5183	Elevators	289	15958	13289
Dig44	4111	259776	67286	9494	Poletelecomm	497	28342	26576
Letter	665	44222	17041	14923	Bank-32nh	613	34402	20178
Isolet	37706	2201246	126480	11469	Census-16H	597	35207	27900

BOOSTING

Boosting

- Motivación: mejorar (*to boost*) métodos débiles (*weak learners*):
  - se usan métodos que por sí solos consiguen un error no muy por encima del azar o *weak learners* (ej: árboles de decisión poco profundos)
  - Se van añadiendo modelos al ensemble secuencialmente, de tal manera que el siguiente modelo corrija los errores del modelo anterior

Boosting

- En Boosting, las muestras de entrenamiento se construyen de manera secuencial, de manera que el siguiente modelo del ensemble intenta solucionar los errores del modelo previo
- En el Boosting original (Adaboost), los datos de entrenamiento son una lista de tuplas ($\mathbf{x}$, y):

  $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_a,y_a), \ldots, (\mathbf{x}_N,y_N)\}$
- Cada dato tiene un peso w_a, inicialmente todos $w_a=1/N$:

  $\{(\mathbf{x}_1,y_1,w_1=1/N), \ldots, (\mathbf{x}_a,y_a,w_a=1/N), \ldots, (\mathbf{x}_N,y_N,w_N=1/N)\}$
- Los pesos se irán adaptando, de manera que los datos difíciles pasarán a tener más peso y los más fáciles, menos.

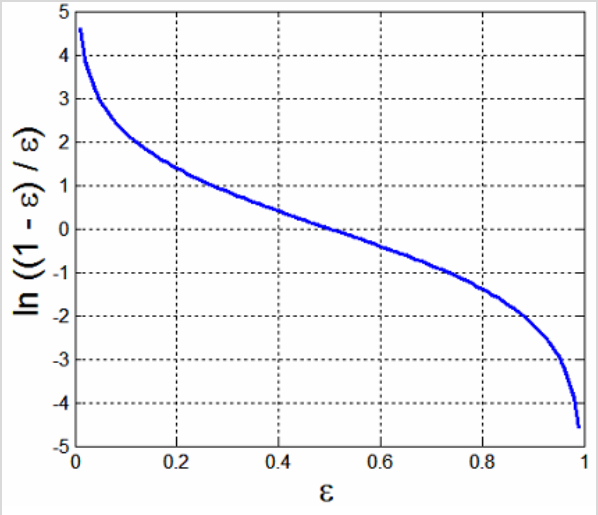
Adaboost.M1 (boosting para clasificación)

1. Inicialmente, todos los datos de entrenamiento tienen el mismo peso (w_a=1/N)
2. Construye un clasificador h_0 (ej: árbol poco profundo o *decisión stump*). Su error es ε_0
3. Repite mientras $0<\varepsilon_i<0.5$
    1. Observa en que datos falla h_{i-1}
    2. Construye un nuevo conjunto de entrenamiento, dándole mas importancia a los datos fallidos:
        1. Si h_{i-1} clasifica mal el dato (x_a,y_a), aumenta el peso $w'_a=w_a^* (1-\varepsilon_{i-1})/\varepsilon_{i-1}$
        2. Si h_{i-1} clasifica bien el dato (x_a,y_a), decrementa el peso $w'_a=w_a^* \varepsilon_{i-1}/(1-\varepsilon_{i-1})$
    3. Construye un nuevo clasificador h_i con los nuevos datos. Su error es ε_i (calculado sobre la muestra con pesos)

El clasificador final *f* es una combinación de todos los h_i. Los coeficientes alfa dependen de lo preciso que sea el clasificador h_i (ε_i)

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Notas:

- *h_i tiene que devolver un valor +1 o -1 (clase positiva o negativa en problemas biclase), o un valor intermedio.*

¿Cómo usar los pesos w_a con las instancias de entrenamiento?

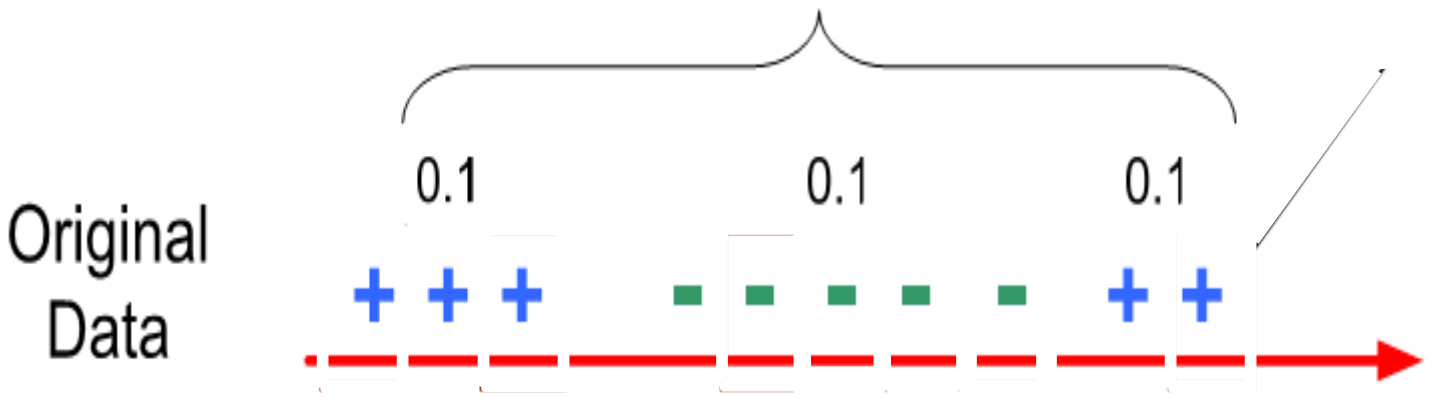
- Algunos algoritmos son capaces de trabajar con datos de entrenamiento donde cada instancia tiene un peso (por ejemplo, los árboles pueden)
- Si no pueden, la nueva muestra de entrenamiento se puede crear muestreando aleatoriamente, con probabilidad de elegir una instancia proporcional a su peso =

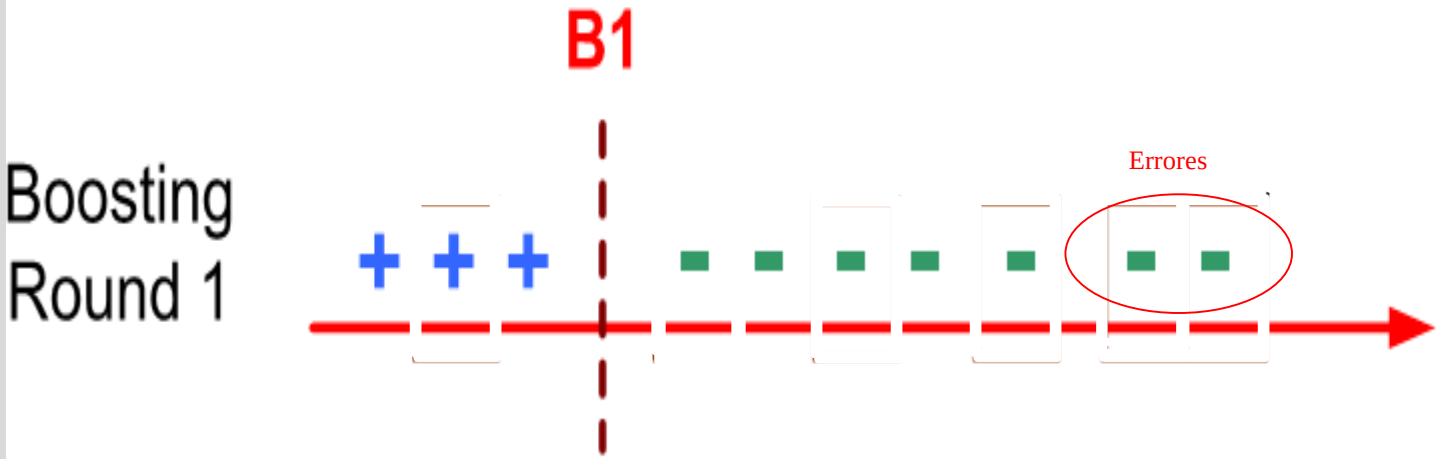
$$prob_a = w_a \ / \ (w_1+w_2+\ldots+w_N)$$

Así, algunas instancias aparecerán dos o más veces (a costa de que otras no aparezcan ninguna).

