Ensamble Learning - Bagging, Random Forests, Boosting

Germán Rosati

IDAES/UNSAM - CONICET - PIMSA

30 de Junio de 2022

Ensambles de modelos

- Técnicas de aprendizaje supervisado donde se combinan varios modelos base.

- Combinando varios modelos base se busca ampliar el espacio de hipótesis posibles para representar los datos, con el fin de mejorar la precisión predictiva del modelo combinado resultante.

- Mucho más precisos que los modelos base que los componen.

Ensambles de modelos

Dos familias

- **Métodos de averaging** (basados en promedios), que consisten en construir varios estimadores de forma independiente y luego hacer un promedio de sus predicciones. El modelo resultante de la combinación, suele ser mejor que cualquier estimador base separado.

    - Ejemplos de esta familia son los métodos de Bagging y su implementación particular, Random Forest.

Ensambles de modelos

Dos familias

- **Métodos de boosting**, donde los estimadores base se construyen secuencialmente y uno trata de reducir el sesgo del estimador combinado, centrándose en aquellos casos en los que se observa una peor performance. La idea es combinar varios modelos débiles para producir un ensamble potente.

    - Ejemplos de esta familia son AdaBoost y Gradient Tree Boosting.

Espacio de hipótesis

- Aprendizaje supervisado: hacer predicciones de la verdadera función de clasificación f aprendiendo el clasificador h.

- Buscamos en un cierto espacio de hipótesis H la función más apropiada para describir la relación entre nuestras características y el objetivo.

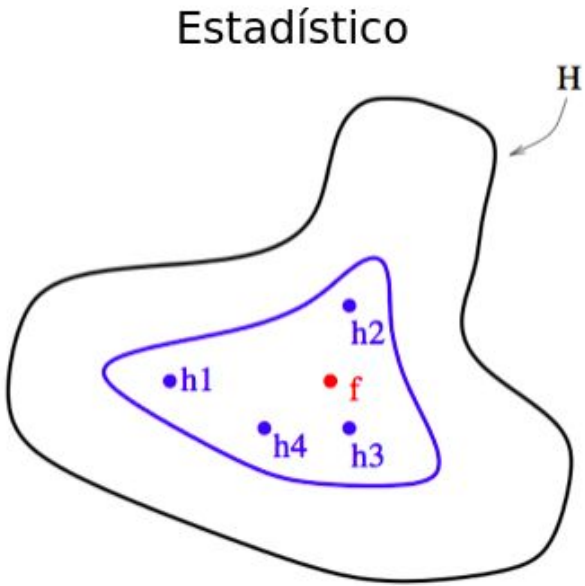
- Puede haber varias razones por las cuales un clasificador base no pueda lograr mayor exactitud al tratar de aproximar la función de clasificación real.

Espacio de hipótesis

- Estos son tres de los posibles problemas:
    - Estadísticos
    - Computacionales
    - De representación

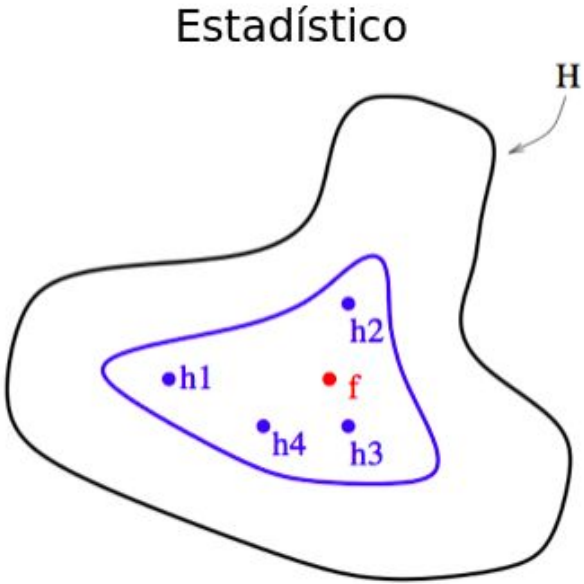
Problema estadístico

- Si la cantidad de datos de entrenamiento disponibles es pequeña, el clasificador base tendrá dificultades para converger a f .
- Un ensamble puede mitigar este problema "promediando" las predicciones de los clasificadores.
- La función real f es mejor aproximada como un promedio de los clasificadores base hi.



