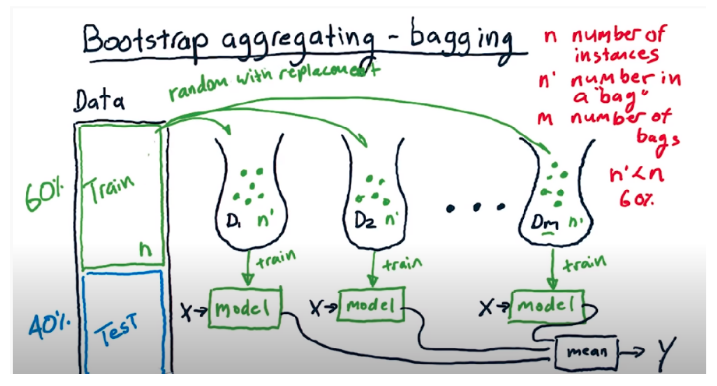
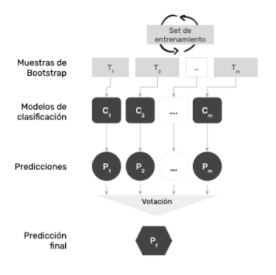
**Bagging. Random Forest y Extra Trees**

**Bagging (Bootstrap Aggregation)**: Muestreo con Reposición de Instancias.

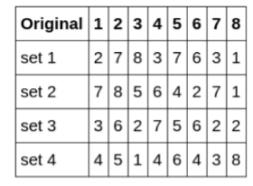


Se extraen varias muestras a partir de una muestra de datos, de manera aleaoria y con reposición (una misma observación puede estar más de una vez en una muestra). Con estas muestras bootstrapped, se entrenan los modelos en forma separada. Lo más probable es que tengamos modelos con mucha varianza. Combinando las predicciones de todos los modelos base tendremos la predicción final.

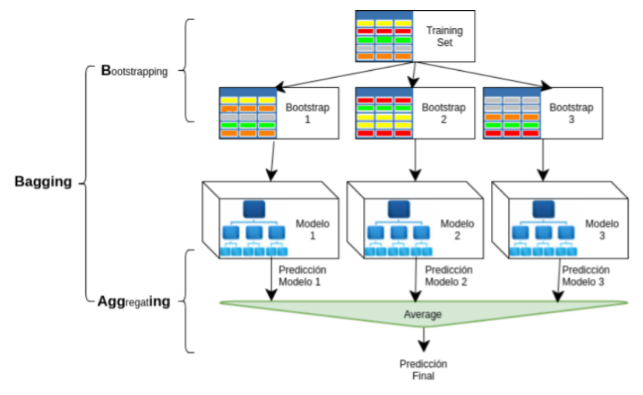


Lo que se hace con **Bagging** es **manipular el set de entrenamiento** haciendo **remuestreo** para generar **k clasificadores base** con **k muestras diferentes**. Estas muestras se generan en forma **independiente**, haciendo un **muestreo con reposición** y usando una **distribución de muestreo uniforme**. Los **k modelos distintos** son **agregados**. Como resultado tendremos un **ensamble** que **decidirá** a partir del **voto mayoritario**. **Cada modelo** **vota** con el **mismo peso**. Se busca que los modelos base tengan una gran varianza y que su overfiteo se compense al agregarlos.

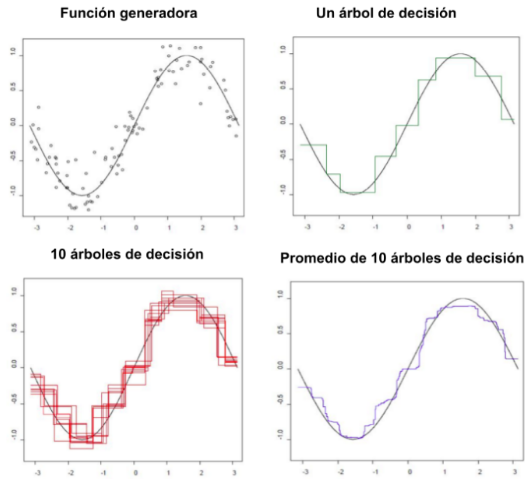
Si tenemos un **set de entrenamiento estándar D** de **tamaño n**; con **bagging** vamos a generar **m nuevos sets de entrenamiento Di**, cada uno **de tamaño n**, pero **muestreando uniformemente en D con reemplazo**. Entonces algunas observaciones de algunos nuevos sets de entrenamiento Di podrían repetirse. Con estas **m muestras** podemos **entrenar m modelos** y combinarlos **promediando la salida** (para **regresión**) o **votando** (para **clasificación**).