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

Pesos iniciales para cada instancia



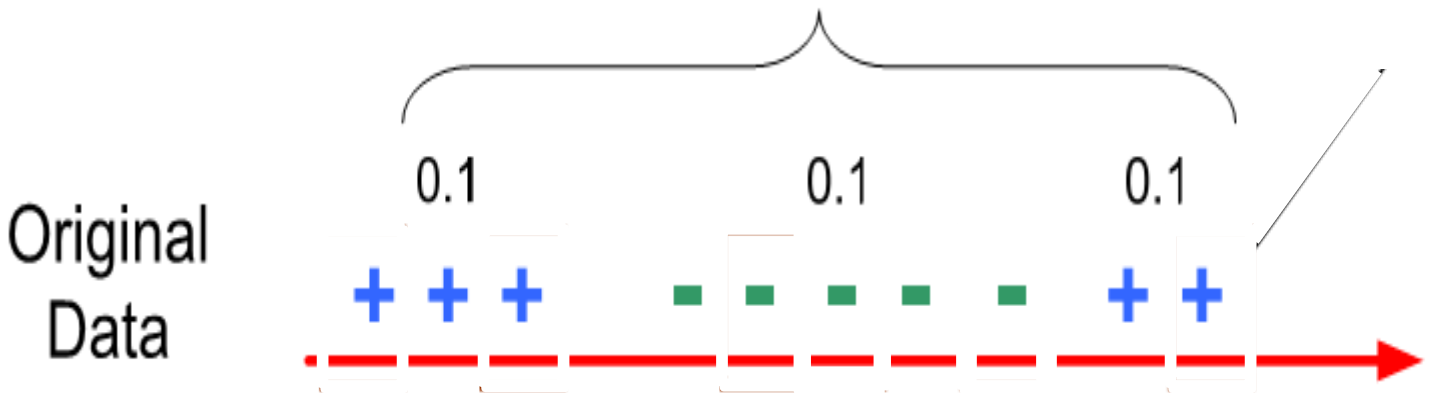


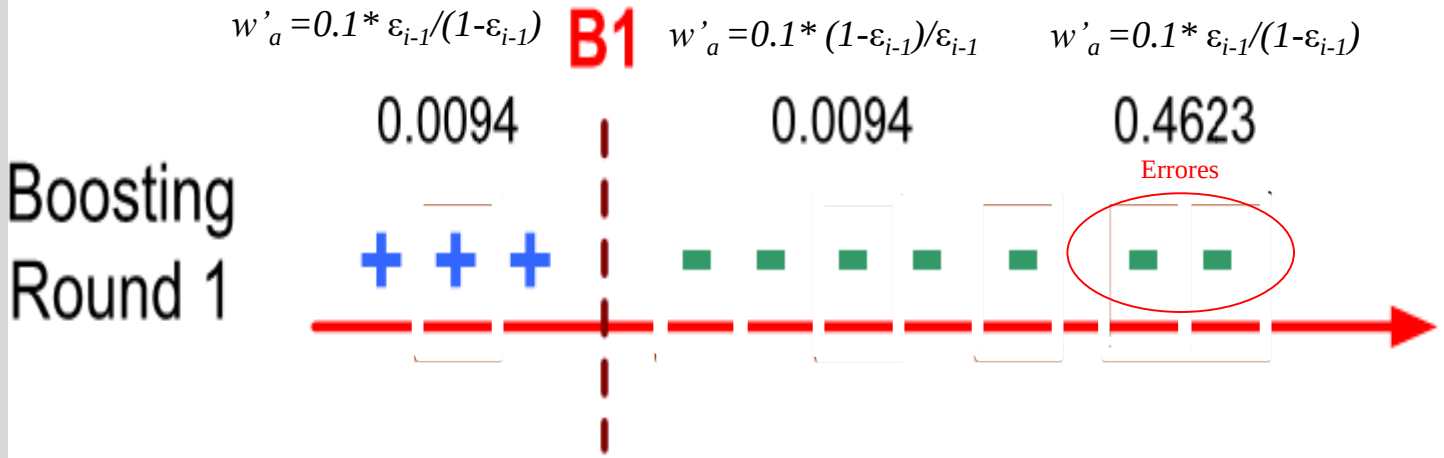
$\alpha = 1.9459$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf

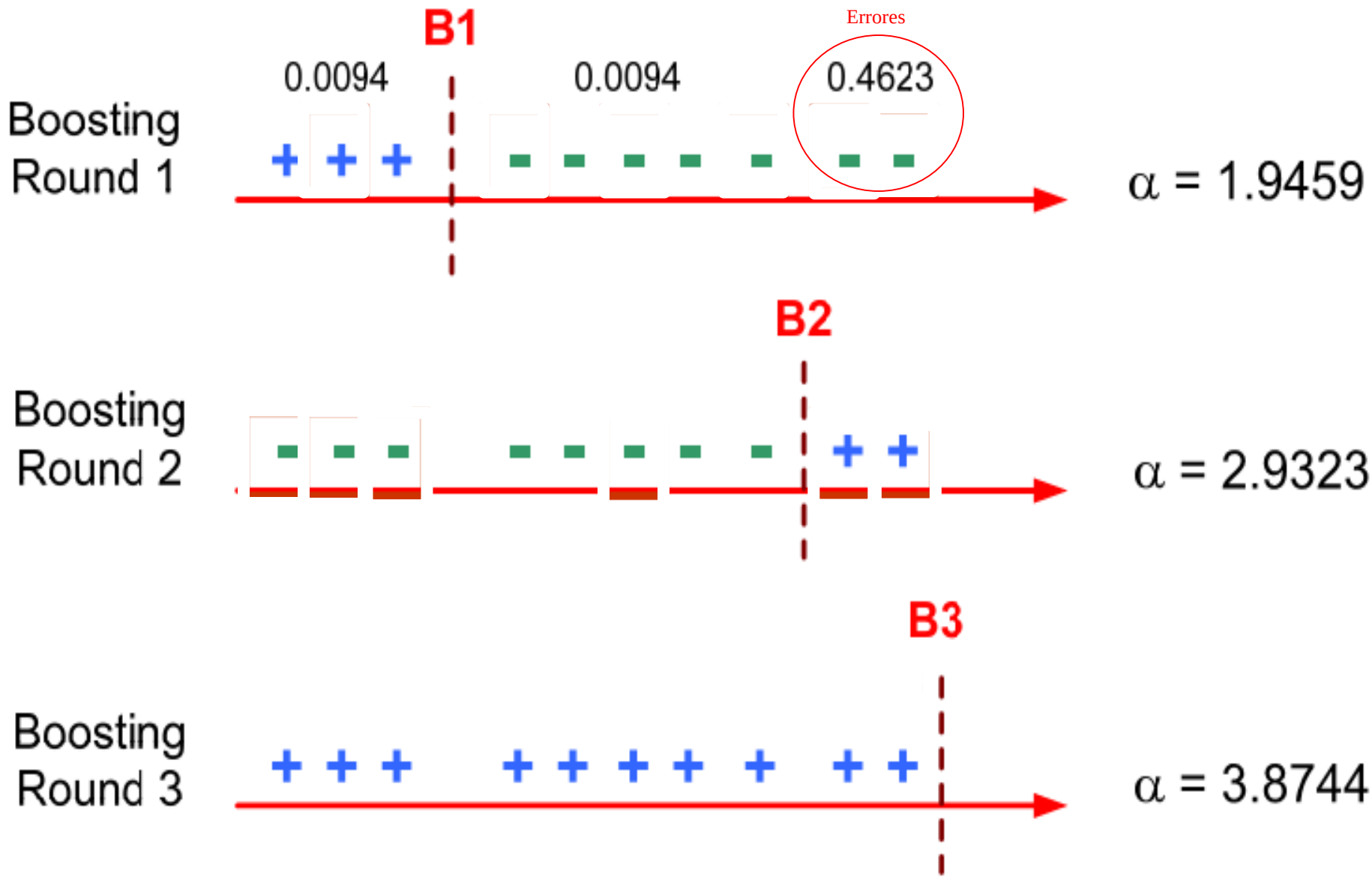
Pesos iniciales para cada instancia



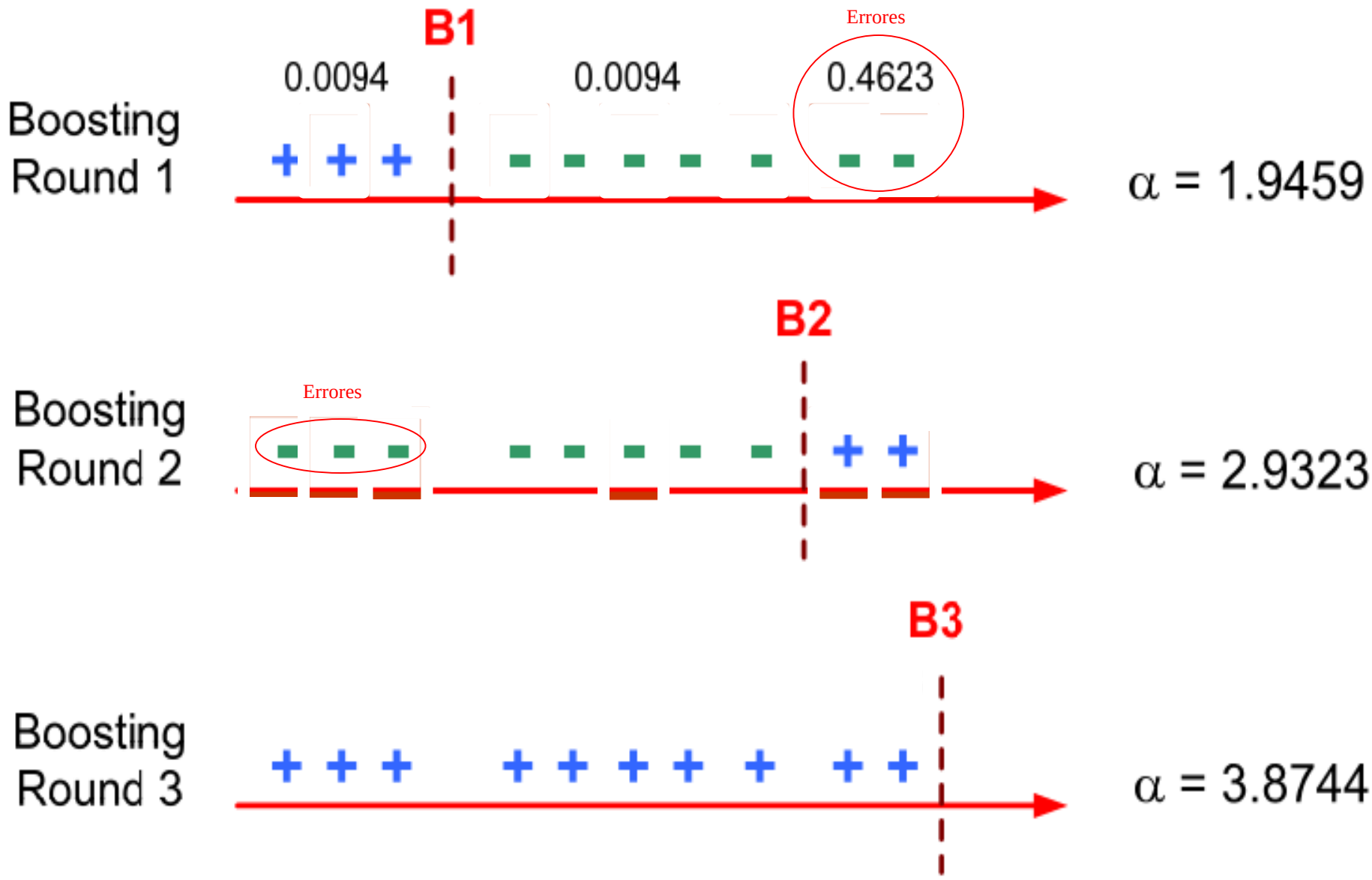


$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

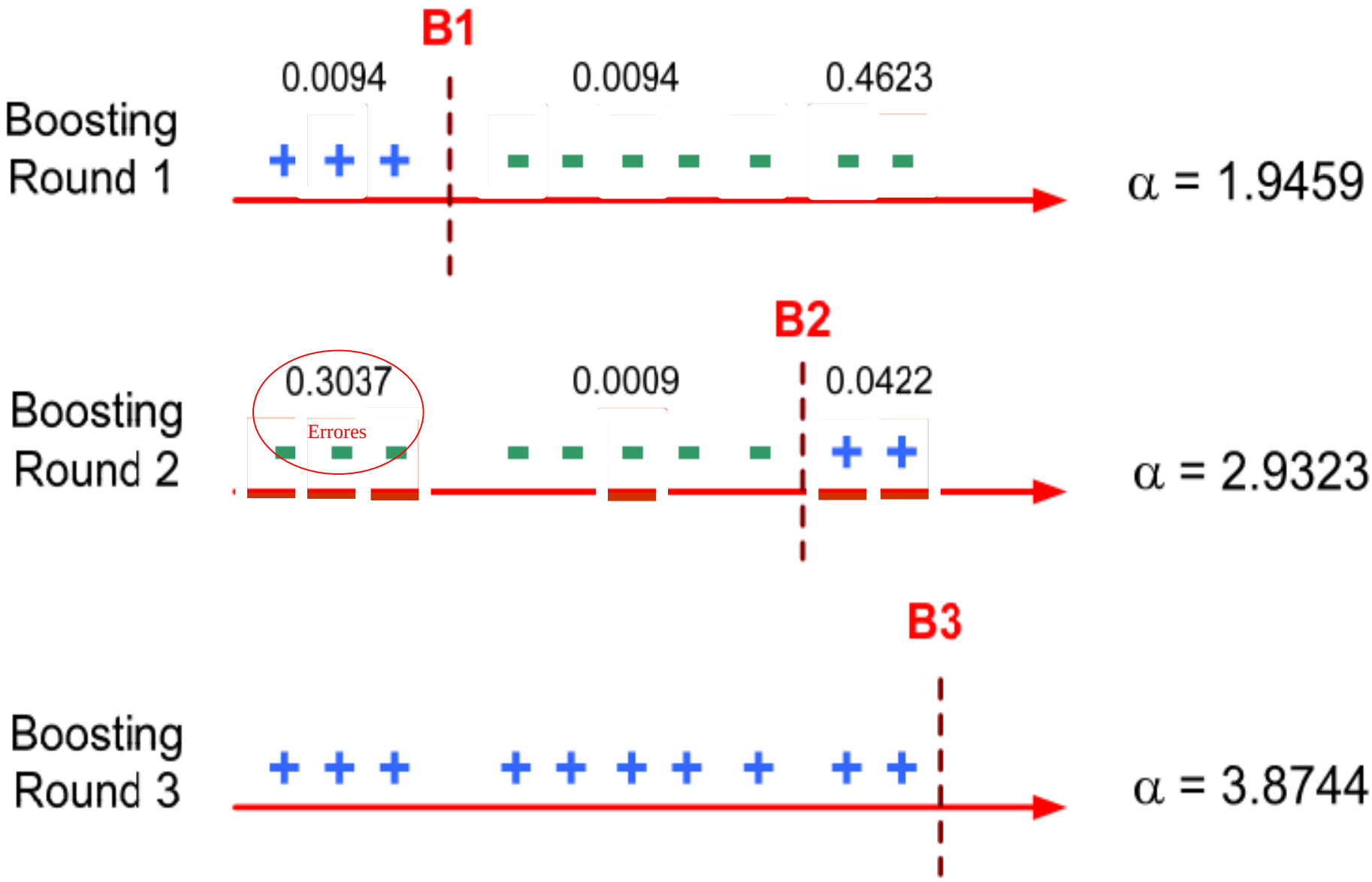
Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



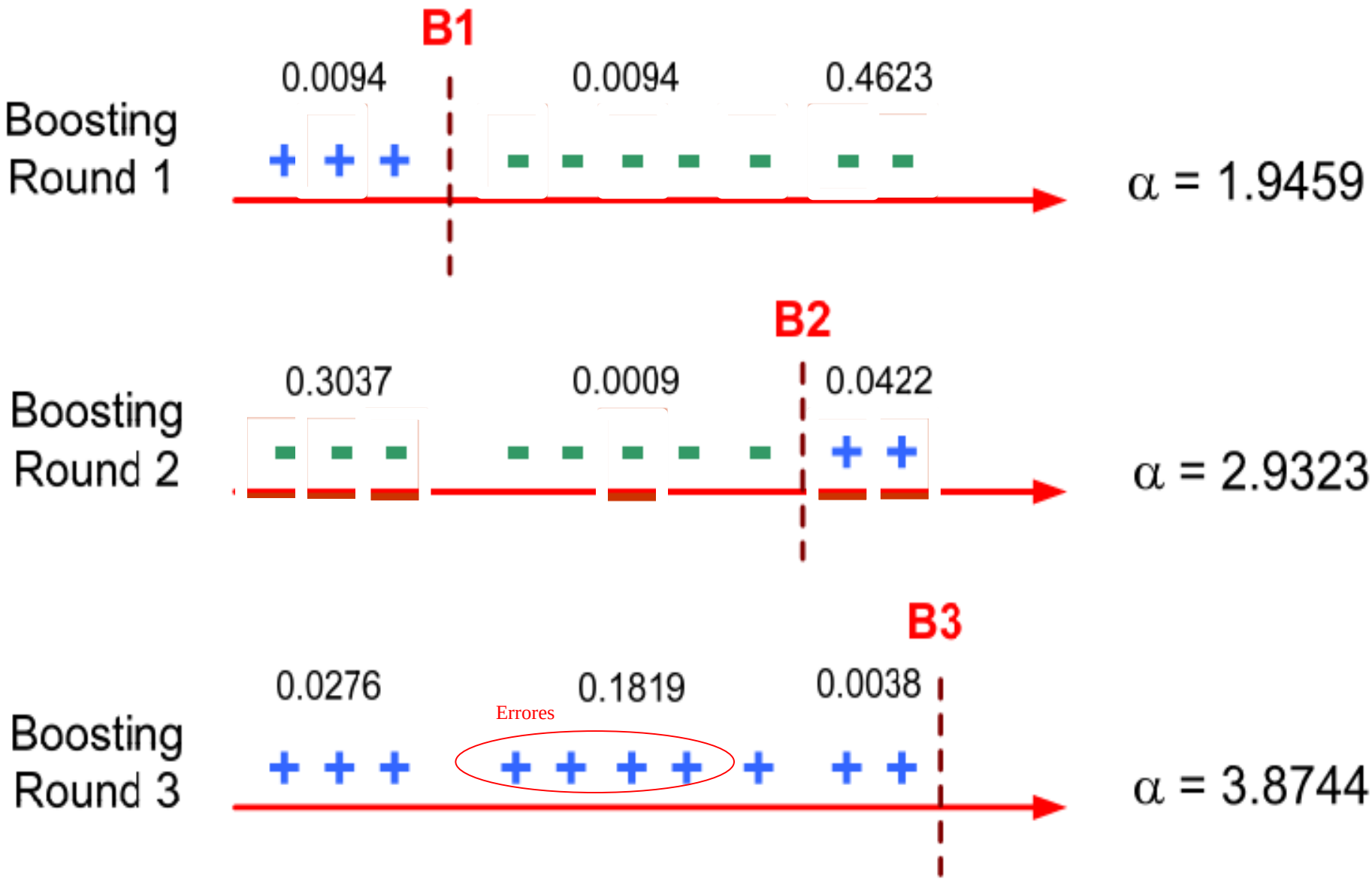
Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