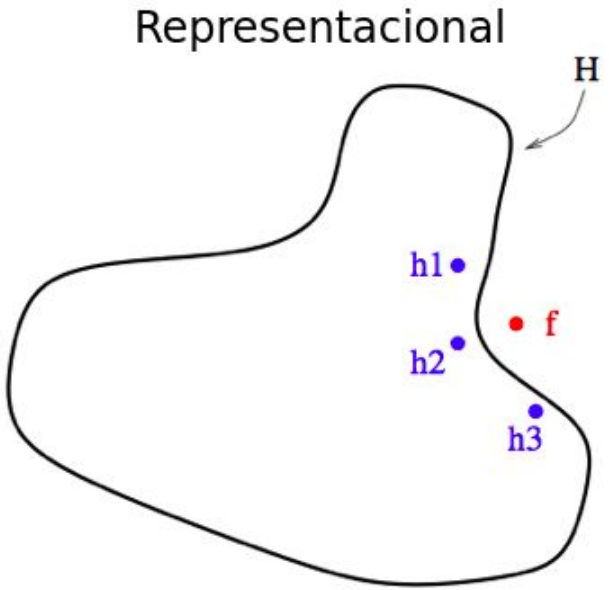
Problema computacional

- Puede ser computacionalmente difícil encontrar el mejor clasificador h.
- Imposible una búsqueda exhaustiva del espacio de hipótesis de todos los posibles clasificadores
- Un conjunto de varios clasificadores base con diferentes puntos de partida aproximar mejor f que cualquier clasificador base individualmente.



Problema de representación

- A veces f no se puede expresar en términos de la hipótesis.
- Si usamos un árbol de decisión como clasificador base, este trabaja formando particiones rectilíneas del espacio de características.
- Pero si f es una línea diagonal, entonces no puede ser representada por un número finito de segmentos rectilíneos. Por lo tanto, el límite de decisión verdadero, no puede ser expresado por un árbol de decisión.



Condiciones de aplicación

- Capacidad predictiva: los clasificadores base deben hacer mejores predicciones que la totalmente aleatoria. (Su AUC debe ser mayor a 0.5)

- Diversidad: los clasificadores base deben cometer distintos errores ante los mismos casos. (Sin diversidad no se puede mejorar la precisión del ensamble al combinar los clasificadores base)

