Ensamble de predictores

Agenda

- ¿Cómo y por qué combinar predictores?
- Estrategias de combinación de predictores
- Bagging
- Random Forests
- Boosting. Adaboost y Gradient Boosting

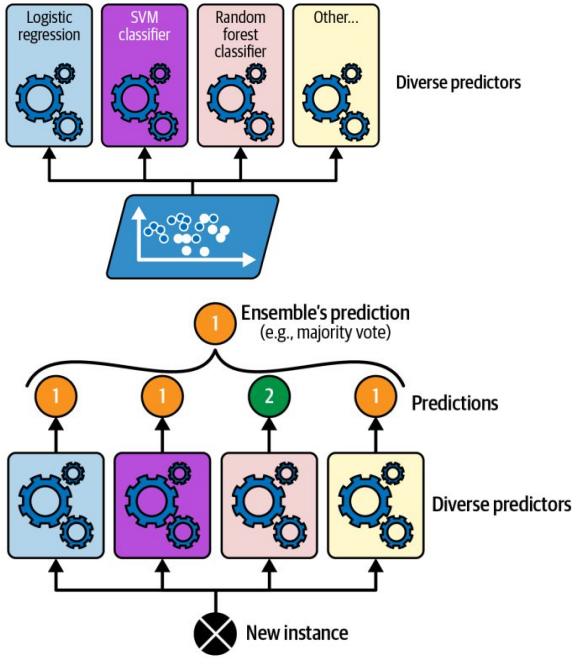
¿Cómo y por qué combinar predictores?

Aprendizaje de ensambles: motivación

- Toma de decisiones difíciles: es común que tengamos en cuenta la opinión de varios expertos para mejorar la decisión.
- Cada modelo se puede ver como un experto y es esperable que si los combinamos obtendremos predicciones más confiables.
- Métodos de ensambles: permiten entrenar y combinar conjuntos de predictores.
- No free lunch theorem: no existe un predictor universalmente mejor que otro para cualquier conjunto de datos.

- Como a priori no hay un mejor predictor para un problema dado ⇒ se prueban varios.
- Es más robusto combinar los predictores individuales siempre y cuando:
  - Tengan un buen desempeño: basta que acierten más del 50%.
  - Sean diversos (independientes): cometan diferentes errores.

Cómo funciona



- La combinación de predictores consta de dos tareas:
  - Entrenar un conjunto de predictores base a partir de los datos.
  - Aplicar una estrategia para combinarlos en un único predictor.

- Ejemplo 1:
  - Entrenar varios predictores.
  - Nueva muestra: cada modelo produce su predicción.
  - Se clasifica según el voto de la mayoría.

- Ejemplo 2:
  - Supongamos que cada clasificador puede cuantificar la confianza en su predicción. (e.g: si en 10-NN, 9 vecinos de x son de la clase Ck , confianza alta).
  - Confianza del clasificador y_m en clasificar la muestra x es $C(y_m|x)$
  - Se clasifica según mayoría ponderada por $C(y_m|x)$

Características deseables del conjunto de predictores base

- Diversidad:
  - predictores especializados en distintas características
  - predictores entrenados sobre distintos datos: subconjuntos al azar
  - distintos tipos de predictor: lineal, k-NN, redes, SVM, ...
  - distintos parámetros: distintos k en k-NN, ...
- Independencia:
  - máxima reducción de error
  - máxima reducción de varianza
- Complementariedad:
  - variedad de especialización

Estrategias de ensamble de predictores

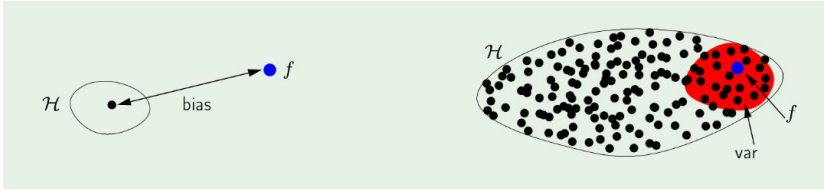
Estrategias de ensamble de predictores

- Entrenamiento en paralelo, sobre un mismo conjunto de entrenamiento
  - Mezcla de expertos:
    distintos tipos de predictor (SVM, k-NN, etc.) entrenados sobre los mismos datos

