

# Ensamble Learning I - Bagging y Random Forests

Germán Rosati

IDAES/UNSAM - CONICET - PIMSA

01 de Noviembre de 2019

# 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.

# Ensamblados 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.

# Ensamblados 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

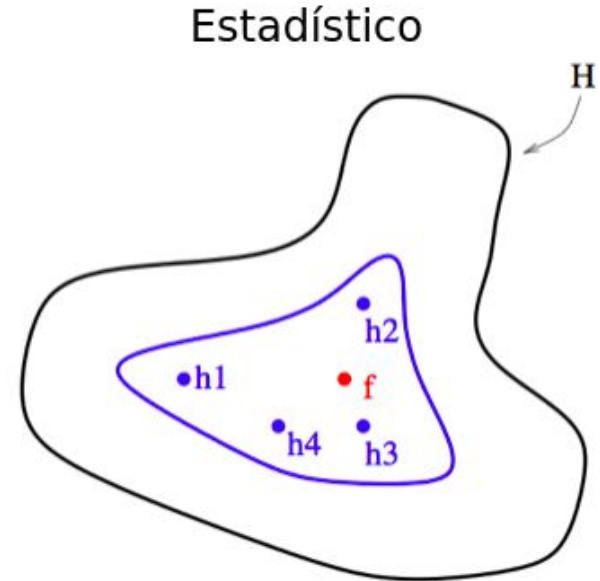
- Aprendizaje supervisado: hacer predicciones de la verdadera función de clasificación  $f$  aprendiendo el clasificador  $h$ .
- Buscamos en un cierto espacio de hipótesis  $H$  la función más apropiada para describir la relación entre nuestras características y el objetivo.
- Puede haber varias razones 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

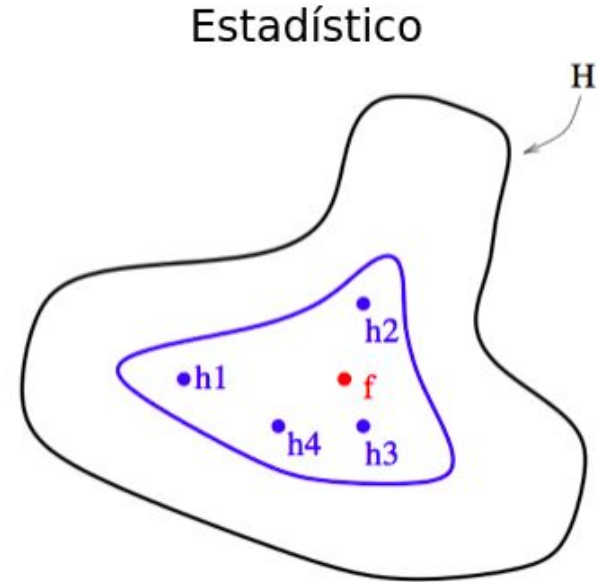
# Problema estadístico

- 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  $h_i$ .



# Problema computacional

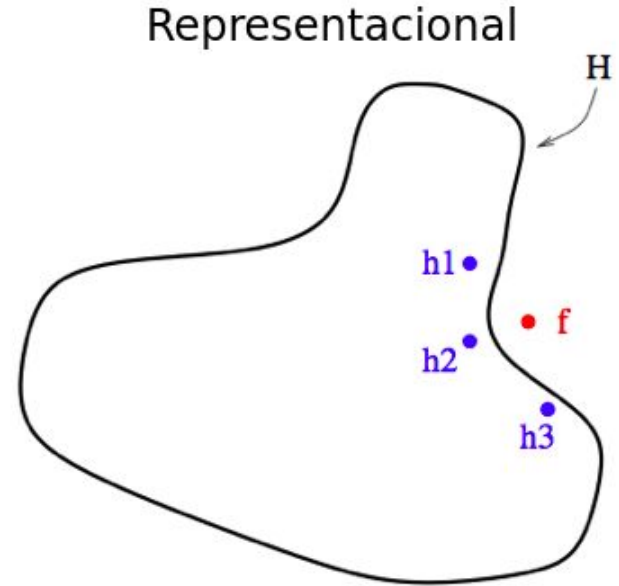
- Puede ser computacionalmente difícil encontrar el mejor clasificador  $h$ .
- Imposible una búsqueda exhaustiva del espacio de hipótesis de todos los posibles clasificadores
- Un conjunto de varios clasificadores base con diferentes puntos de partida aproximar mejor  $f$  que cualquier clasificador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