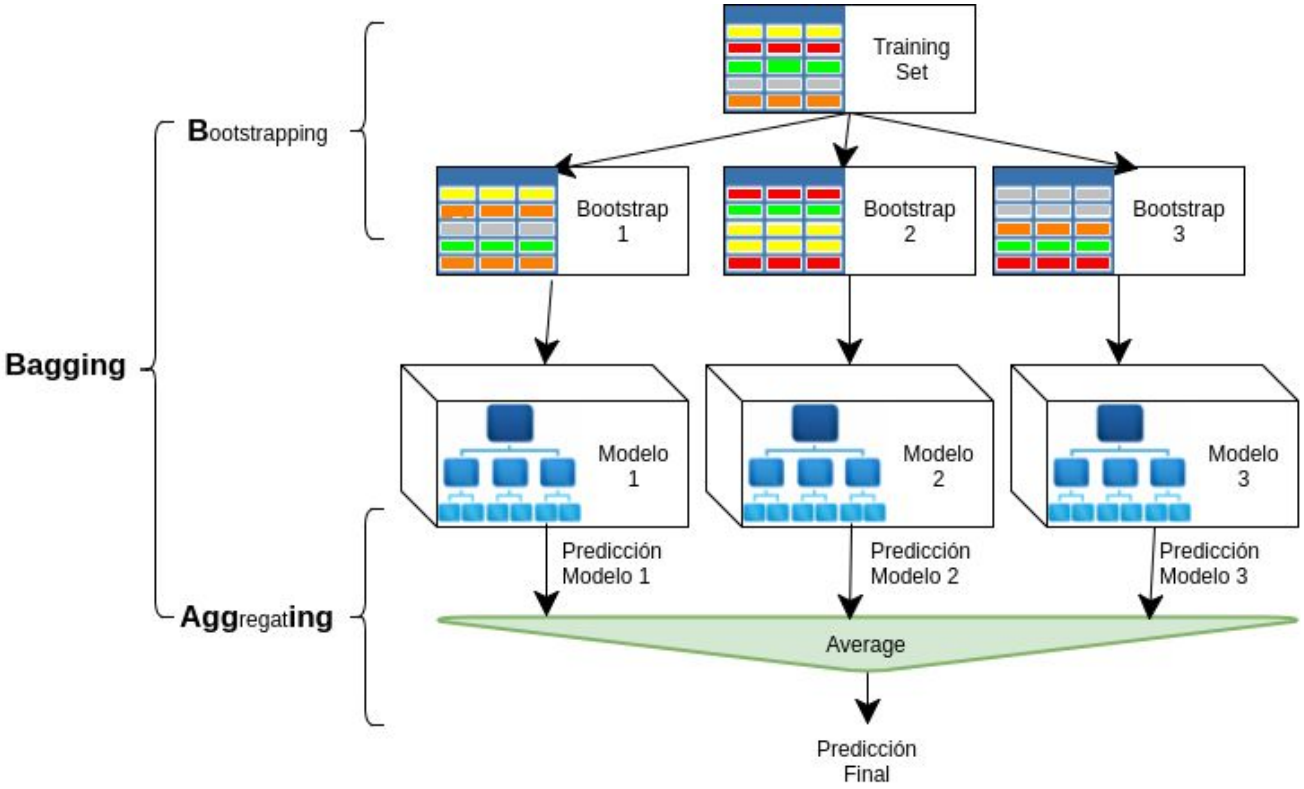
Bagging

- El Bagging reduce la varianza del error de generalización al combinar múltiples clasificadores de base (siempre que estos satisfagan los requisitos anteriores).

- Si el clasificador base es estable, entonces el error del ensamble se debe principalmente al sesgo, y el bagging puede no ser efectivo.

- Dado que cada muestra de datos de entrenamiento es igualmente probable, el bagging no es muy susceptible a overfitting con datos ruidosos.

Bagging


Bagging
Bootstrapping
Aggregating
Training Set
Bootstrap 1
Bootstrap 2
Bootstrap 3
Modelo 1
Modelo 2
Modelo 3
Predicción Modelo 1
Predicción Modelo 2
Predicción Modelo 3
Average
Predicción Final

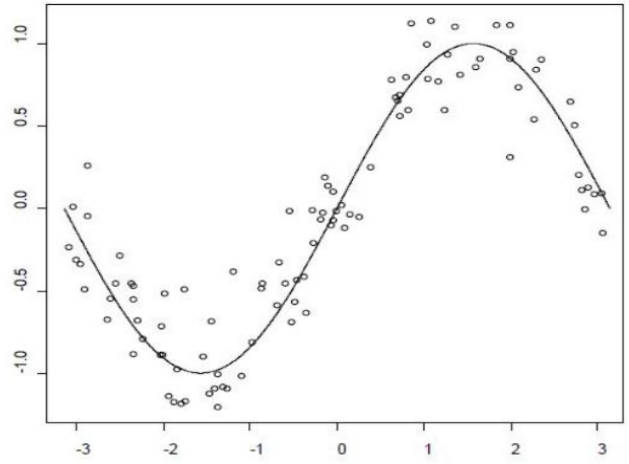
Bagging

Algorithm 5.6 Bagging Algorithm

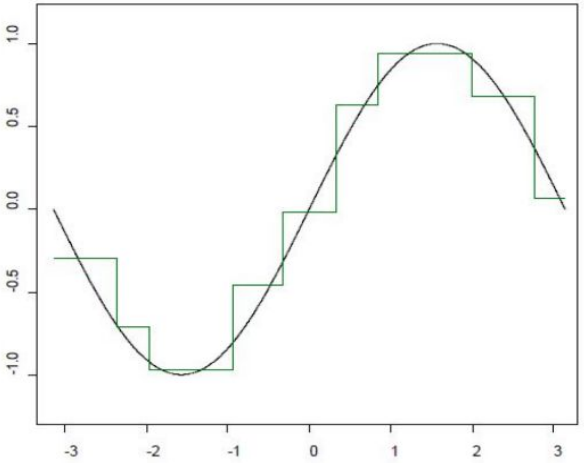
1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