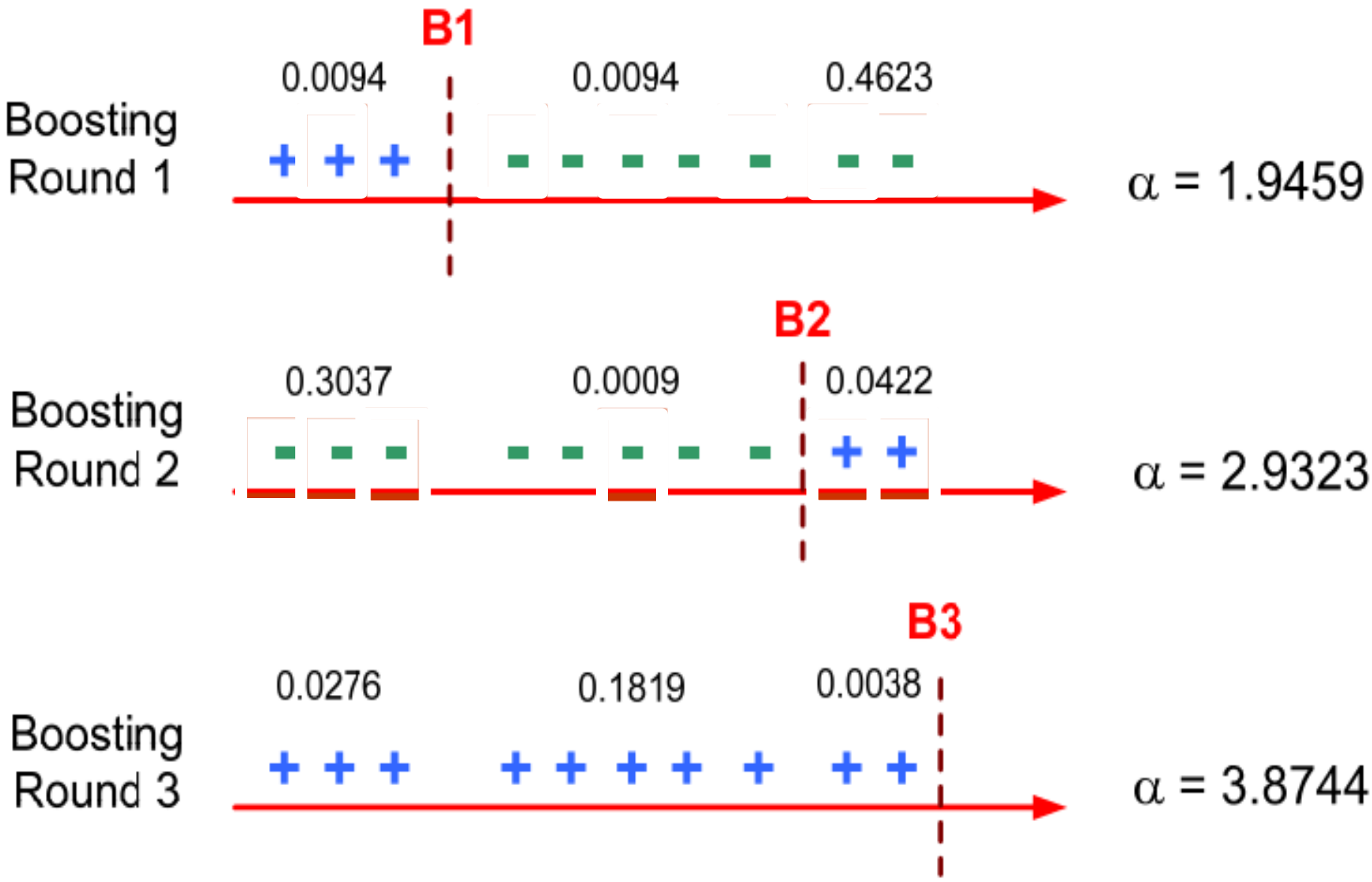
Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



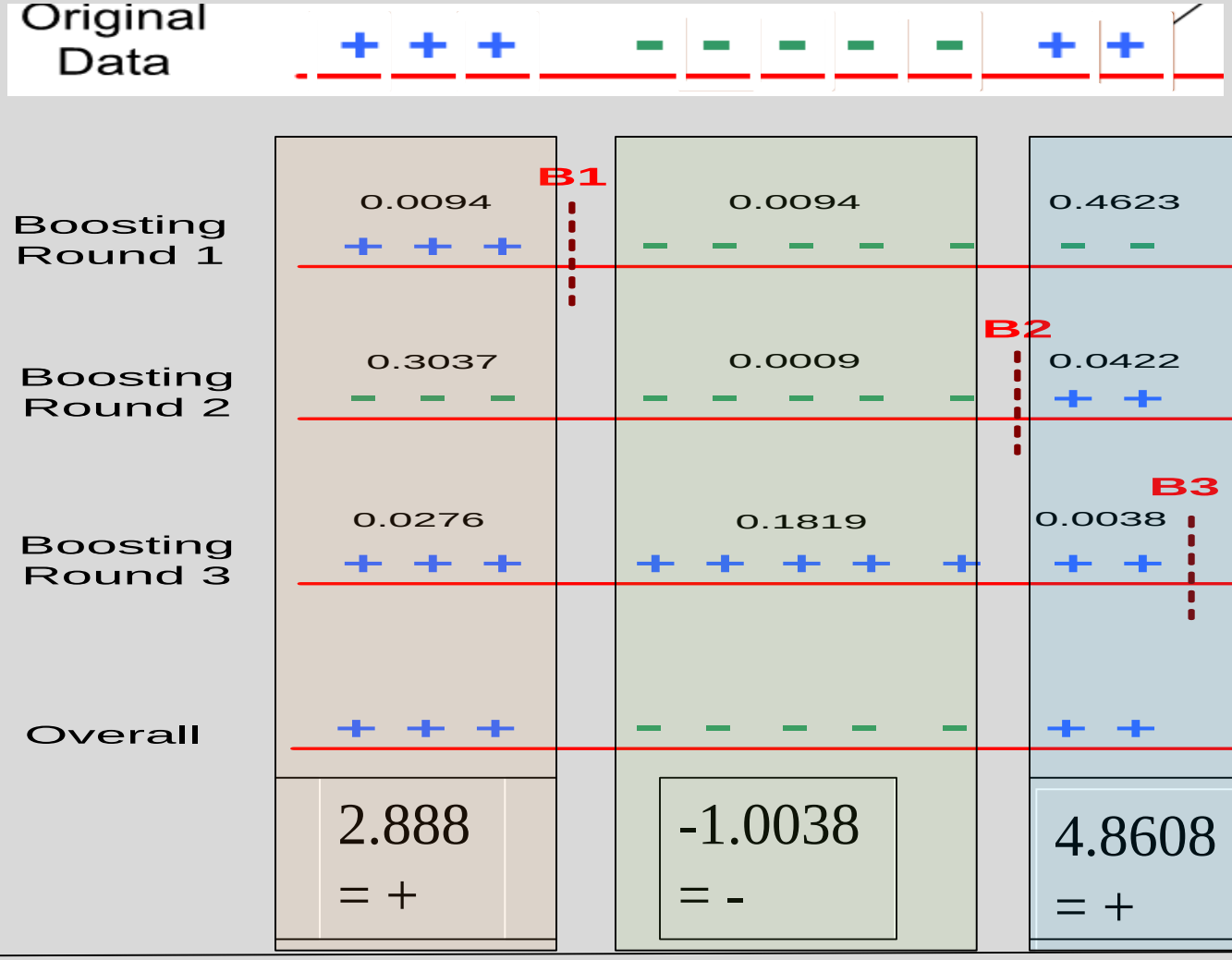
Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



Source: https://nptel.ac.in/content/storage2/nptel_data3/html/mhrd/ict/text/106105152/lec40.pdf



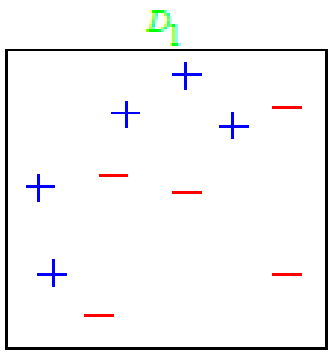
1.9459*B1(x) + 2.9323*B2(x)+ 3.8744*B3(x) = f(x)