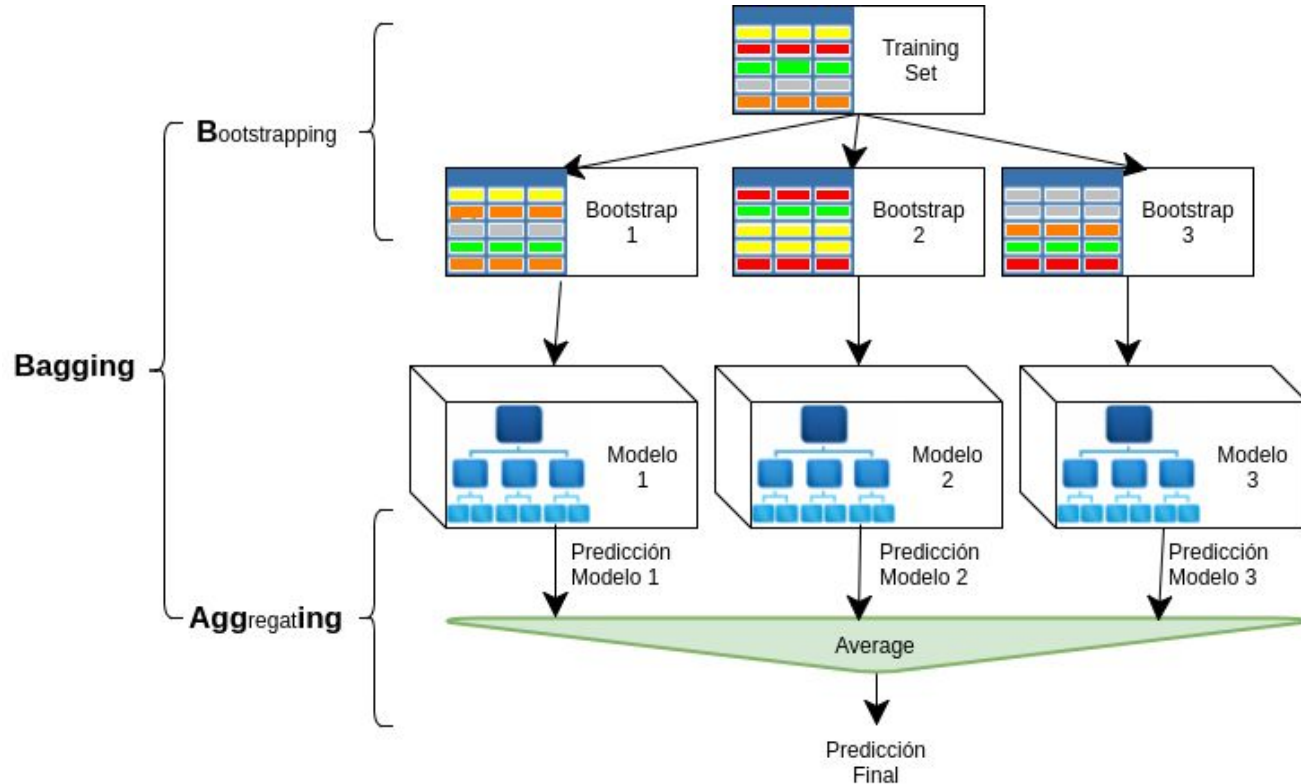
# Condiciones de aplicación

- 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

---

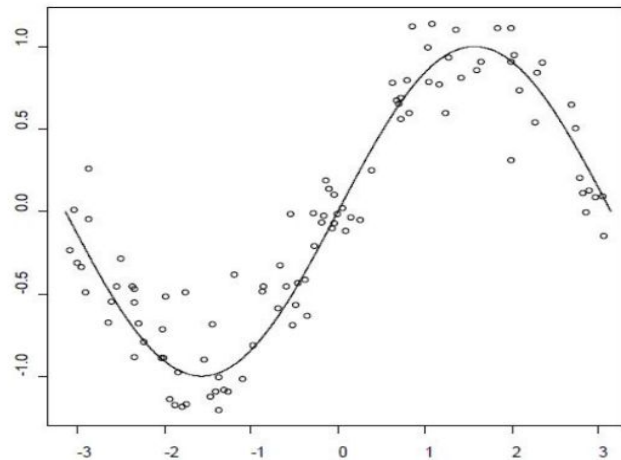
## Algorithm 5.6 Bagging Algorithm

---

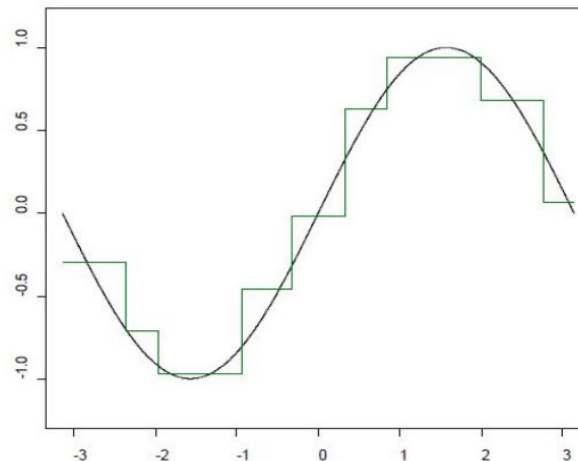
- 1: Let  $k$  be the number of bootstrap samples.
  - 2: **for**  $i = 1$  to  $k$  **do**
  - 3:   Create a bootstrap sample of size  $n$ ,  $D_i$ .
  - 4:   Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .
  - 5: **end for**
  - 6:  $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$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

