DigitalHouse > Coding School

DATA SCIENCE

MÓDULO 6

Ensamble y Bagging. Random Forest y Extra Trees.

Ensamble y Bagging.
Random Forest y Extra Trees

OBJETIVOS DE LA CLASE

1. Explicar el poder de los clasificadores de ensamble
2. Conocer la diferencia entre un clasificador base y un clasificador de ensamble
3. Describir cómo funciona el bagging
4. Presentar los modelos Random Forest y Extra Trees. Analizar su diferencia con el Bagging de árboles de decisión.
5. Implementar estos modelos en scikit-learn

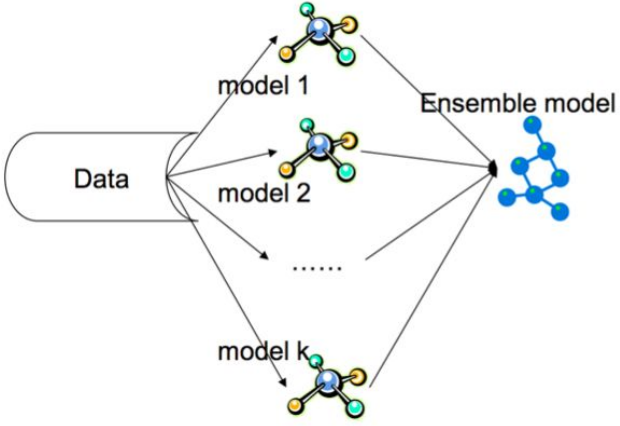
Técnicas de Ensamble

Ensamble

- Los métodos de **Ensamble** son 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.

- Los ensambles suelen ser mucho más precisos que los modelos base que los componen.



Ensamble

Generalmente se distinguen dos familias de métodos de ensamble:

- Los 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**.

- La otra familia de métodos de ensamble son los 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**.

El espacio de Hipótesis

En cualquier tarea de **aprendizaje supervisado**, nuestro objetivo es hacer predicciones de la verdadera función de clasificación *f* aprendiendo el clasificador *h*. En otras palabras, 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**.
Estos son tres de los posibles problemas:

- Estadísticos
- Computacionales
- De representación

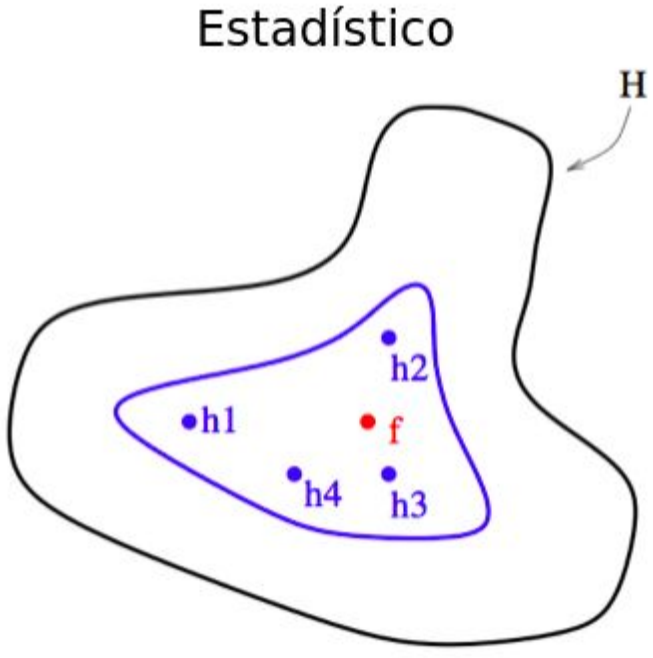
El problema estadístico

Si la cantidad de datos de entrenamiento disponibles es **pequeña**, el clasificador base tendrá dificultades para converger a *f* .

Un clasificador de ensamble puede mitigar este problema "**promediando**" las predicciones de los clasificadores base para mejorar la convergencia.

Esto puede representarse gráficamente como una búsqueda en un espacio en el que múltiples aproximaciones parciales son promediadas para obtener una mejor aproximación al objetivo.

La función real *f* es mejor aproximada como un promedio de los clasificadores base *hi*.