- Entrenamiento en paralelo, con distintos conjuntos de entrenamiento
  - Bagging (Boostrap Aggregating):
    distintos modelos del mismo tipo entrenados sobre distintos muestreos de los datos
  - Random forest:
    combina bagging sobre árboles de decisión con selección aleatoria de atributos

- Entrenamiento secuencial o en cascada
  - Boosting: al pasar por la cascada de predictores, a las muestras difíciles de clasificar se les va aumentando el peso en el costo total, para que los predictores cuesta arriba se focalicen en su predicción. La decisión es un voto ponderado por el desempeño de cada predictor.

Compromiso sesgo-varianza

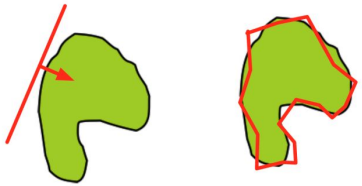


Error cuadrático medio = (Sesgo)^2 + Varianza + Ruido

- Ruido: independiente del predictor; error irreducible, intrínseco al problema.
- Varianza: sensibilidad del predictor ante fluctuaciones en el conjunto de entrenamiento.
- Sesgo: error medio de predicción (sobre todos los conjuntos de datos) del predictor.

Reducción de varianza: si los conjuntos de entrenamiento son independientes, la combinación reduce la varianza sin afectar el sesgo (e.g. bagging).

Reducción del sesgo: la combinación o el promedio de modelos simples tiene mayor capacidad que cada modelo por separado (e.g. boosting, mezcla de expertos).



Sesgo: un clasificador lineal vs. mezcla de clasificadores lineales

¿Por qué funciona?

M clasificadores independientes, todos con probabilidad de error p:

$$P(m \text{ errores}) = \binom{M}{m} p^m (1-p)^{M-m}.$$

Votación por mayoría:

$$P(\text{clasificación incorrecta}) = \sum_{m=\lceil M/2 \rceil}^{M} \binom{M}{m} p^m (1-p)^{M-m}.$$

	$p = 0.3$
M	P(clasificación incorrecta)
11	0.078225
21	0.026390
121	0.000002

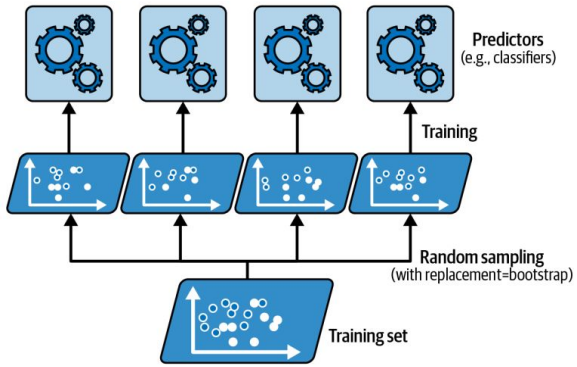
	$p = 0.49$
M	P(clasificación incorrecta)
11	0.472948
121	0.412750
10001	0.022731

Bagging

Predictores Bagging

Bagging: Bootstrap Aggregating*

- M subconjuntos de entrenamiento: muestreo con reposición (bootstrap) de los datos.
- M predictores: cada uno entrenado con uno de los subconjuntos de entrenamiento.
- Las predicciones se combinan por mayoría (eventualmente de forma ponderada).
- Predictor base inestable (sesgo pequeño, varianza grande),
  - e.g. árboles de decisión.