Bagging

Función generadora

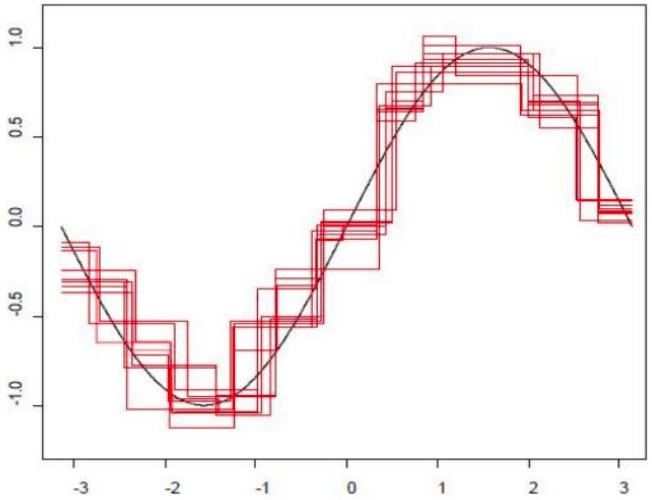


Un árbol de decisión

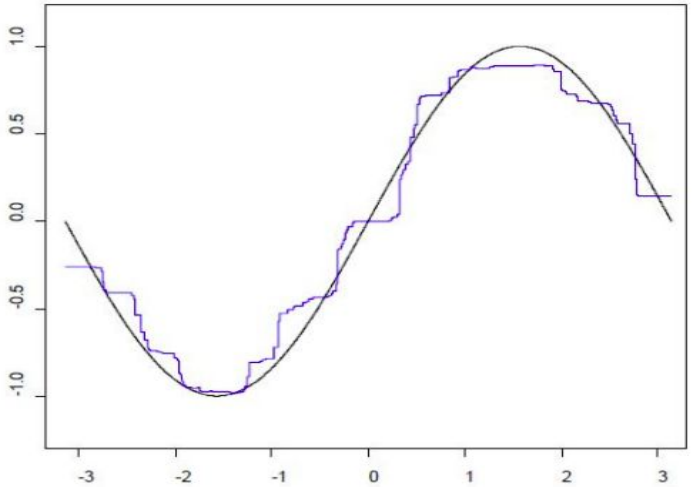


Bagging

10 árboles de decisión



Promedio de 10 árboles de decisión



Random Forest

- Random forest es muy similar a un bagging de árboles de decisión
- La diferencia: además de generar variabilidad sobre los registros, se genera variabilidad sobre los predictores.
- Bagging genera B predicciones a partir de B remuestras bootstrap del TrS original y de M predictores del TrS original
- De esta forma, en bagging entran el total de los M predictores.

Random Forest

- Esto puede generar árboles muy correlacionados... ¿por qué?

- Si una o algunas variables son predictores muy fuertes para la variable target, estas variables serán seleccionadas en muchos de los árboles base del bagging, haciendo que queden correlacionados.

- Seleccionando un subconjunto aleatorio de las variables en cada división, contrarrestamos esta correlación entre los árboles base, fortaleciendo el modelo final.

