

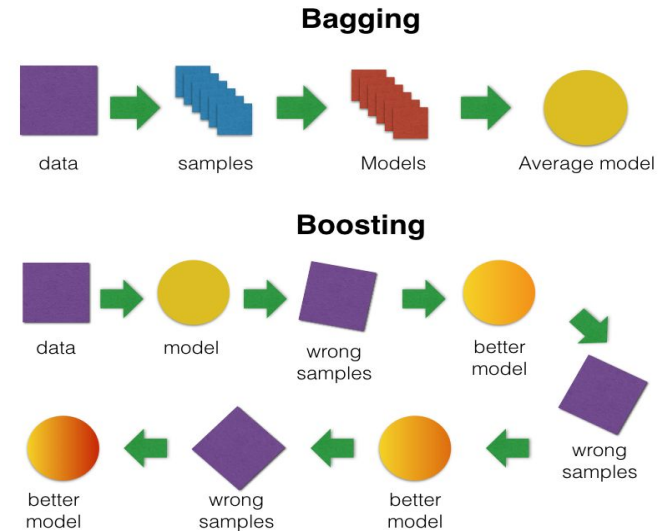
Ensamble Learning II - Boosting

Diplomatura en Ciencias Sociales Computacionales
y Humanidades Digitales

EIDAES/UNSAM

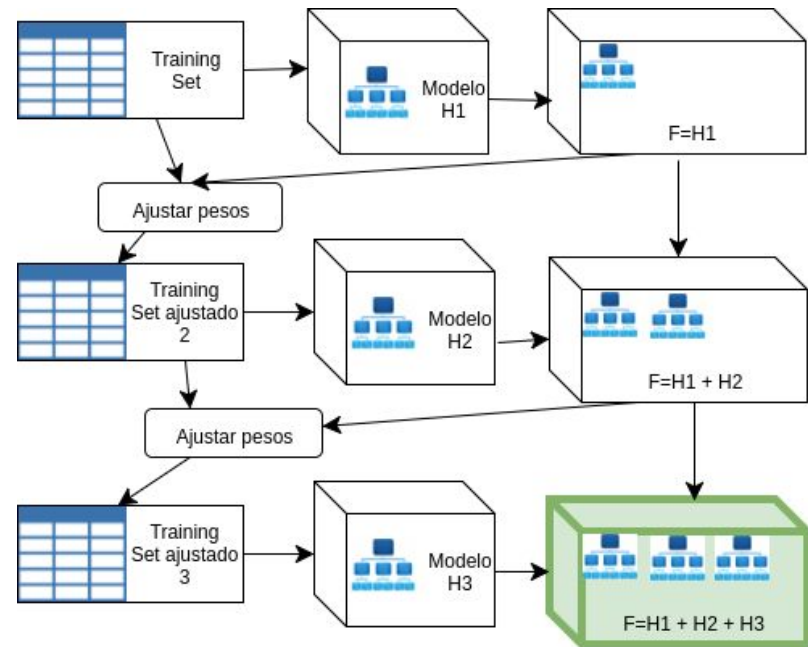
Introducción

- 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



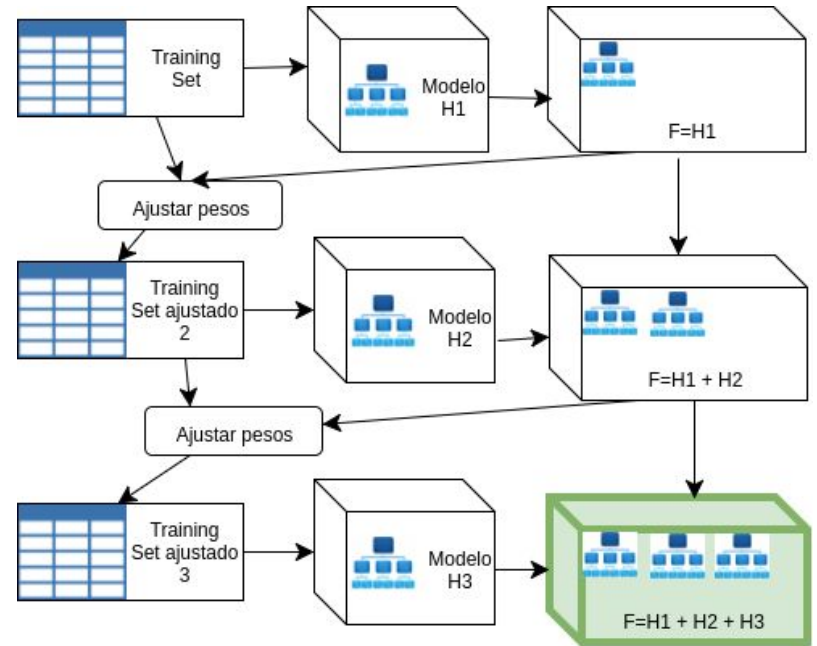
Boosting

- 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 $w_i = 1/N$, $i = 1, 2, \dots, N$.
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 $\alpha_m = \log((1 - \text{err}_m) / \text{err}_m)$.
 - (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots, N$.

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

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

Algoritmo AdaBoost.M1

El algoritmo entrenará M clasificadores.

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.

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

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

Algoritmo AdaBoost.M1

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

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.

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

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 G_m .

ERR $_m$ 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 the weights w_i .

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

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 G_m , dándole mayor importancia a su voto en el comité.

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
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 $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots$.
3. Output $G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right]$.

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 $w_i = 1/N$, $i = 1, 2, \dots, N$.

2. For $m = 1$ to M :

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

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

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 $G_m(x)$ hace su aporte con su voto ponderado por su coeficiente α_m .

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
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 $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots, N$.
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*:

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

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:

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

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

Inicializamos el modelo con un valor constante.

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*:

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

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:

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

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 *pseudo-residuals*:

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

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:

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

4. Update the model:

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

3. Output $F_M(x)$.

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

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*:

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

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:

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

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 pseudo-residuals

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

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

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:

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

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:

Actualizamos el modelo agregando el learner nuevo a la predicción

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

2. For $m = 1$ to M :

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

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

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:

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

4. Update the model:

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

3. Output $F_M(x)$.



Síntesis

- Ensembles: 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