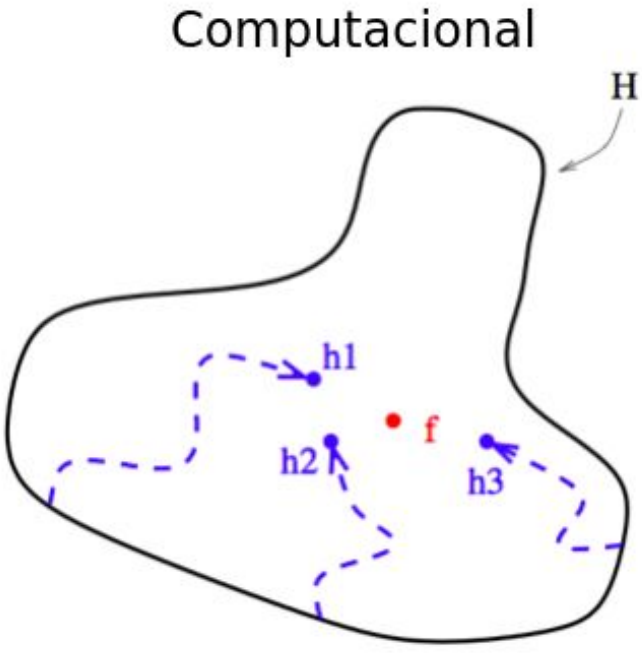


El Problema Computacional

Incluso con suficientes datos de entrenamiento, puede ser computacionalmente difícil encontrar el mejor clasificador *h*.

Por ejemplo, si nuestro clasificador base es un árbol de decisión, una búsqueda exhaustiva del espacio de hipótesis de todos los posibles clasificadores es un problema extremadamente complejo (NP-completo). Por ésta razón usamos un **algoritmo voraz**.

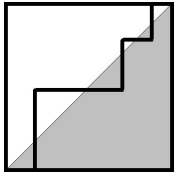
Un conjunto compuesto de varios clasificadores base con **diferentes puntos de partida** puede proporcionar una mejor aproximación de *f* que cualquier clasificador base individualmente. La verdadera función *f* es usualmente mejor aproximada usando varios puntos de partida para explorar el espacio de hipótesis *H*.



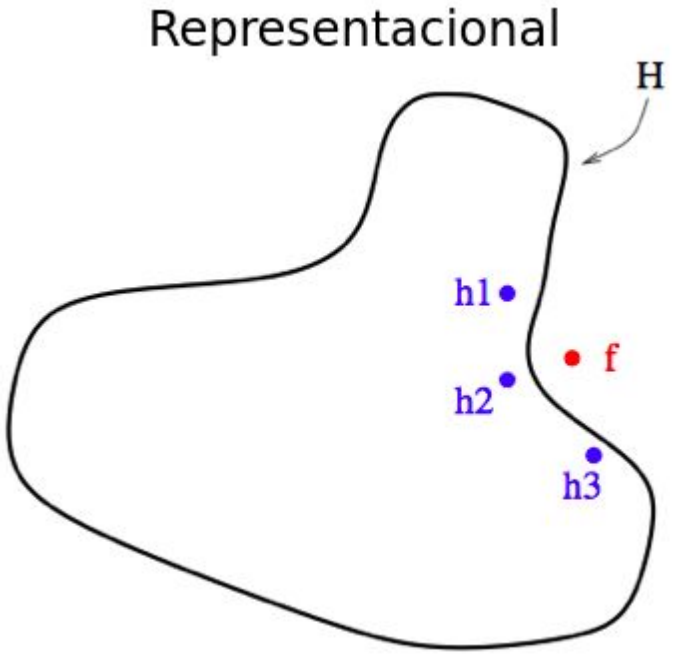
El Problema de representación

A veces *f* no se puede expresar en términos de la hipótesis.

Por ejemplo, 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.



Sin embargo, todavía puede ser posible aproximar *f*, e incluso expandir el espacio de funciones representables, usando métodos de ensamble.



Condición necesaria y suficiente

La condición necesaria y suficiente que se debe cumplir para que un clasificador de ensamble mejore los resultados de cualquiera de sus clasificadores base es que estos sean precisos y diversos.