****

* Con Bagging, **se reduce la varianza** del error de generalización al **combinar múltiples clasificadores base**.
* Si el **clasificador** es **estable**, entonces el **error de ensamble puede deberse** **principalmente** al **sesgo**. En estos casos, el bagging no funciona. Es **clave** trabajar con **clasificadores inestables**, que den resultados muy distintos ante diferentes sets de entrenamientos para que este método funcione.
* Como **cada muestra** de datos es **igualmente probable**, es **más difícil que el bagging** **overfitee con datos ruidosos**.
* Los métodos de bagging **funcionan mejor** con **modelos fuertes y complejos**¸ tales como árboles de decisión completamente desarrollados. En cambio, los **métodos de boosting** **funcionan mejor** con **modelos débiles,** tales como árboles de decisión superficiales.

****

Ejemplo gráfico de cómo funciona esto con distintos árboles de decisión:



En scikit-learn, los métodos de bagging se ofrecen como un **meta-estimador** unificado de **BaggingClassifier** y **BaggingRegressor**, usando como **entrada** un **estimador de base definido por el usuario** junto **con parámetros** que especifican la **estrategia** para **construir subsets aleatorios**. Con **max\_samples** y **max\_features** es posible controlar el tamaño de los subsets (muestras y features). Con **Bootstrap** y **bootstrap\_features** es posible controlar si las **muestras** y las **features** se toman **con o sin reemplazo**. Con **n\_estimators** es posible controlar la cantidad de clasificadores base a entrenar. Al usar un subset de las instancias de entrenamiento disponibles, se puede estimar el error de generalización con aquellas instancias “fuera de bolsa” (**out-of-bag**)**.** Para esto hay que establecer **oob\_score = True**.

**En Python:** Vamos a trabajar sobre el Dataset Hitters.csv para entrenar un modelo de regresión usando BaggingRegressor para predecir el valor de log(Salary). Como modelo vamos a usar una regresión lineal múltiple con dos variables predictoras. Vamos a calcular la predicción de cada instancia como el promedio de las predicciones de cada uno de los 3 modelos:

import pandas as pd

import numpy as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import BaggingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

data\_raw = pd.read\_csv(‘../Data/Hitters.csv’)

print(data\_raw.shape)

data\_complete = data\_raw.dropna()

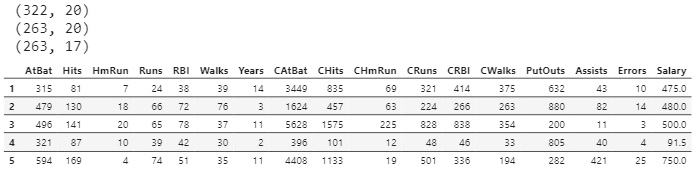
print(data\_complete.shape)

data\_columns = [‘AtBat’, ‘Hits’, ‘HmRun’, ‘Runs’, ‘RBI’, ‘Walks’, ‘Years’, ‘CAtBat’, ‘CHits’, ‘CHmRun’, ‘CRuns’, ‘CRBI’, ‘CWalks’, ‘PutOuts’, ‘Assists’, ‘Errors’, ‘Salary’]

data = data\_complete.loc[:, data\_columns]

print(data.shape)

data.head()



X = data\_complete.drop(‘Salary’, axis = 1)

print(X.shape)

y = np.log(data.Salary)

print(y.shape)

X\_train, X\_test, y\_train\_y\_test = train\_test\_split(X, y, random\_state = 127)



scaler = StandardScaler()

X\_train\_scl = scaler.fit\_transform(X\_train)

X\_test\_scl = scaler.transform(X\_test)

# **Parámetros** para **modelos de bagging**; **n\_estimators** (cantidad de modelos a entrenar); **max\_samples** (cantidad de muestras del conjunto de entrenamiento que se usan para entrenar cada modelo base); **bootstrap** (si el muestreo es o no con repetición); **max\_features** (cantidad de featurse por predictor base); **bootstrap\_features** (si las features se seleccionan con reposición), **n\_jobs** (cantidad de jobs que se ejecuta en paralelo; con -1 se usan todos los procesadores disponibles).

base\_regressor = LinearRegression()

fit\_base = base\_regressor.fit(X\_train\_scl, y\_train)

predict\_base = base\_regressor.predict(X\_test\_scl)

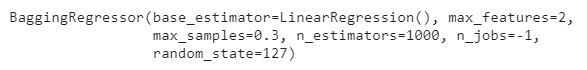
performance\_base = mean\_squared\_error(y\_test, predict\_base)

performance\_base



bag\_linreg = BaggingRegressor(base\_estimator = base\_regressor, n\_estimators = 1000, max\_samples = 0.3, bootstrap = True, max\_features = 2, bootstrap\_features = False, n\_jobs = -1, random\_state = 127)

bag\_linreg.fit(X\_train\_scl, y\_train)



prediction = bag\_linreg.predict(X\_test\_scl)

performance = mean\_squared\_error(y\_test, prediction)