1.9459*(+1) + 2.9323*(-1)+ 3.8744*(+1) = 2.888 > 0

1.9459*(-1) + 2.9323*(-1)+ 3.8744*(+1) = -1.0038 < 0

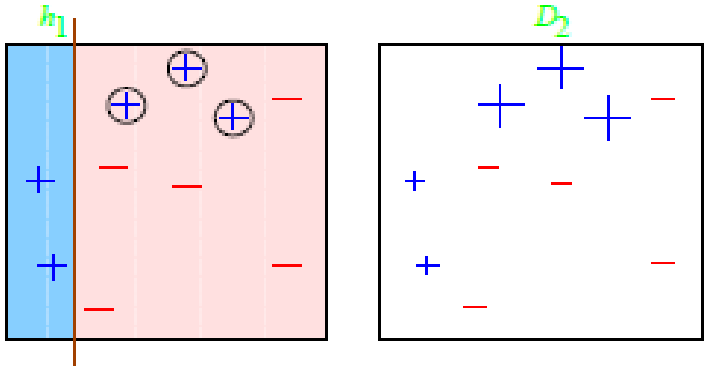
1.9459*(-1) + 2.9323*(+1)+ 3.8744*(+1) = 4.8608 > 0

Ejemplo de Adaboost en 2D



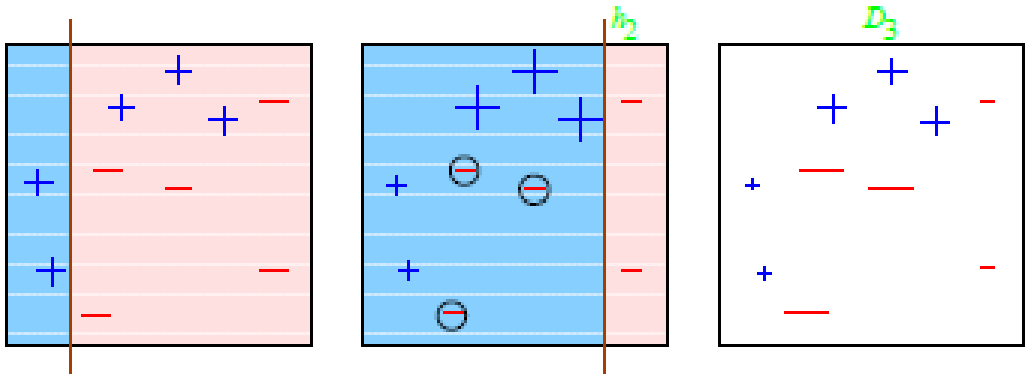
Weak hypotheses == vertical or horizontal half-planes

Round 1



Source:
https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/

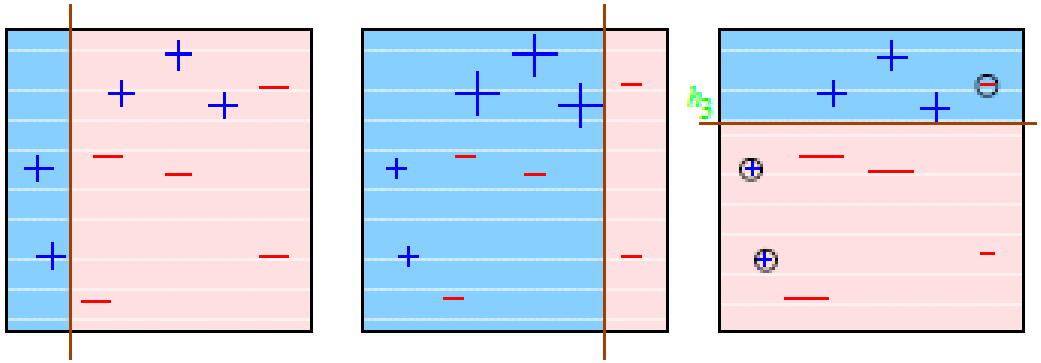
Round 2



$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Round 3

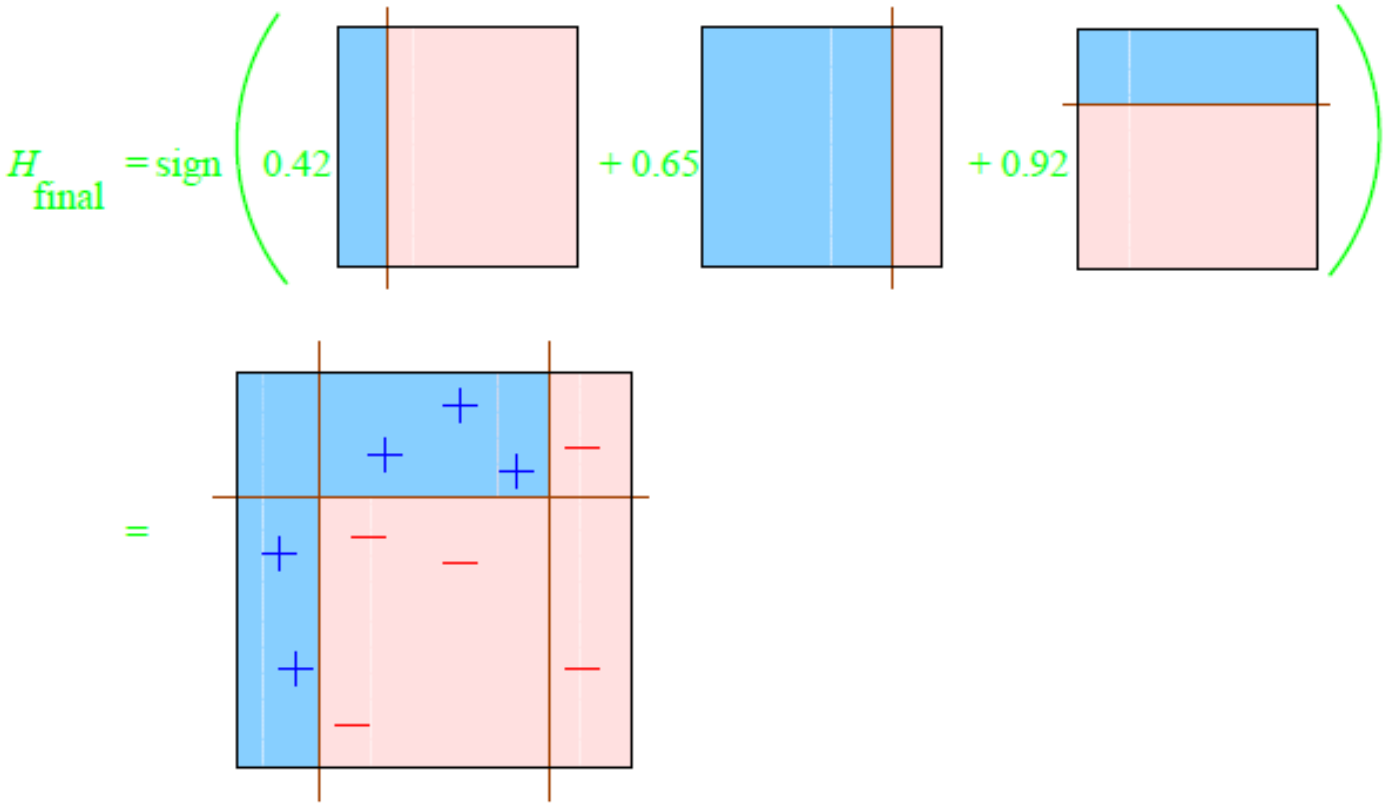


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

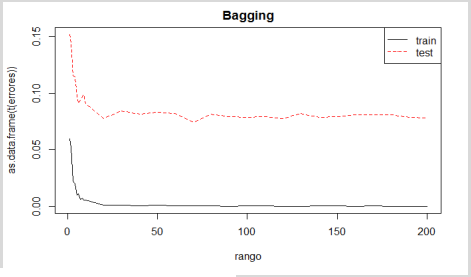
Source: https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/