- **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)

Imaginemos que tenemos un ensamble de tres clasificadores base { h1, h2, h3 } y consideramos un caso nuevo x. Si los tres clasificadores son similares, cuando h1(x) sea erróneo, probablemente h2(x) y h3(x) también lo serían y no ganaríamos nada al combinarlos. Pero si son bastante diversos, los errores que cometan estarán poco correlacionados, y cuando h1(x) sea erróneo, posiblemente h2(x) y h3(x) sean correctos y el voto mayoritario sería correcto.
Esto es cierto, siempre y cuando, el error rate individual de cada uno sea menor al 50%

BAGGING

Introducción a Bagging

El **Bagging** o **B**ootstrap **agg**regat**ing** es un método que consiste en manipular el set de entrenamiento haciendo **remuestreo** para generar k clasificadores base.

Bootstrap es el procedimiento de generar, a partir del set de entrenamiento, ***k* muestras diferentes**.

Las muestras se crean de forma independiente haciendo un muestreo con **reposición** sobre los datos de entrenamiento, usando una distribución de muestreo uniforme.

Luego con estas k muestras, son entrenados *k* clasificadores.

Por último, estos **k modelos distintos son agregados y el resultado será un ensamble que tomará una decisión por voto mayoritario**. En otras palabras, cada modelo en el ensamble vota con igual peso.

El bagging entrena a cada modelo en el ensamble usando un subset aleatorio, del set de entrenamiento, con el fin de promover la varianza de los modelos base. Como ejemplo, el algoritmo random forest combina árboles de decisión aleatorios con bagging para lograr una precisión de clasificación muy alta.

Introducción a Bagging

Original	1	2	3	4	5	6	7	8
set 1	2	7	8	3	7	6	3	1
set 2	7	8	5	6	4	2	7	1
set 3	3	6	2	7	5	6	2	2
set 4	4	5	1	4	6	4	3	8

Dado un set de entrenamiento estándar *D* de tamaño *n*, el bagging genera *m* nuevos sets de entrenamiento *Di* , cada uno de tamaño *n*, muestreando uniformemente en *D* con reemplazo.

Mediante el muestreo con reemplazo, algunas observaciones pueden repetirse en cada *Di*. Los *m* modelos se ajustan usando las *m* muestras anteriores y se combinan promediando la salida (para regresión) o votando (para clasificación).