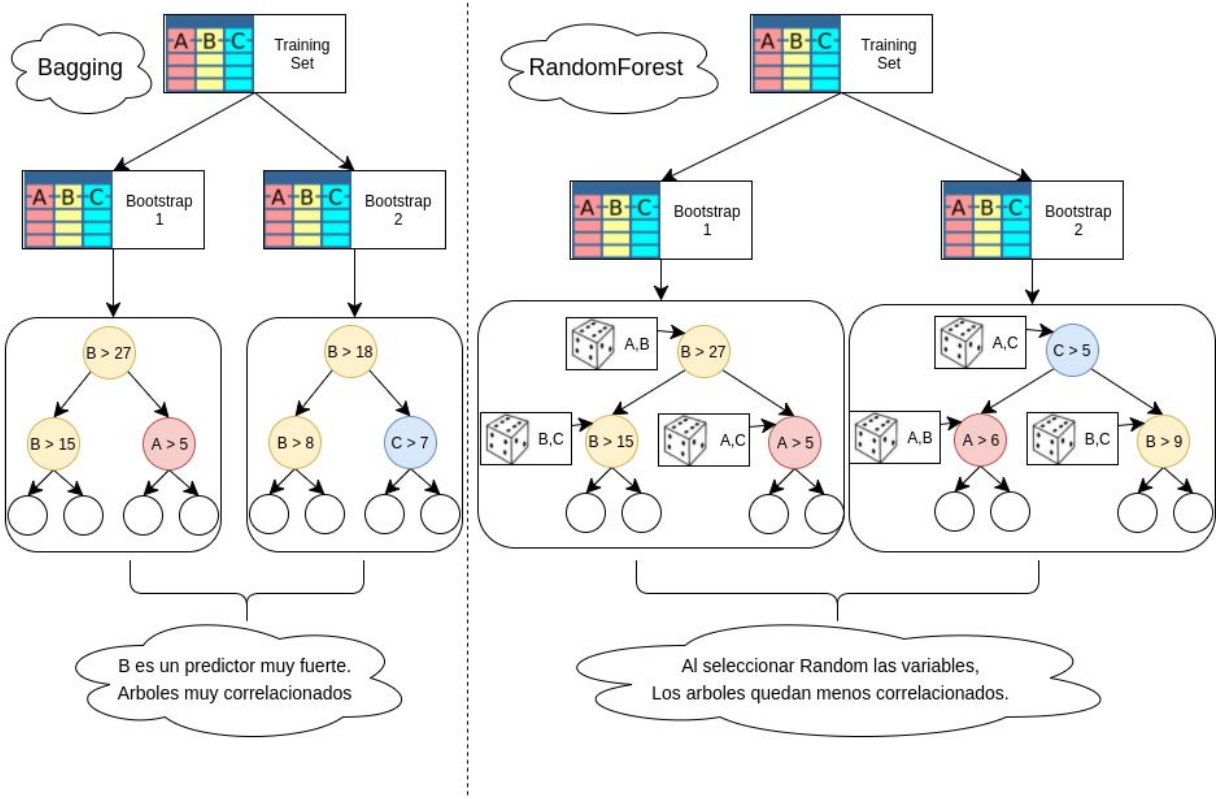
Random Forest

- Para un problema de clasificación con p variables, se suelen utilizar √p de las variables en cada división.

- Para problemas de regresión, recomiendan utilizar p/3.

- Pero también podría considerarse como un hiperparámetro para tunear.

Random Forest



Random Forest

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

Extra randomized trees

- Como Random Forest ordinario, pero con una capa random adicional.

- En lugar de calcular la combinación variable/división óptima local (ej ganancia de información), para cada variable en consideración se generan una división aleatoria (dentro del rango de la variable). Y luego se selecciona la variable/división que maximice la ganancia.

- La diferencia principal es que la división para cada variable no será la óptima, sino una seleccionada aleatoriamente.

Split_a_node(S)

Input: the local learning subset S corresponding to the node we

Output: a split $[a < a_c]$ or nothing

– If **Stop_split**(S) is TRUE then return nothing.

– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;

– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;

– Return a split s_* such that $Score(s_*, S) = \max_{i=1,\ldots,K} Score(s_i, S)$.

Genera una división candidata para cada una de las K variables

Pick_a_random_split(S,a)

Inputs: a subset S and an attribute a

Output: a split

– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;

– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;

– Return the split $[a < a_c]$.

La división se genera con un random en el rango de la variable

Stop_split(S)

Input: a subset S

Output: a boolean

– If $|S| < n_{\min}$, then return TRUE;

– If all attributes are constant in S, then return TRUE;

– If the output is constant in S, then return TRUE;

– Otherwise, return FALSE.

Split_a_node(S)

Input: the local learning subset S corresponding to the node we want to split

Output: a split $[a < a_c]$ or nothing

– If **Stop_split**(S) is TRUE then return nothing.

– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;

– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = $ **Pick_a_random_split**$(S, a_i), \forall i = 1, \ldots, K$;

– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Selecciona K variables igual que Random Forest

Pick_a_random_split(S,a)

Inputs: a subset S and an attribute a

Output: a split

– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;

– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;

– Return the split $[a < a_c]$.

Stop_split(S)

Input: a subset S

Output: a boolean

– If $|S| < n_{\min}$, then return TRUE;

– If all attributes are constant in S, then return TRUE;

– If the output is constant in S, then return TRUE;

– Otherwise, return FALSE.

Split_a_node(S)

Input: the local learning subset S corresponding to the node we want to split

Output: a split $[a < a_c]$ or nothing

– If **Stop_split**(S) is TRUE then return nothing.

– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;

– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = $ **Pick_a_random_split**$(S, a_i), \forall i = 1, \ldots, K$;

– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Retorna la división que obtenga el máximo Score.

Pick_a_random_split(S,a)

Inputs: a subset S and an attribute a

Output: a split

– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;

– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;

– Return the split $[a < a_c]$.

Stop_split(S)

Input: a subset S

Output: a boolean

– If $|S| < n_{\min}$, then return TRUE;

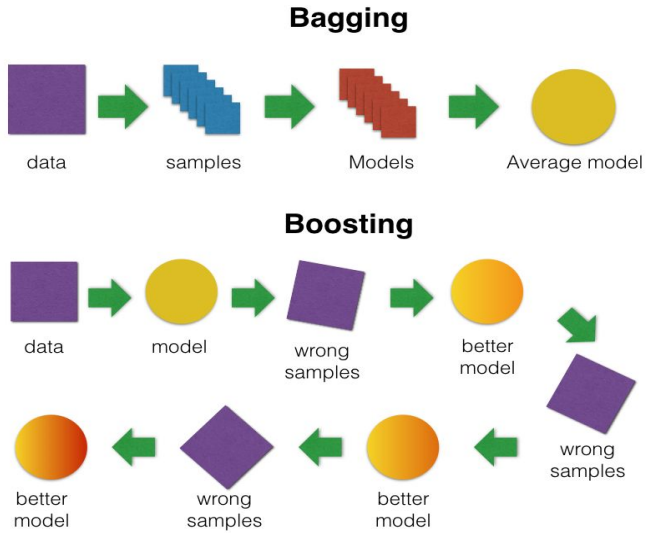
– If all attributes are constant in S, then return TRUE;

– If the output is constant in S, then return TRUE;

– Otherwise, return FALSE.

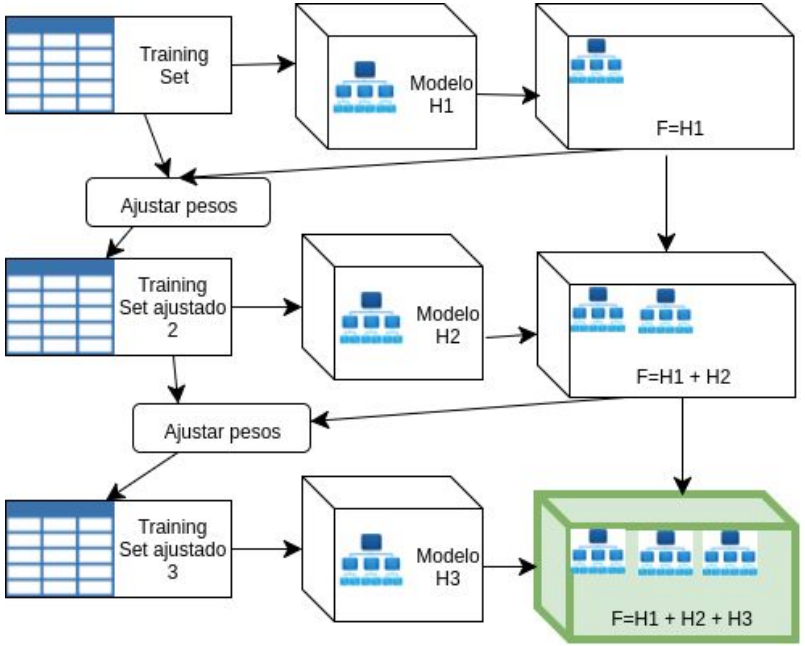
Introducción

- Bagging y Random Forests: modelos en subsets separados y luego combinamos su predicción
- Paralelizando el entrenamiento y combinando los resultados
- El Boosting es otra técnica de ensamble la cual es secuencial