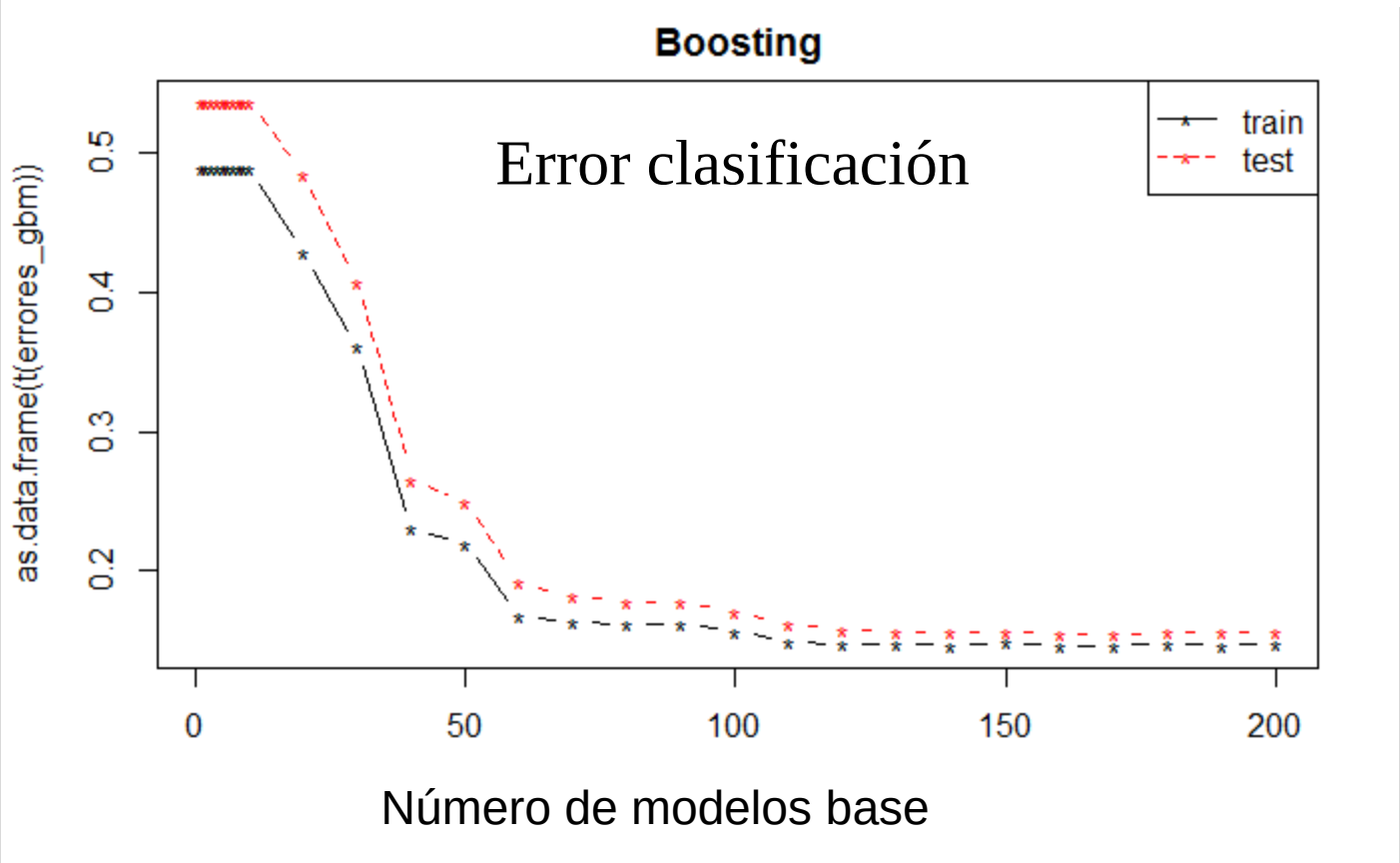
Final Hypothesis



Source: Singer and Lewis Tutorial

Boosting





GBM

Boosting

- Es uno de los mejores algoritmos de aprendizaje automático.
- Pero le puede afectar el ruido. Ya que cada modelo se centra en los fallos del anterior, si los fallos son debidos al ruido, Boosting intentará memorizarlos.

GRADIENT BOOSTING

- Adaptación de la idea de boosting para problemas de regresión (aunque también se puede utilizar para clasificación)
- En cada iteración, ajusta modelos a nuevos datos cuyas salidas son los errores del modelo anterior y la salida real

GRADIENT BOOSTING

- Supongamos que tenemos un modelo $f_0(x)$ que es capaz de aproximar la variable de respuesta y
- A los errores que comete $f_0(x)$ los llamaremos pseudo-residuos:
  - $h_0(x_i) = y_i - f_0(x_i)$
- Si construimos un modelo $h_0(x)$ que aproxime los pseudo-residuos $\{(x_1, y_1-f_0(x_1)), \ldots, (x_N, y_N-f_0(x_N))\}$
- El modelo $f_0(x) + h_0(x)$ aproximará mejor la variable de respuesta
  - En el caso óptimo $f_0(x_i) + h_0(x_i) = f_0(x_i) + y_i - f_0(x_i) = y_i$
- Repetir muchas veces

GRADIENT BOOSTING

- Repetir muchas veces:
  - $f_0(x) =$
  - $f_1(x) = f_0(x) + h_0(x)$
  - $f_2(x) = f_1(x) + h_1(x) = f_0(x) + h_0(x) + h_1(x)$
  - $f_3(x) = f_2(x) + h_2(x) = f_0(x) + h_0(x) + h_1(x) + h_2(x)$
  - …
  - $f_T(x) = f_{T-1}(x) + h_{T-1}(x) = f_0(x) + h_0(x) + \ldots + h_{T-1}(x)$

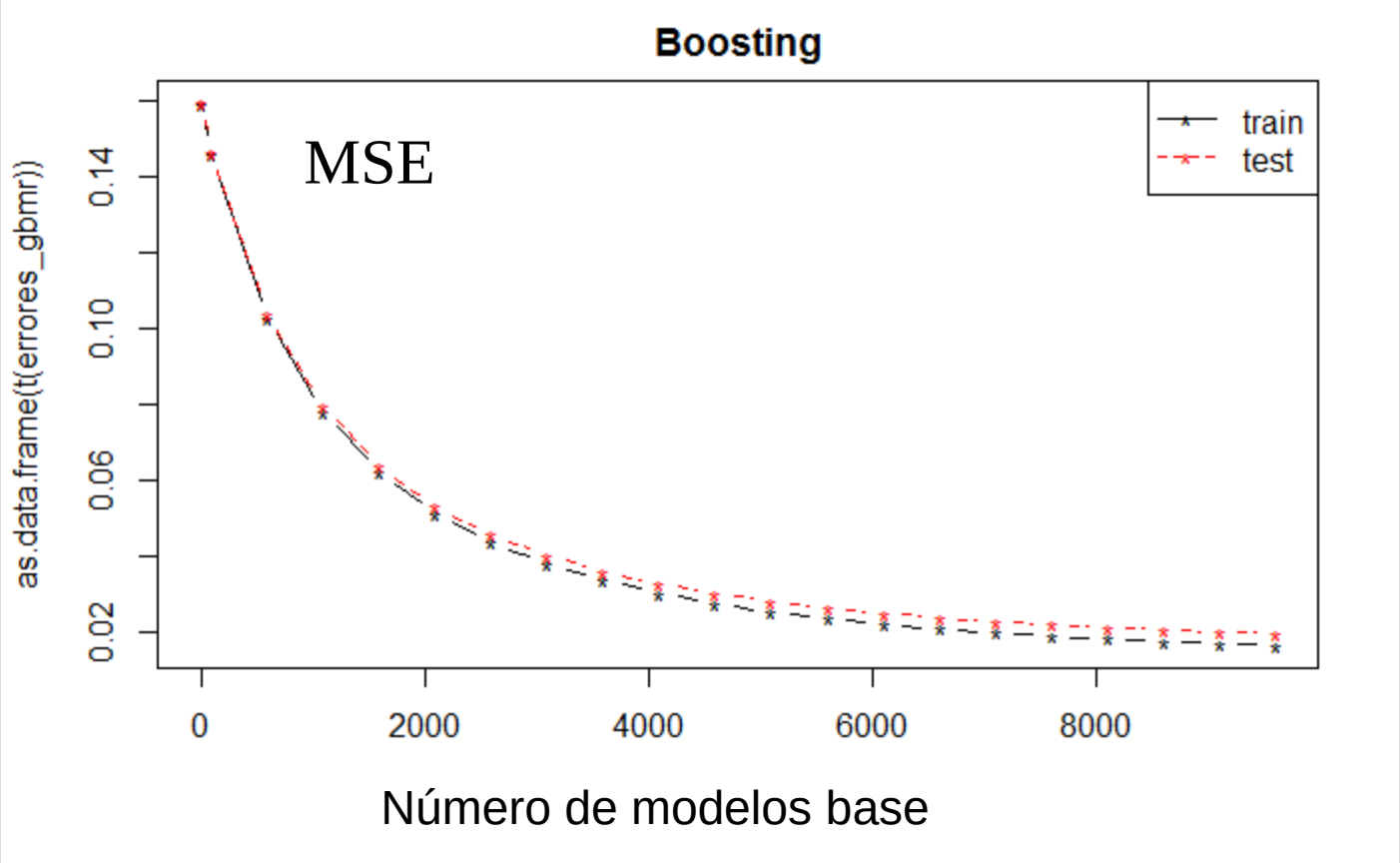
GRADIENT BOOSTING

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

1. Inicializa con $f_0(x)$ = media de las salidas
2. FOR t = 0 TO (T-1)
   - Calcula un nuevo conjunto de datos, cuya salida son los "pseudo-residuos", donde para cada (x_i, y_i) obtenemos un dato $(x_i, y_i - f_t(x_i))$
     - Donde el ensemble parcial (hasta el momento): $f_t(x) = f_0(x) + \alpha_0 h_0(x) + \ldots + \alpha_{t-1} h_{t-1}(x)$
   - Aprende un nuevo modelo $h_t(x)$ que se ajuste a los datos de los pseudo-residuos
   - Calcula α_t de manera que se optimice el error de:

     $f_t(x) + \alpha_t h_t(x)$
   - Actualiza el ensemble parcial: $f_{t+1}(x) = f_t(x) + \alpha_t h_t(x)$