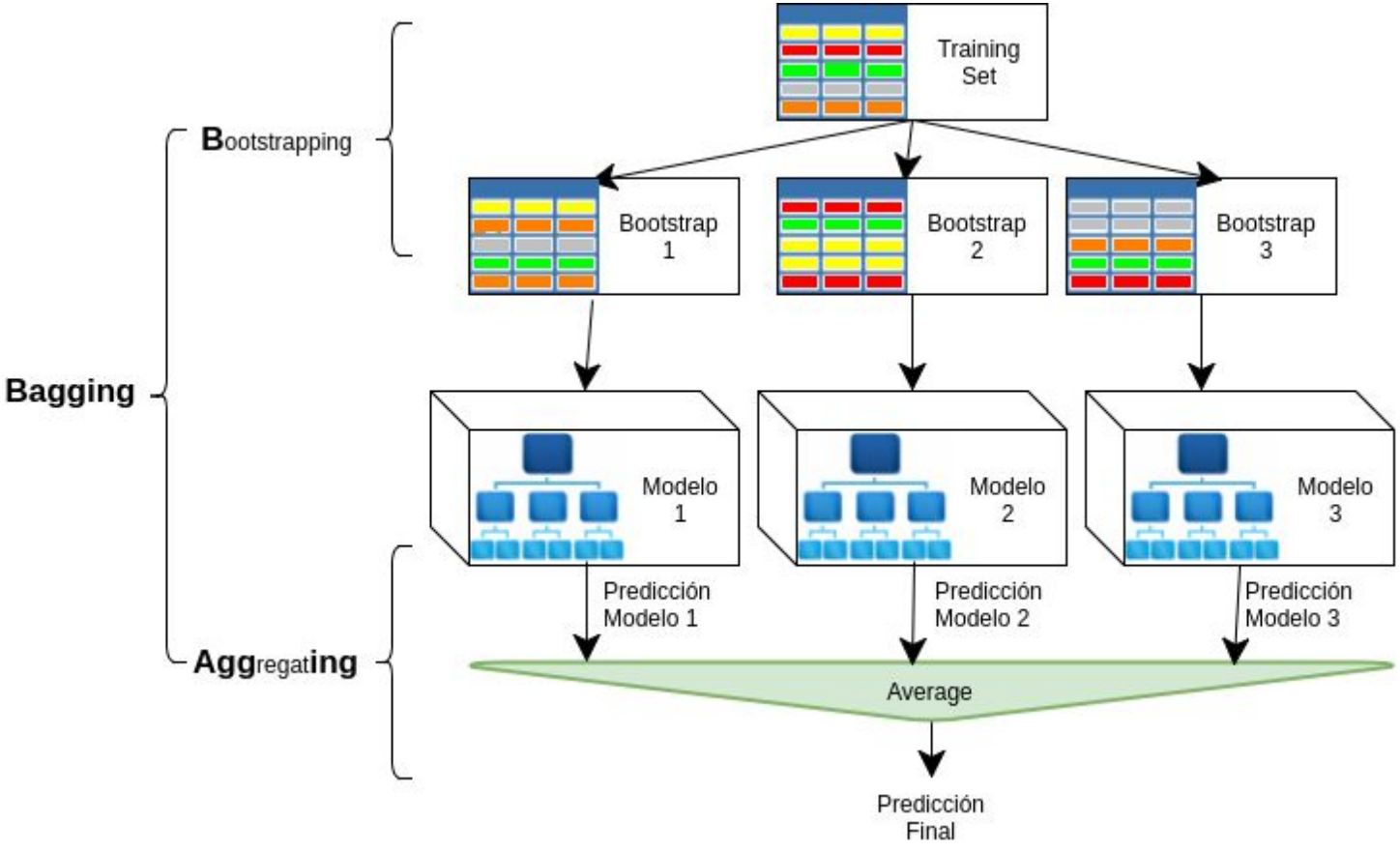
Características

- 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.

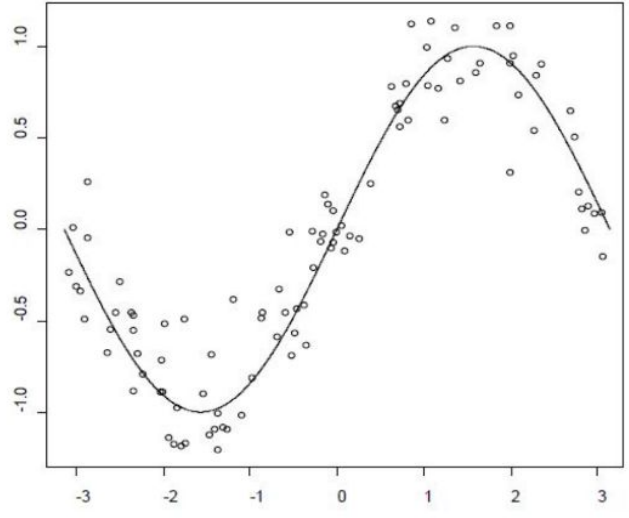
- Dado que proporcionan una manera de reducir el overfitting, los métodos de bagging **funcionan mejor con modelos fuertes y complejos** (por ejemplo, árboles de decisión completamente desarrollados), en contraste a los métodos de boosting que usualmente funcionan mejor con modelos débiles (por ejemplo, árboles de decisión superficiales).


Bootstrapping
Bagging
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

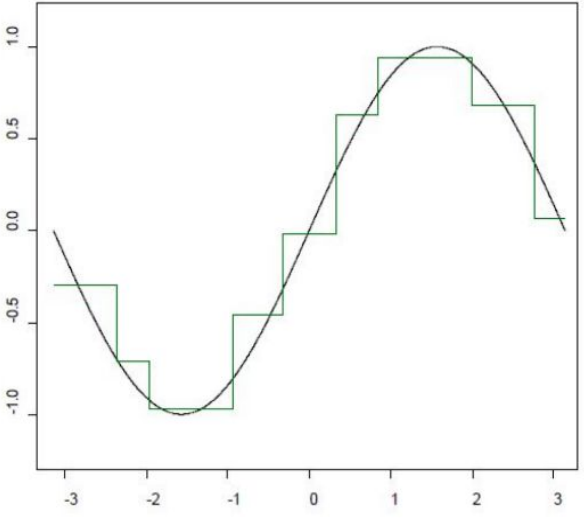
Bagging Predictors (Abstract del Paper Leo Breiman 1994)

— "Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The **aggregation averages** over the versions when predicting a numerical outcome and does a **plurality vote** when predicting a class. The multiple versions are formed by making **bootstrap replicates of the learning set** and using these as new learning sets. Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy. The **vital element is the instability of the prediction method**. If **perturbing the learning set** can cause significant changes in the predictor constructed, then bagging can **improve accuracy**."