# Bagging

**Función generadora**

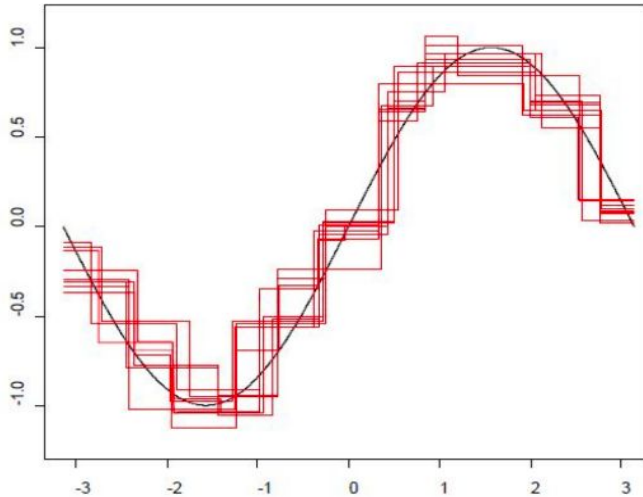


**Un árbol de decisión**

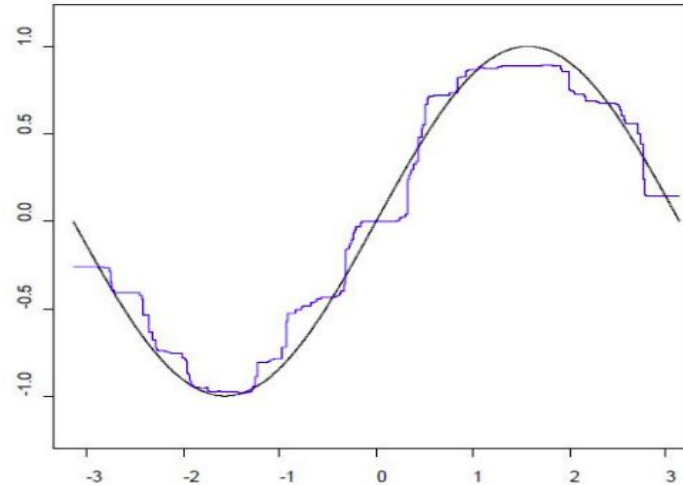


# Bagging

**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.



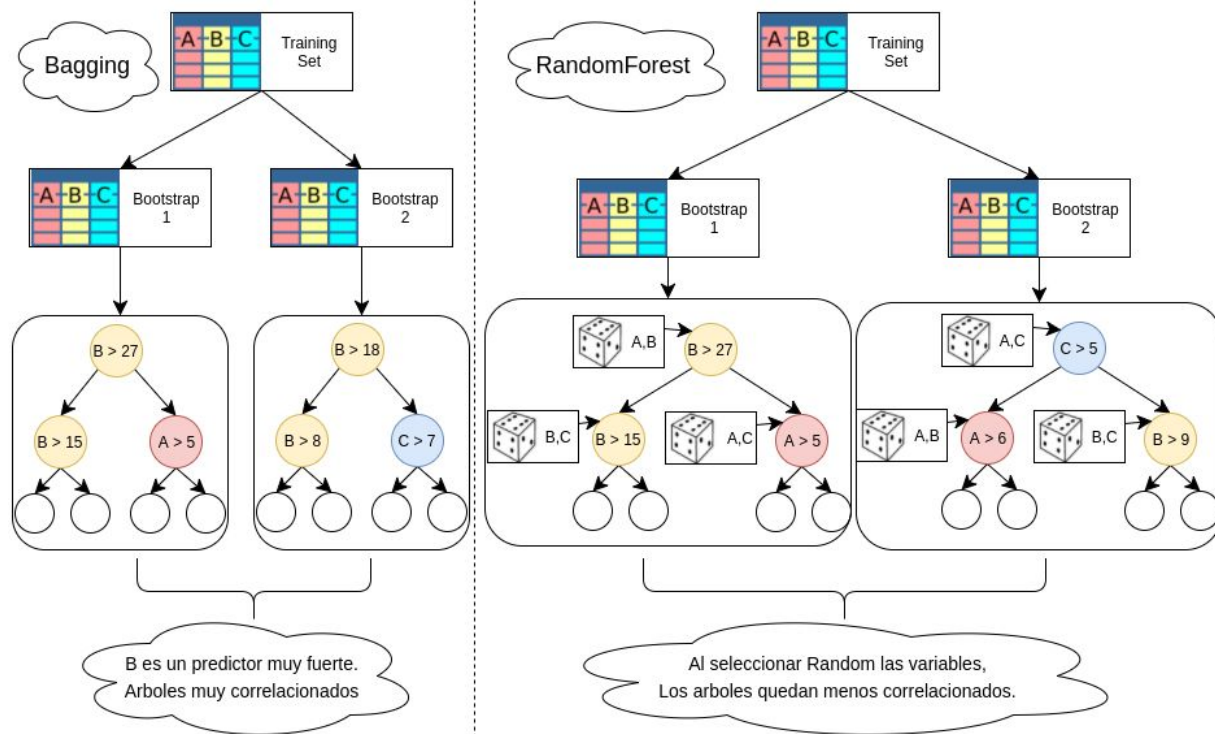
# Random Forest

- 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  $\sqrt{p}$  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  $\{a_1, \dots, a_K\}$  among all non constant (in  $S$ ) candidate attributes;~~
- Draw  ~~$K$  splits  $\{s_1, \dots, s_K\}$ , where  $s_i = \text{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \dots, K$ ;~~
- Return a split  $s_*$  such that  ~~$\text{Score}(s_*, S) = \max_{i=1, \dots, K} \text{Score}(s_i, S)$ .~~

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  ~~$a_{\max}^S$  and  $a_{\min}^S$  denote the maximal and minimal value of  $a$  in  $S$ ;~~
- ~~Draw a random cut-point  $a_c$  uniformly in  $[a_{\min}^S, a_{\max}^S]$ ;~~
- 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  $\{a_1, \dots, a_K\}$  among all non constant (in  $S$ ) candidate attributes;
- Draw  $K$  splits  $\{s_1, \dots, s_K\}$ , where  $s_i = \text{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \dots, K$ ;