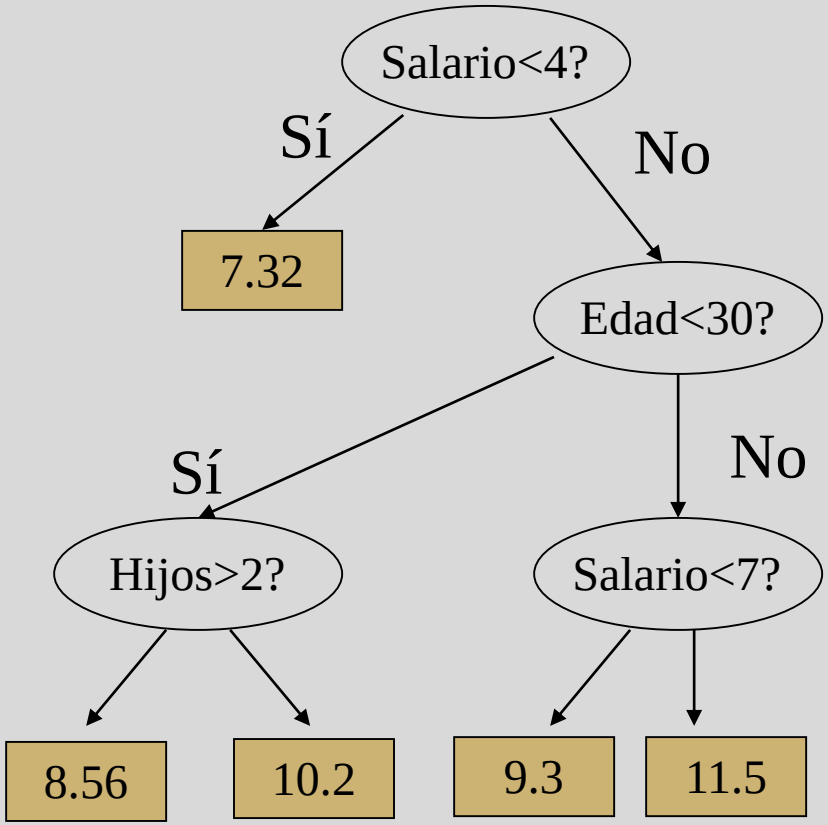
Gradient Boosting

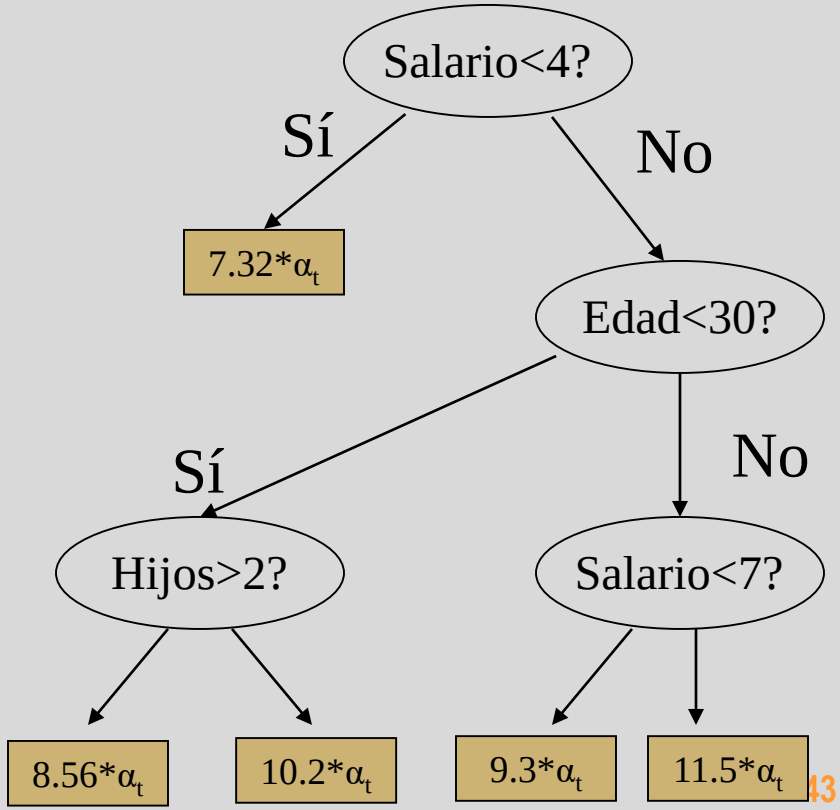


Advertencia: posible overfitting si el número de modelos base es grande

GRADIENT BOOSTED TREES

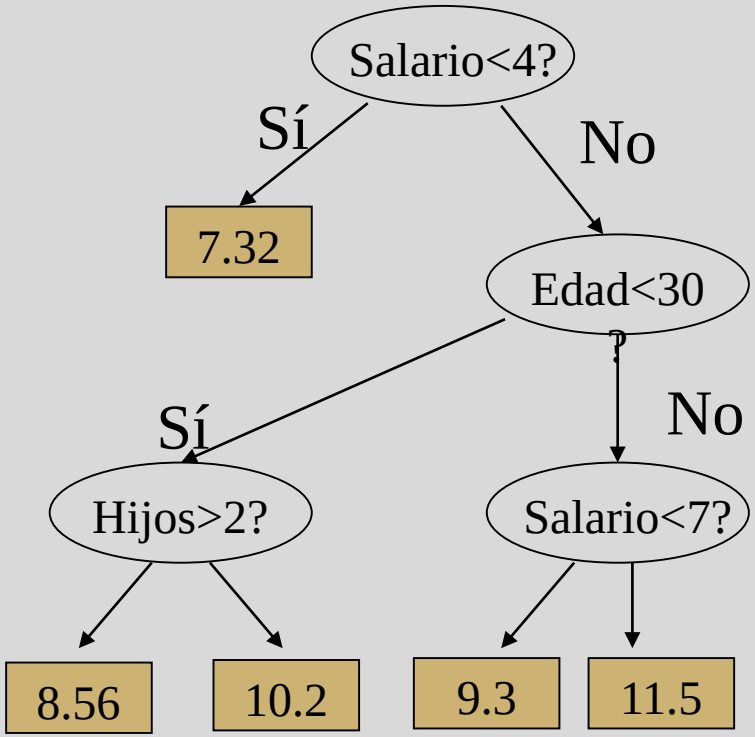
- Es igual que el algoritmo anterior, solo que en lugar de optimizar la constante α_t, que sería equivalente a que dado el árbol de la izquierda, encontrar el α_t que minimiza el error en el árbol de la derecha **...**

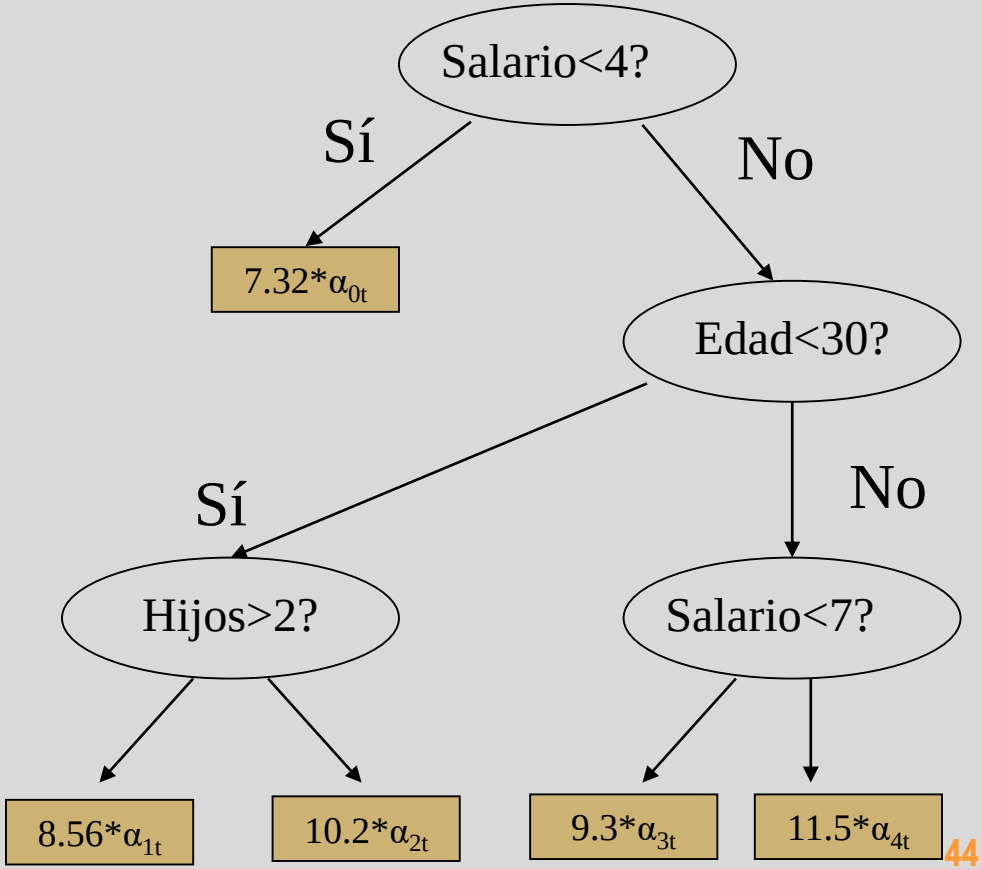




GRADIENT BOOSTED TREES

- … se optimiza un alfa distinto para cada hoja: $\alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \alpha_4$





¿Por qué se llama **Gradient** Boosting?