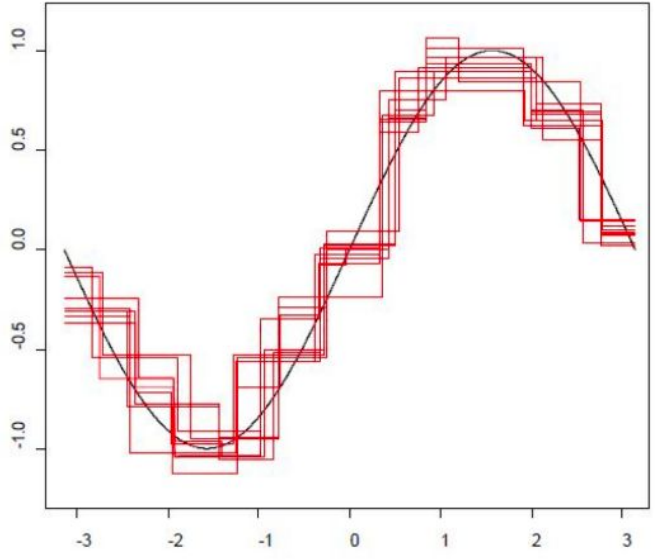
Función generadora



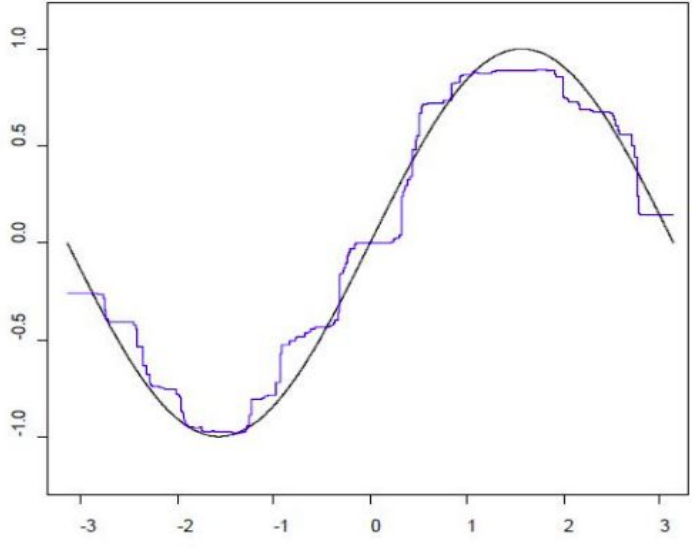
Un árbol de decisión



10 árboles de decisión



Promedio de 10 árboles de decisión



Práctica Guiada Ensambles y Bagging

El meta-estimador Bagging en Scikit Learn

En scikit-learn, los métodos de bagging se ofrecen como un **meta-estimador** unificado de **BaggingClassifier** y **BaggingRegressor**, tomando como entrada un estimador de base definido por el usuario junto con parámetros que especifican la estrategia para construir subsets aleatorios.

En particular, ***max_samples*** y ***max_features*** controlan el tamaño de los subsets (en términos de muestras y características), mientras que ***bootstrap*** y ***bootstrap_features*** controlan si las muestras y características se toman con o sin reemplazo. A su vez, ***n_estimators*** controla la cantidad de clasificadores base a estimar (y, por ende, la cantidad de remuestras a extraer).

Cuando se utiliza un subset de los ejemplos disponibles, el error de generalización se puede estimar con los ejemplos fuera de bolsa **(out-of-bag) poniendo *oob_score=True*.**