- Return a split  $s_*$  such that  $\text{Score}(s_*, S) = \max_{i=1, \dots, K} \text{Score}(s_i, S)$ .

### **Pick\_a\_random\_split( $S, a$ )**

*Inputs:* a subset  $S$  and an attribute  $a$

*Output:* a split

- Let  $a_{\max}^S$  and  $a_{\min}^S$  denote the maximal and minimal value of  $a$  in  $S$ ;
- Draw a random cut-point  $a_c$  uniformly in  $[a_{\min}^S, a_{\max}^S]$ ;
- Return the split  $[a < a_c]$ .

Selecciona K variables igual que  
Random Forest

### **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  $\{a_1, \dots, a_K\}$  among all non constant (in  $S$ ) candidate attributes;
- Draw  $K$  splits  $\{s_1, \dots, s_K\}$ , where  $s_i = \text{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \dots, K$ ;
- Return a split  $s_*$  such that  $\text{Score}(s_*, S) = \max_{i=1, \dots, K} \text{Score}(s_i, S)$ .

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  $a_{\max}^S$  and  $a_{\min}^S$  denote the maximal and minimal value of  $a$  in  $S$ ;
- Draw a random cut-point  $a_c$  uniformly in  $[a_{\min}^S, a_{\max}^S]$ ;
- 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.





## Síntesis

- Ensamblas: 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