L. Breiman, “Bagging predictors,” Mach. Learn., vol. 24, p. 123–140, Aug. 1996
B. Efron, “Bootstrap methods: Another look at the jackknife,” The Annals of Statistics, vol. 7, pp. 1–26, Jan. 1979

Bagging en Scikit-Learn

Entrenar 500 árboles de decisión sobre 100 instancias tomadas al azar

•con reposición (bootstrap=True): Bagging

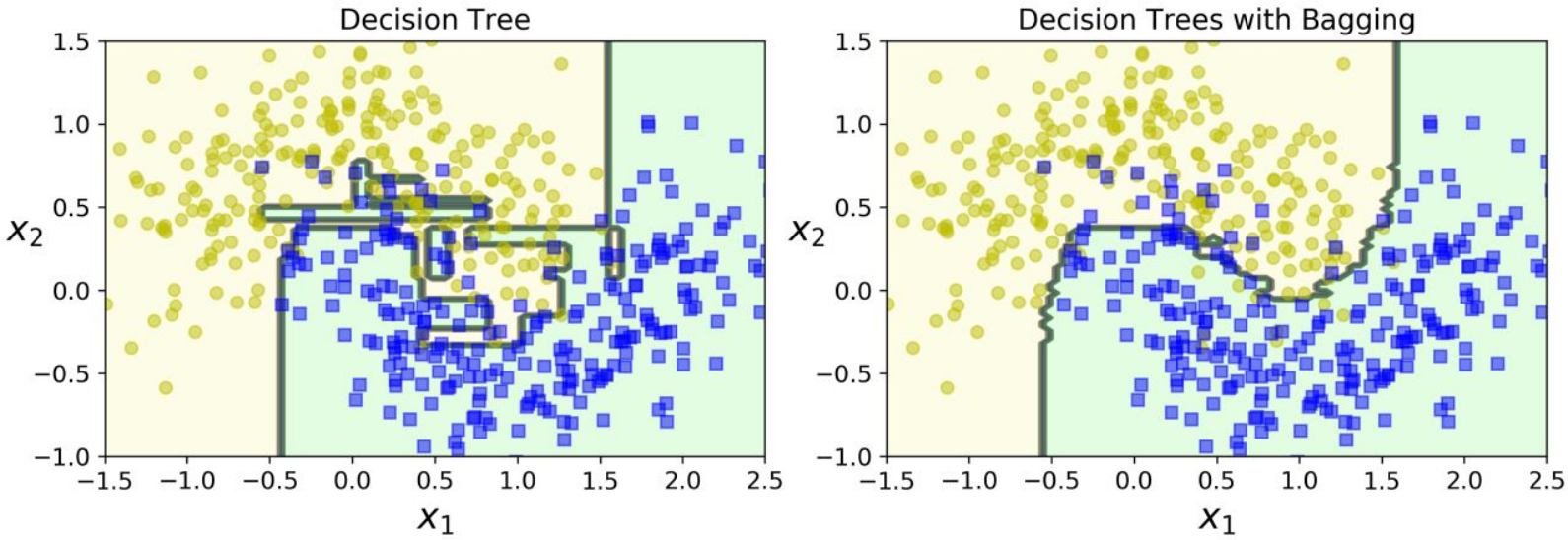
•sin reposición (bootstrap=False): Pasting

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Por defecto, BaggingClassifier hace soft voting (voto ponderado) cuando el clasificador base puede estimar la probabilidad de clase.