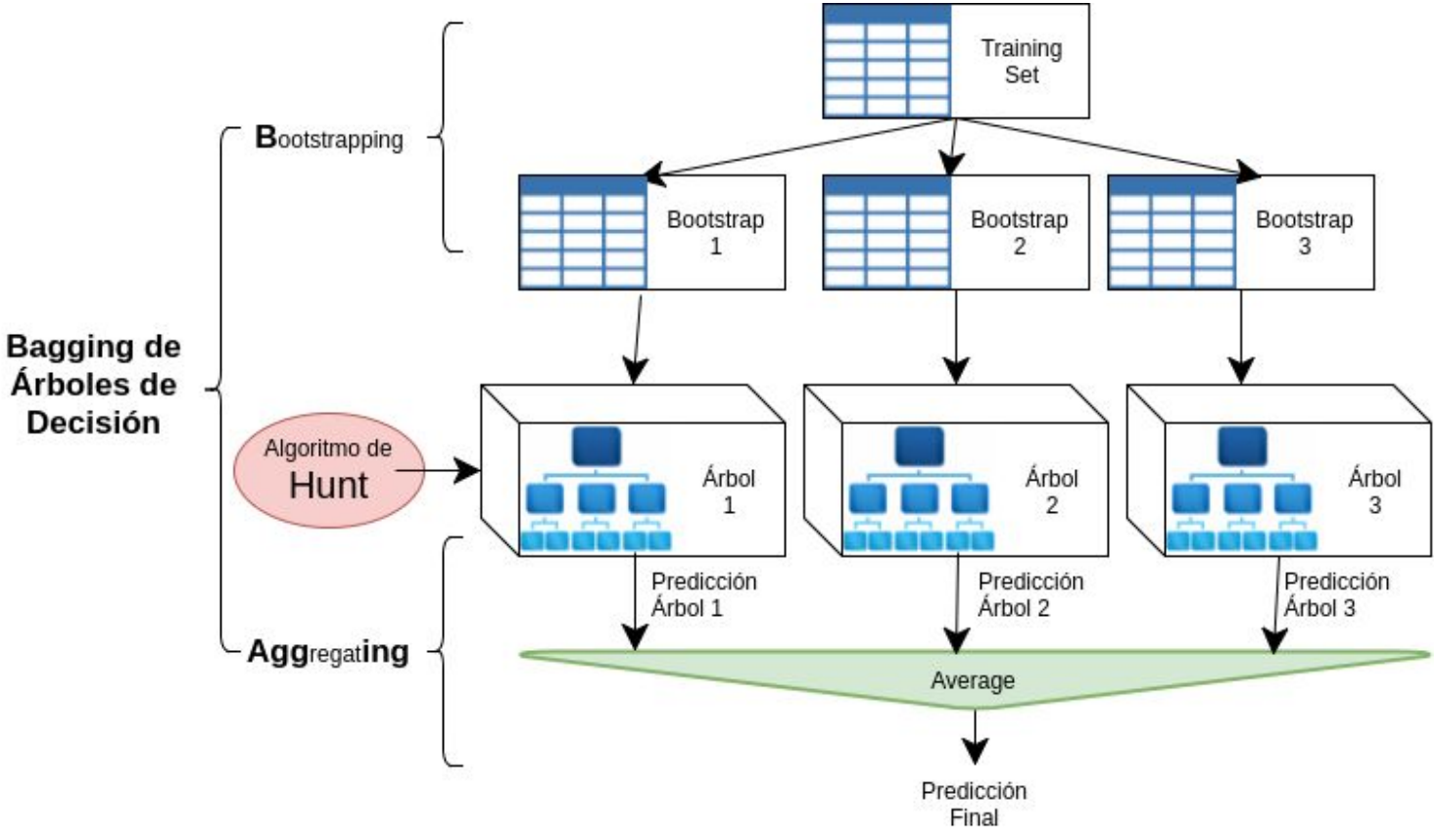
Random Forest y Extra Trees

Introducción

- Como ya vimos, los árboles de decisión son modelos muy poderosos de machine learning. Son muy fáciles de usar porque requieren el seteo de muy pocos parámetros y se desempeñan bastante bien.