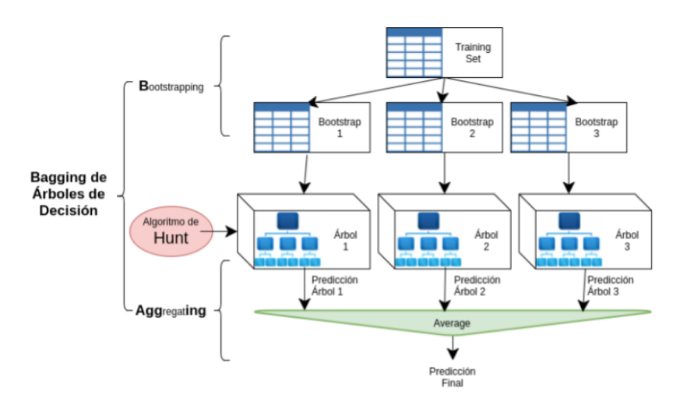
performance



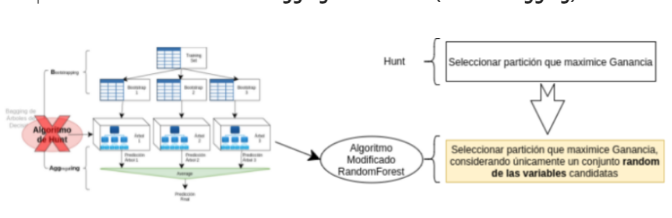
En este caso, obtuvimos una mejor performance con bagging que con el modelo base.

**Random Forest**

Los **árboles de decisión** son modelos muy poderosos de machine learning. Requieren de **pocos parámetros** y funcionan bastante bien, pero **tienden a overfitear** si crecen muy profundamente. Esto se puede **contrarrestar usando bagging**.



**Random Forest** es un **algoritmo de aprendizaje de árbol modificado** que **selecciona en cada división** un **subconjunto aleatorio de variables.** Esta es la única diferencia que tiene con bagging. A este proceso a veces se lo llama **feature bagging**:

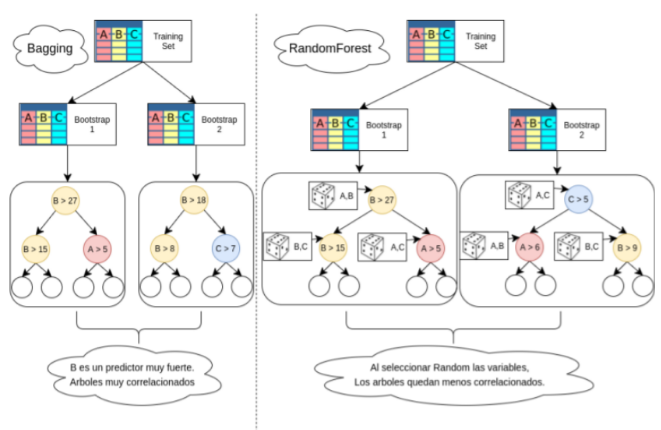


Es otra manera de **promediar** **múltiples árboles de decisión profundos**, entrenados con **diferentes partes del set de entrenamiento**, para reducir la varianza. La **contrapartida** es que **aumenta un poco el sesgo** y se **pierde** un poco la **interpretatividad**; pero generalmente **aumenta considerablemente** el **rendimiento del modelo final**. El **fundamento** detrás de esta selección de sólo algunas variables es que **si hay alguna variable muy correlacionada con la variable objetivo**, que aparece en muchos árboles del modelo de ensamble, **esto hará** que dichos **árboles estén correlacionados entre sí** y el **ensamble no funcione** porque **los clasificadores no tienen** **diversidad.**

Si tenemos un **problema de clasificación** de p variables, suelen usarse variables en cada división.

Si tenemos un **problema de regresión** con p variables, suelen usarse p/3.

Otro posible approach es considerar la cantidad de variables a seleccionar por subset de entrenamiento como un hiperparámetro a optimizar.



* Se seleccionan k de m features totales. Se crea un árbol de decisión a partir de esas k features.
* Se crean n árboles en total, variando la cantidad de k features.
* Se guarda el resultado de cada árbol. Entonces tendremos n salidas.
* Se calculan los votos para cada clase seleccionada. La más votada será la clasificación final de nuestro modelo de random forest.