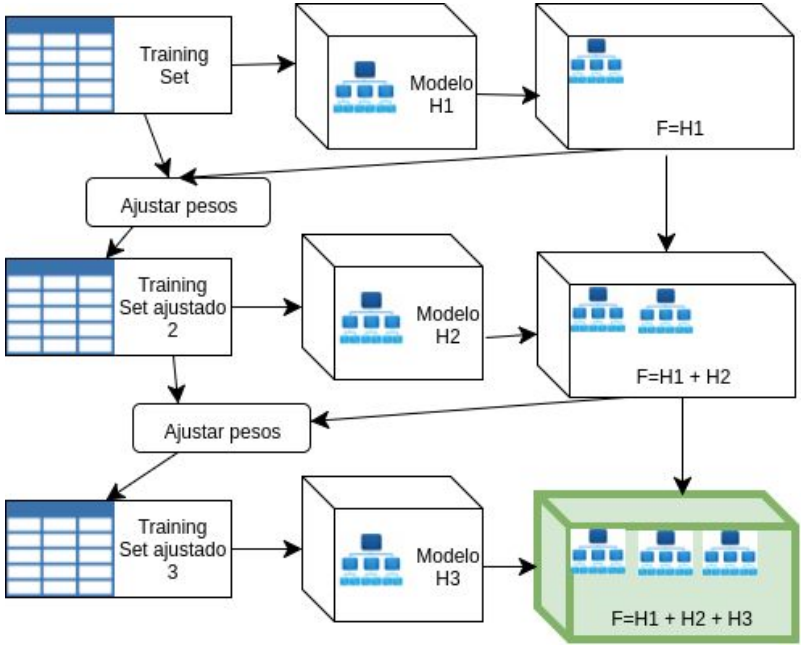
Boosting

- Meta-algoritmo: procedimiento iterativo => el modelo final se construye por pasos
- Aprender de los errores cometidos en los pasos previos.
- Sobre los errores del modelo anterior:
    - cambiar la ponderación en el siguiente modelo
    - entrenando un modelo que prediga los mismos.



AdaBoost

- 1 iteración: pesos uniformes para todos los registros. Luego, los pesos se ajustan para enfatizar los errores en la iteración anterior
- Predicción final: voto ponderado según cada error de entrenamiento, de los distintos modelos base
- Modelo base débil => re-entrenarlo en las muestras mal clasificadas.



Algoritmo AdaBoost.M1

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Se inicializan todos los pesos iguales.
Habrá un peso Wi asociado a cada uno de los ejemplos Xi del set de entrenamiento. Siendo N la cantidad de ejemplos en el set de entrenamiento

Algoritmo AdaBoost.M1

El algoritmo entrenará M clasificadores.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Algoritmo AdaBoost.M1

Se entrena el clasificador Gm, considerando el set de entrenamiento y el peso wi asignado a cada uno de los ejemplos.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Algoritmo AdaBoost.M1

Se calcula el error de clasificación ponderado de Gm.
ERRm será la suma del peso de los ejemplos mal clasificados / suma todos los pesos
Mínimo de 0 cuando no haya errores.
Máximo de 1 cuando sean todos errores.
Se puede ver que los ejemplos de alto peso mal clasificados influyen más que los de pesos bajos.

1. Initialize th

2. For $m = 1$

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.

   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.

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

Algoritmo AdaBoost.M1

Se calcula el coeficiente de aporte de este Clasificador en el ensamble. El valor será mayor cuanto más preciso sea el clasificador Gm, dándole mayor importancia a su voto en el comité.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Algoritmo AdaBoost.M1

Se calcula el coeficiente de aporte de este Clasificador en el ensamble. El valor será mayor cuanto más preciso sea el clasificador Gm, dándole mayor importancia a su voto en el comité.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

	err	(1-err) / err	**log((1-err) / err)**
Buen clasificador	0.01	99	**1.99**
Azar	0.5	1	**0**
Mal clasificador	0.99	0.01	**-2**

Algoritmo AdaBoost.M1

Se calcula el coeficiente de aporte de este Clasificador en el ensamble. El valor será mayor cuanto más preciso sea el clasificador Gm, dándole mayor importancia a su voto en el comité.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots,$
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

	err	(1-err) / err	**log((1-err) / err)**
Buen clasificador	0.01	99	**1.99**
Azar	0.5	1	**0**
Mal clasificador	0.99	0.01	**-2**

Observar que este coeficiente es el que determina el peso del voto de este clasificador en el comité resultante

Algoritmo AdaBoost.M1

Se recalculan los pesos de los ejemplos del set de entrenamiento. Aumentando los pesos de aquellos ejemplos mal clasificados.

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the trainin
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Existen variaciones de este algoritmo donde además se disminuye el peso de los ejemplos bien clasificados. Y se agrega un paso posterior de normalización de los pesos.

Algoritmo AdaBoost.M1

Se recalculan los pesos de los ejemplos del set de entrenamiento. Aumentando los pesos de aquellos ejemplos mal clasificados.