- L = *Loss*:
  - Cuadrática (MSE): $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$
  - Absoluta (MAE): $L(y_i, f(x_i)) = |y_i - f(x_i)|$
- J = error de regresión: media de la "pérdida" (*loss*) para todos los datos:
  - $J = \frac{1}{N}\sum_{i=1}^{i=N} L(y_i, f(x_i))$
- Supongamos que representamos el modelo f por sus valores en entrenamiento
  - O sea, $f = (f(x_1), f(x_2), \ldots, f(x_N)) = (f_1, f_2, \ldots, f_N)$
  - Es decir, vamos a considerar que f es un punto en un espacio N-dimensional
- ¿Cómo deberíamos cambiar $(f(x_1), f(x_2), \ldots, f(x_N))$ para reducir el error J?

Descenso del gradiente

- Supongamos datos $\{(x_1, 15), (x_2, 25)\}$
- Tengo un modelo f_0 que hace estas predicciones:
  - $f_0(x_1) = f_1^{(0)}$
  - $f_0(x_2) = f_2^{(0)}$
- Su MSE es:
  - $J(f_1^{(0)}, f_2^{(0)}) = L(f_1^{(0)}, 15) + L(f_2^{(0)}, 25) = \frac{1}{2}(f_1^{(0)} - 15)^2 + \frac{1}{2}(f_2^{(0)} - 25)^2$
- Concretando, sean las predicciones del modelo inicial (no muy buenas) $f^{(0)} = (f_1^{(0)}, f_2^{(0)}) = (0,0)$
- ¿En qué dirección y con qué "velocidad" deberían cambiar las predicciones del modelo?
  - O sea, ¿cómo deberían ser las predicciones del siguiente modelo $f_1 = f_0 + h_0$?

Descenso del gradiente

- El gradiente $\nabla J = (\frac{\partial J}{\partial f_1}, \frac{\partial J}{\partial f_2})$ me dice:
  - La dirección en la que el error *J* crece más rápido (y por tanto, me debería mover en la dirección contraria $-\nabla J$)
  - La magnitud con la que crece el error. Por ejemplo, podemos interpretar $\frac{\partial J}{\partial f_1}$ como "cuánto crece el error si cambio la predicción para x_1 en una unidad"

Descenso del gradiente

- En el caso de MSE (y suponiendo que comenzamos en (f_1=0, f_2=0):
  - $\frac{\partial J}{\partial f_1} \sim \frac{\Delta J}{\Delta f_1} = \frac{\frac{1}{2}(1-15)^2 - \frac{1}{2}(0-15)^2}{1-0} = -14.5$
  - $\frac{\partial J}{\partial f_2} \sim \frac{\Delta J}{\Delta f_2} = \frac{\frac{1}{2}(1-25)^2 - \frac{1}{2}(0-25)^2}{1-0} = -24.5$
- Así que la dirección en la que el MSE crece más rápido es $\nabla J = \left(\frac{\partial J}{\partial f_1}, \frac{\partial J}{\partial f_2}\right) = (-14.5, -24.5)$
- O de manera equivalente, la dirección en la que decrece más rápido es $-\nabla J = (14.5, 24.5)$

Descenso del gradiente

- En el caso de MAE (y suponiendo que comenzamos en (f_1=0, f_2=0):
  - $\frac{\partial J}{\partial f_1} \sim \frac{\Delta J}{\Delta f_1} = \frac{|1-15| \ -|0-15|}{1-0} = -1$
  - $\frac{\partial J}{\partial f_2} \sim \frac{\Delta J}{\Delta f_2} = \frac{|1-25| \ -|0-25|}{1-0} = -1$
- Así que la dirección en la que el MAE crece más rápido es $\nabla J = \left(\frac{\partial J}{\partial f_1}, \frac{\partial J}{\partial f_2}\right) = (-1, -1)$
- O de manera equivalente, la dirección en la que decrece más rápido es $-\nabla J = \ (1, 1)$

MSE vs. MAE

- Observesé que no es lo mismo optimizar MSE que MAE
- Con MSE es más importante mejorar la predicción f_2 que f_1:
  - $-\nabla J = (14.5, 24.5)$
- Con MAE, f_2 es igual de importante que f_1:
  - $-\nabla J = (1, 1)$

Descenso del gradiente

- ¿Cómo usarlo en gradient boosting?

1. Comienzo con las predicciones del modelo f_0, que son $(f_1^{(0)}, f_2^{(0)})$ (para los datos x_1 y x_2)
2. Ahora calculo el negativo del gradiente de J:
   - $-\nabla J = \left(-\frac{\partial J}{\partial f_1}, -\frac{\partial J}{\partial f_2}\right)$
3. Y esto me dice que debería actualizar las predicciones $(f_1^{(0)}, f_2^{(0)})$ siguiendo esa dirección:
   - $f_1^{(1)} = f_1^{(0)} - \alpha \frac{\partial J}{\partial f_1}$
   - $f_2^{(1)} = f_2^{(0)} - \alpha \frac{\partial J}{\partial f_2}$
   - $f_1(x) = f_0(x) - \alpha \nabla J$
     - Para algún α pequeño (para dar pasos pequeños)

4. Si llamo $h_0(x) = -\nabla J$, me sale lo que hemos contado de Boosting:
   - $f_1(x) = f_0(x) + \alpha\, h_0(x)$

Descenso del gradiente

- Los gradientes *J* (o $h_0(x)$) se pueden calcular de manera analítica, derivando:
  - Si J es el MSE, entonces:
    - $\frac{\partial J}{\partial f_i} = \frac{\partial \frac{1}{N}\sum_{j=1}^{j=N}\frac{1}{2}(f_j - y_j)^2}{\partial f_i} = \frac{\partial \frac{1}{N2}(f_i - y_i)^2}{\partial f_i} = \frac{1}{N}\frac{1}{2}2(f_i - y_i) \propto (f_i - y_i) = -h(x_i)$
    - Es decir, el modelo $f_1(x) = f_0(x) + h_1(x)$ se dirige en la dirección que minimiza el MSE
  - Si J es el MAE, entonces:
    - $\frac{\partial J}{\partial f_i} = \frac{\partial \frac{1}{N}\sum_{j=1}^{j=N}|f_j - y_j|}{\partial f_i} = \frac{\partial \frac{1}{N}|f_i - y_i|}{\partial f_i} = \frac{1}{N}\, sgn(f_i - y_i) \propto sgn(f_i - y_i) = -h(x_i)$
    - Ahora los pseudo-residuos tienen la forma: $-sgn(f_i - y_i) = h(x_i)$
    - Es decir, el modelo $f_1(x) = f_0(x) + h_1(x)$ se dirige en la dirección que minimiza el MAE
    - Recordemos que h1 es un modelo que se ajusta al conjunto de datos $\{(x_1, \text{sgn}(f_0(x_1)-y_1), \ldots, (x_N, \text{sgn}(f_0(x_N)-y_N)\}$

Gradient Boosting para clasificación

- Adaboost lo hacía dándole pesos a cada instancia
- Gradient Boosting es para regresión, pero las clases son categóricas (discretas)
- Pero podemos convertir un problema de clasificación en uno de regresión, ajustando la probabilidad de la clase (y no la clase en sí)
- Si hay varias clases, aprendemos un ensemble para cada clase (dicho de otra manera, codificamos la clase como si fuera one-hot-encoding)

GBT y sobreaprendizaje

- Controlar el número de árboles T en el modelo (T pequeño => menos sobreaprendizaje)
- Shrinkage (0<v<1) o learning rate: $f_t(x) = f_{t-1}(x)+v*\alpha_t h_t(x)$
  - Esto consigue que el nuevo modelo tenga menos peso del que tendría y por tanto sobreajusta menos a los datos
  - v pequeños suele implicar que haya que poner más árboles (T) en el ensemble, y por tanto, el aprendizaje tarda mas
- Controlar profundidad de los árboles (modelos base). Cuanto más pequeña, menos sobreajuste. Valores típicos entre 4 y 8
- Stochastic gradient boosting (siguiente transparencia)

Stochastic Gradient Boosting

- En cada iteración no se usa el conjunto completo de datos sino una submuestra más pequeña, obtenida de los datos originales mediante muestreo SIN reemplazo
- Ejemplo: con porcentaje de submuestra 0.8, se usan muestras con un 80% de los datos originales obtenidos aleatoriamente (pero sin que se repitan los datos)
- Impide el sobreajuste (porque los datos son distintos cada vez)
- Permite hacer una estimación out-of-bag
- Otra manera de submuestrear es el submuestreo de columnas (similar a Random Forests)

Varias implementaciones

- Gbm:
  - Minimización de MSE: “distribución gausiana”
  - Minimización de MAE: “distribución de Laplace”
  - Para clasificación, “distribución de Bernouilli”

- xgboost:
  - Utiliza la derivada segunda de la función de coste
  - Regularizado
  - Paralelo, distribuido, etc. (rápido)