Se puede validar el modelo de Random Forest con **Out of Bag (OOB)**. Se predice con los árboles de entrenamiento aquellas observaciones que nunca vieron porque quedaron *out of bag****,*** es decir, fuera del subconjunto de entrenamiento que se uso con los mismos. Se calcula la predicción como la etiqueta más frecuentemente predicha por estos árboles. OOB es el número de filas predichas correctamente sobre la muestra de filas OOB. La **principal diferencia entre esta métrica y el score de validación** es que el **OOB se calcula usando sólo** un **subconjunto de árboles de decisión** que no vieron estas observaciones; el **Score de Validación** se **calcula con todos los árboles de decisión** del ensamble.

La validación en un conjunto completo de ensambles suele ser mejor que en un subconjunto; pero si el conjunto de datos total no es lo suficientemente grande, no podemos reservar una porción para validar después. El OOB es una buena forma de compensar esto. Dicho esto, vale aclarar que **OOB** y **Score de Validación no son lo mismo**. Se calculan de manera distinta. **No deben compararse**.

**En Python:** Seguimos trabajando con el dataset Hitters.csv

from sklearn.ensemble import RandomForestRegressor

random\_forest = RandomForestRegressor(n\_estimators = 1000, criterion = ‘mse’, max\_depth = 4, bootstrap = True, n\_jobs = -1, random\_state = 127, max\_samples = 0.3)

random\_forest.fit(X\_train\_scl, y\_train)

**

prediction = random\_forest.predict(X\_test\_scl)

performance = mean\_squared\_error(y\_test, prediction)

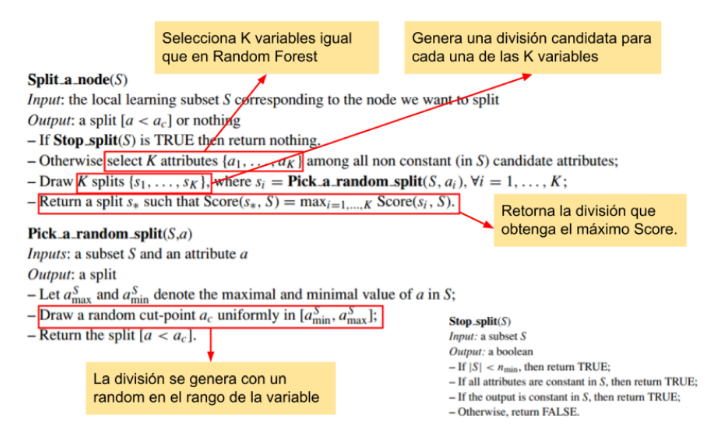
performance



Obtuvimos una performance major que la que habíamos conseguido usando el modelo de bagging de regresiones lineales.

**Extra Trees** (**Ext**remely **Ra**ndomized **Trees**):

* Se adiciona un paso más de aleatorización para producir árboles muy aleatorizados (ExtraTrees).
* Se entrenan estos árboles usando bagging y el método de selección aleatoria de variables. Como un Random Forest, pero con una capa Random adicional.
* Para cada variable en consideración se genera una división aleatoria, dentro del rango de la variable. Una vez hecho esto, recién ahora se busca la variable/división que maximice la ganancia.
* La **principal diferencia** es: la división para cada variable no será la óptima; si no una seleccionada random.



**En Python:** volvemos a tomar el caso del dataset Hitters.csv

from sklearn.ensemble import ExtraTreesRegressor

extra\_tree = ExtraTreeRegressor(n\_estimators = 1000, criterion = ‘mse’, max\_depth = 5, bootstrap = True, n\_jobs = -1, nrandom\_state = 171, max\_samples = 0.3)

extra\_tree.fit(X\_train\_scl, y\_train)



prediction = extra\_tree.predict(X\_test\_scl)

performance = mean\_squared\_error(y\_test, prediction)

performance



Obtuvimos una performance mejor que la del bagging de regresiones lineales y muy similar a la del random forest. Pero contamos con la **ventaja** de un **menor costo computacional que con Random Forest**.

import timeit

repeat\_count = 10

time\_rf = timeit.timeit(stmt=’random\_forest.fit(X\_train\_scl, y\_train)’, global=globals(), number = repeat\_count)

print(time\_rf/repeat\_count)

time\_et = timeit.timeit(stmt=’extra\_tree.fit(X\_train\_scl, y\_train)’, global=globals(), number = repeat\_count)

print(time\_et/repeat\_count)



**Conclusiones:**

* Los **ensambles** son **modelos de predicción** que tienen **menos varianza que** sus **modelos base**.
* Son **robustos frente a outliers y ruido**.
* **Demandantes computacionalmente**, pero **fácilmente paralelizables**.
* **Extra Trees** tiene una **performance similar** a **Random Forest,** pero a un **costo computacional menor.**