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the trainin
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Existen variaciones de este algoritmo donde además se disminuye el peso de los ejemplos bien clasificados. Y se agrega un paso posterior de normalización de los pesos.

Algoritmo AdaBoost.M1

Se obtiene como resultado el ensamble G(x) donde cada Gm(x) hace su aporte con su voto ponderado por su coeficiente Am.

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Gradient Boosting

- El Gradient Boosting es una generalización de boosting para funciones de pérdida diferenciables. Es un procedimiento preciso y efectivo que se puede usar para problemas de regresión y clasificación.

- Modelos de Gradient Boosting de árboles se utilizan en una variedad de áreas, incluyendo ranking de búsqueda web, ecología, etc.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss fu

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Inicializamos el modelo con un valor constante.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

Para cada iteración (m=1 to M) calculamos los residuos.

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

Para cada iteración (m=1 to M) fiteamos un modelo (por ejemplo, un árbol de decisión) sobre los residuos sobre el training set

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

   1. Compute so-called *pse*

$$r_{im} = -\left[\frac{\partial L(y_i, }{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Este es el pase mágico… lo que se busca es encontrar el valor de gamma, que permite calcular la contribución de cada modelo.

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

Actualizamos el modelo agregando el learner nuevo a la predicción

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1} L(y_i, \gamma).$$

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

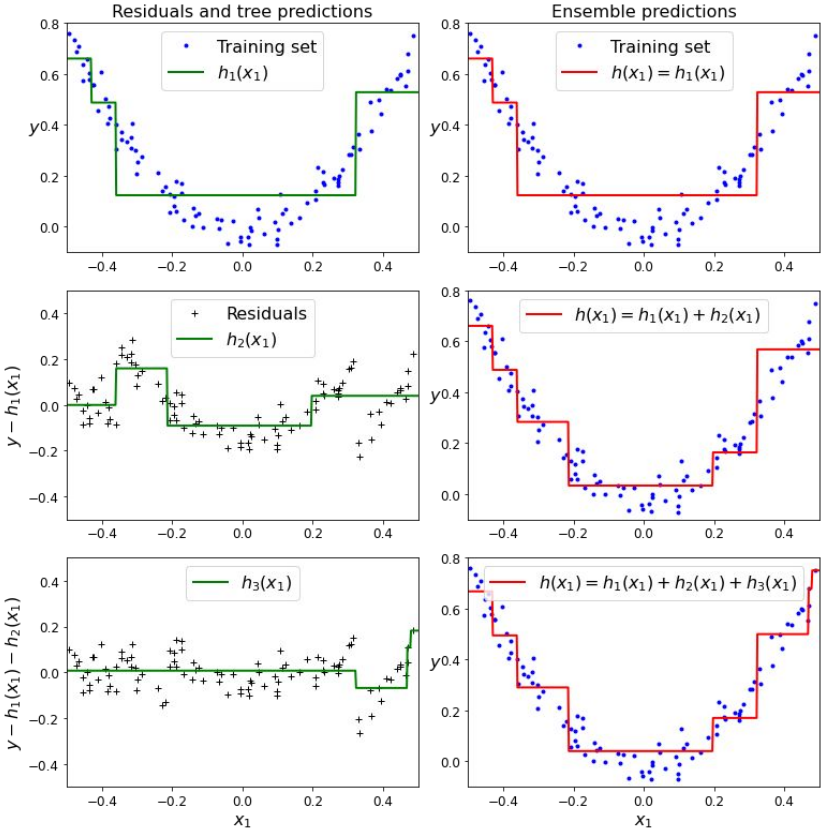
   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

- Es más o menos equivalente a fitear un modelo de forma secuencial sobre los residuos del anterior.
- Versión Python https://github.com/ageron/handson-ml2/blob/master/07_ensemble_learning_and_random_forests.ipynb
- Versión R

  https://gefero.github.io/flacso_ml/clase_3/notebook/boosting_intuicion_notebook.nb.html



Síntesis

- Ensambles: herramientas potentes
- Uso de la aleatoriedad para incrementar la capacidad del modelo
- Bagging = Bootstrap Aggregating
- Random Forest = Bagging + random selection de features
- Extra Randomized Trees: Random Forest + random splits
- Boosting: entrenamiento secuencial