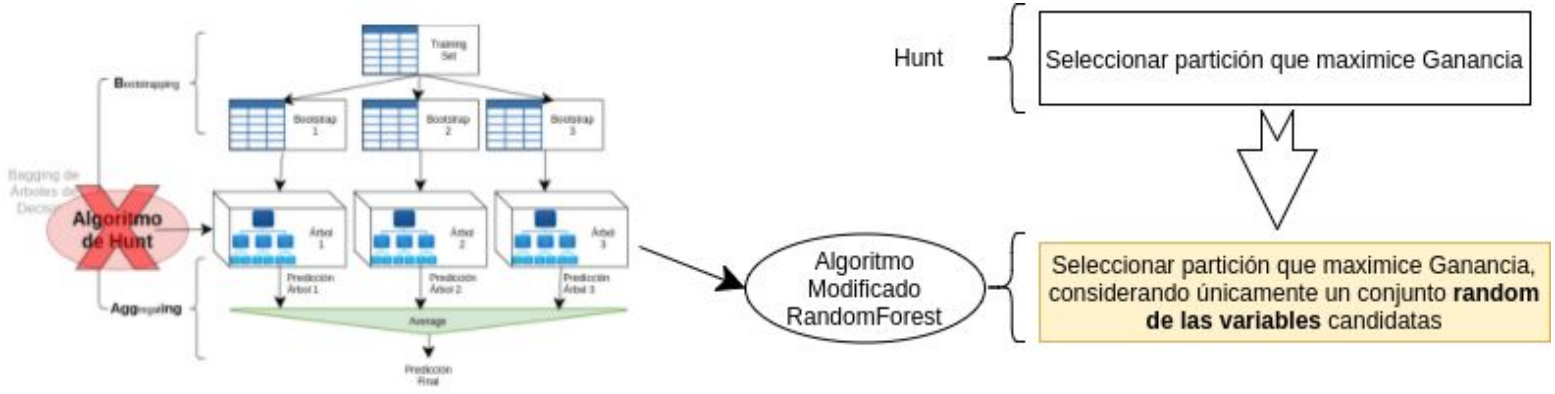
- Pero los **Árboles de Decisión** tienen algunas limitaciones, en particular, los árboles que crecen muy profundamente tienden a aprender patrones altamente irregulares, es decir que **sobre-ajustan**.

- El **Bagging** ayuda a mitigar este problema al exponer diferentes árboles a diferentes sub-sets del set de entrenamiento.


Training Set
Bootstrapping
Bootstrap 1
Bootstrap 2
Bootstrap 3
Bagging de Árboles de Decisión
Algoritmo de Hunt
Árbol 1
Árbol 2
Árbol 3
Predicción Árbol 1
Predicción Árbol 2
Predicción Árbol 3
Aggregating
Average
Predicción Final

- Los ***Random forests*** se diferencian del bagging de árboles de decisión en una sola cosa: usan un algoritmo de aprendizaje de árbol modificado que selecciona, en cada división candidata, un **subconjunto aleatorio de variables**. Este proceso se denomina a veces **bagging de variables (feature bagging)**.

- Los Random forests son una forma adicional de promediar múltiples árboles de decisión profundos, entrenados en diferentes partes del mismo set de entrenamiento, con el objetivo de reducir la varianza. Esto se produce a expensas de un pequeño aumento en el sesgo y cierta pérdida de interpretabilidad, pero en general **aumenta considerablemente el rendimiento** del modelo final.

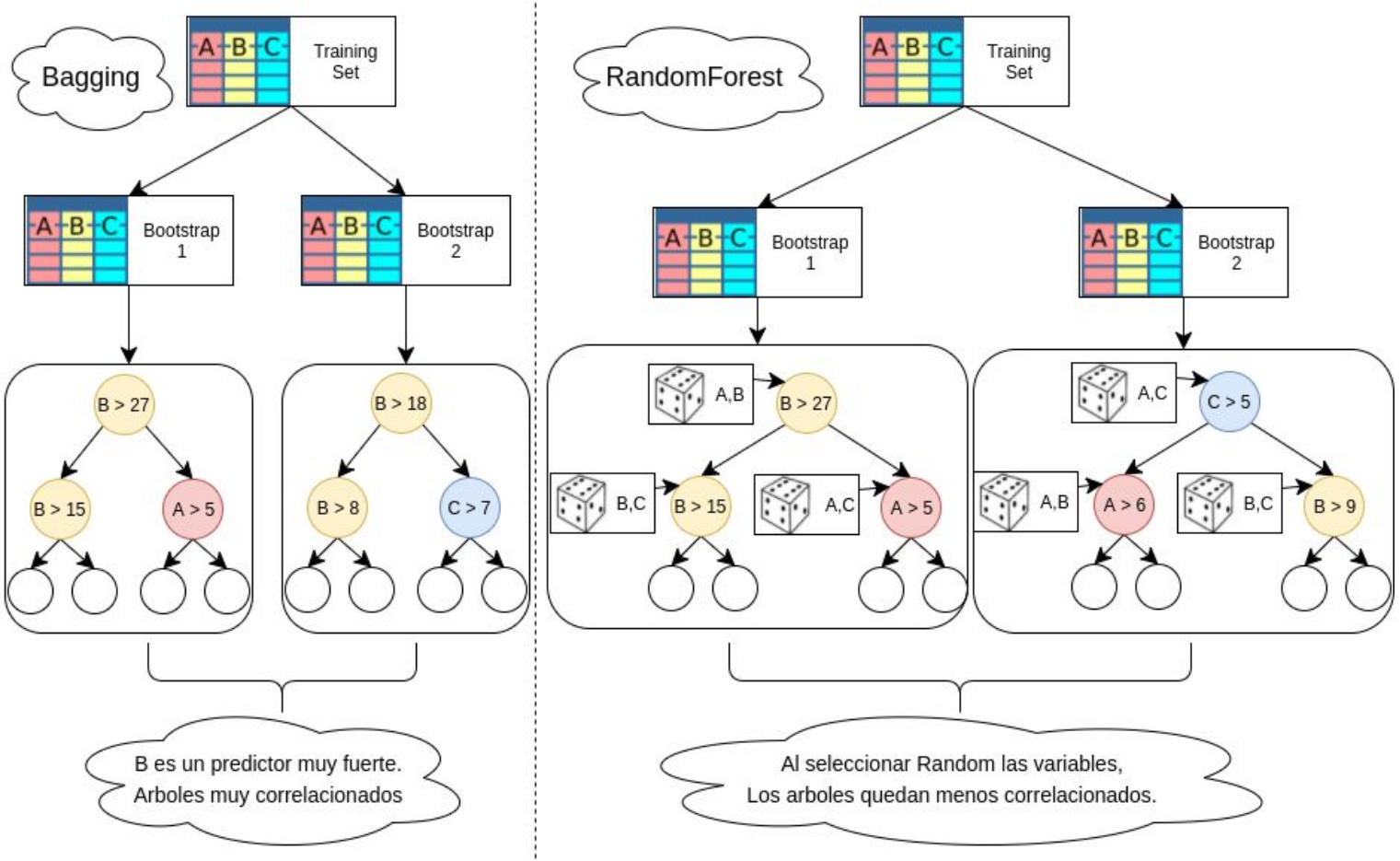


- La razón para hacer esto es la correlación de los árboles en una muestra de bootstrap normal: si una o algunas variables son **predictores muy fuertes** para la variable target, estas variables **serán seleccionadas en muchos de los árboles** base del bagging, haciendo que queden correlacionados. Seleccionando un subconjunto aleatorio de las variables en cada división, contrarrestamos esta correlación entre los árboles base, fortaleciendo el modelo final.

- 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**.


Bagging
A B C
Training Set
A B C
Bootstrap 1
A B C
Bootstrap 2
B > 27
B > 15
A > 5
B > 18
B > 8
C > 7
B es un predictor muy fuerte.
Arboles muy correlacionados
RandomForest
A B C
Training Set
A B C
Bootstrap 1
A B C
Bootstrap 2
A,B
B > 27
B,C
B > 15
A,C
A > 5
A,C
C > 5
A,B
A > 6
B,C
B > 9
Al seleccionar Random las variables,
Los arboles quedan menos correlacionados.

Algoritmo 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 $\{T_b\}_1^B$.

Extremely randomized trees

- La adición de un paso más de aleatorización produce árboles muy aleatorizados o **ExtraTrees**.

- Éstos se entrenan usando bagging y el método de selección aleatoria de variables, como en un *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 random**.

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 $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– 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 n
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 $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

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 $a^S_{\max}$ 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 Score(s_*, S) = $\max_{i=1,\ldots,K}$ 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.

CONCLUSIÓN

- Vimos los modelos Ensamble y Bagging. Vimos **Random Forest** y **ExtraTrees.**

- Vimos cómo mejoran el rendimiento de los modelos base individuales gracias a su mayor capacidad para aproximar la función de predicción real en un problema de aprendizaje supervisado.

  - Algunos de estos métodos funcionan mejor en unos casos, algunos mejor en otros.
    Por ejemplo, los Árboles de Decisión son más ágiles y fáciles de comunicar, pero tienen tendencia a sobre-ajustarse.

- Por otro lado, los métodos de Ensamble se desenvuelven mejor en escenarios más complejos, pero pueden llegar a ser muy complicados y difíciles de explicar.