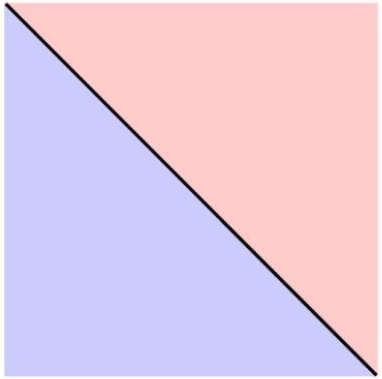
Bagging en Scikit-Learn

Árbol de decisión vs. ensamble bagging con 500 árboles

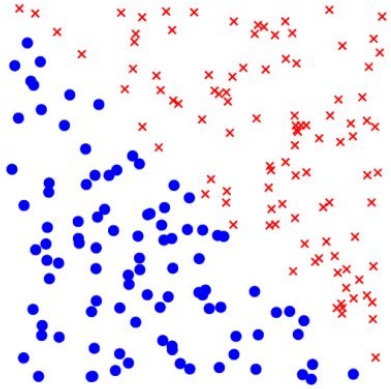


Predictores Bagging

Ejemplo: dos clases, los y_m árboles de decisión



Frontera verdadera

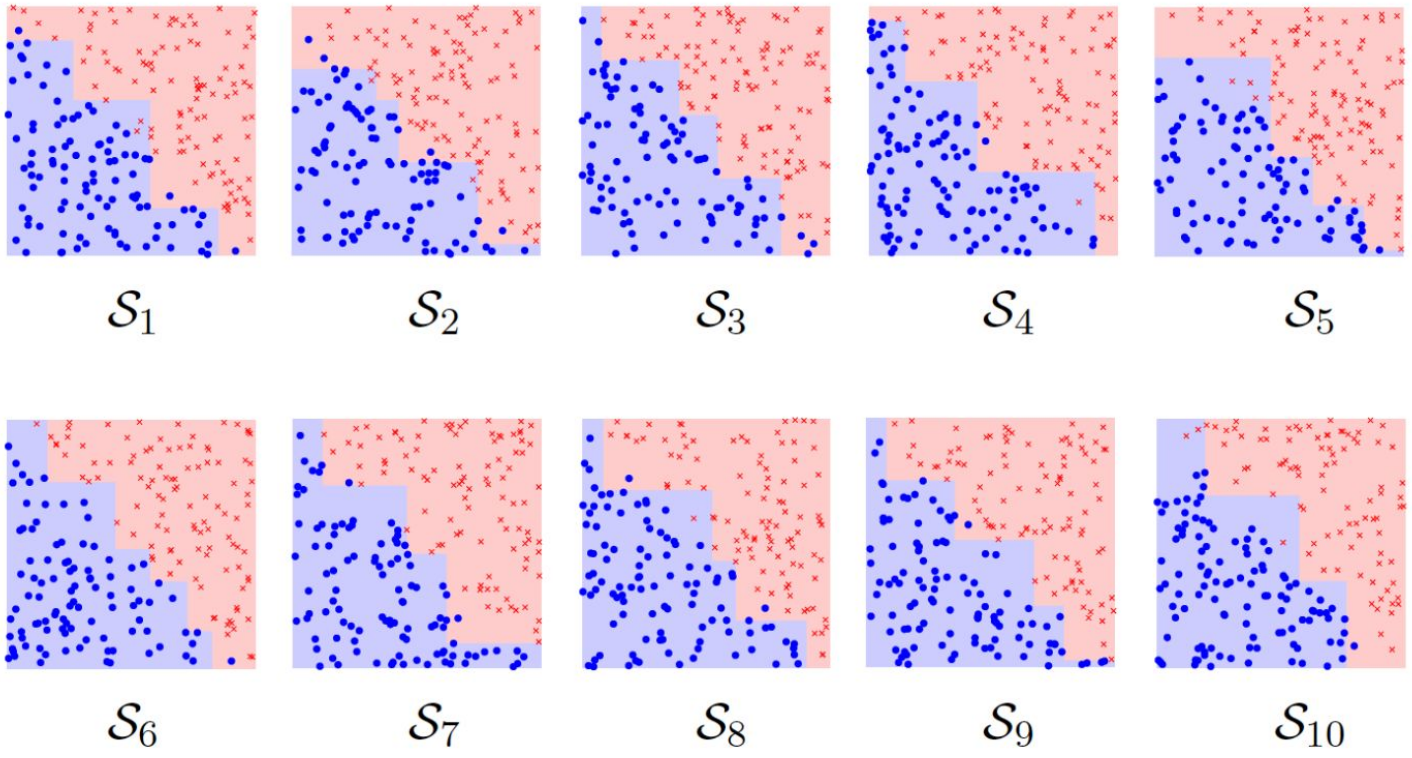


Un muestreo $\mathcal{S}_m$

Árboles de decisión:

- Sesgo bajo: en promedio, frontera de decisión cercana a las verdadera.
- Varianza grande: frontera de decisión muy sensible a la muestra específica.

Predictores Bagging



Predictores Bagging

Random Forest

Random Forest

Sobre los predictores base, vimos que:

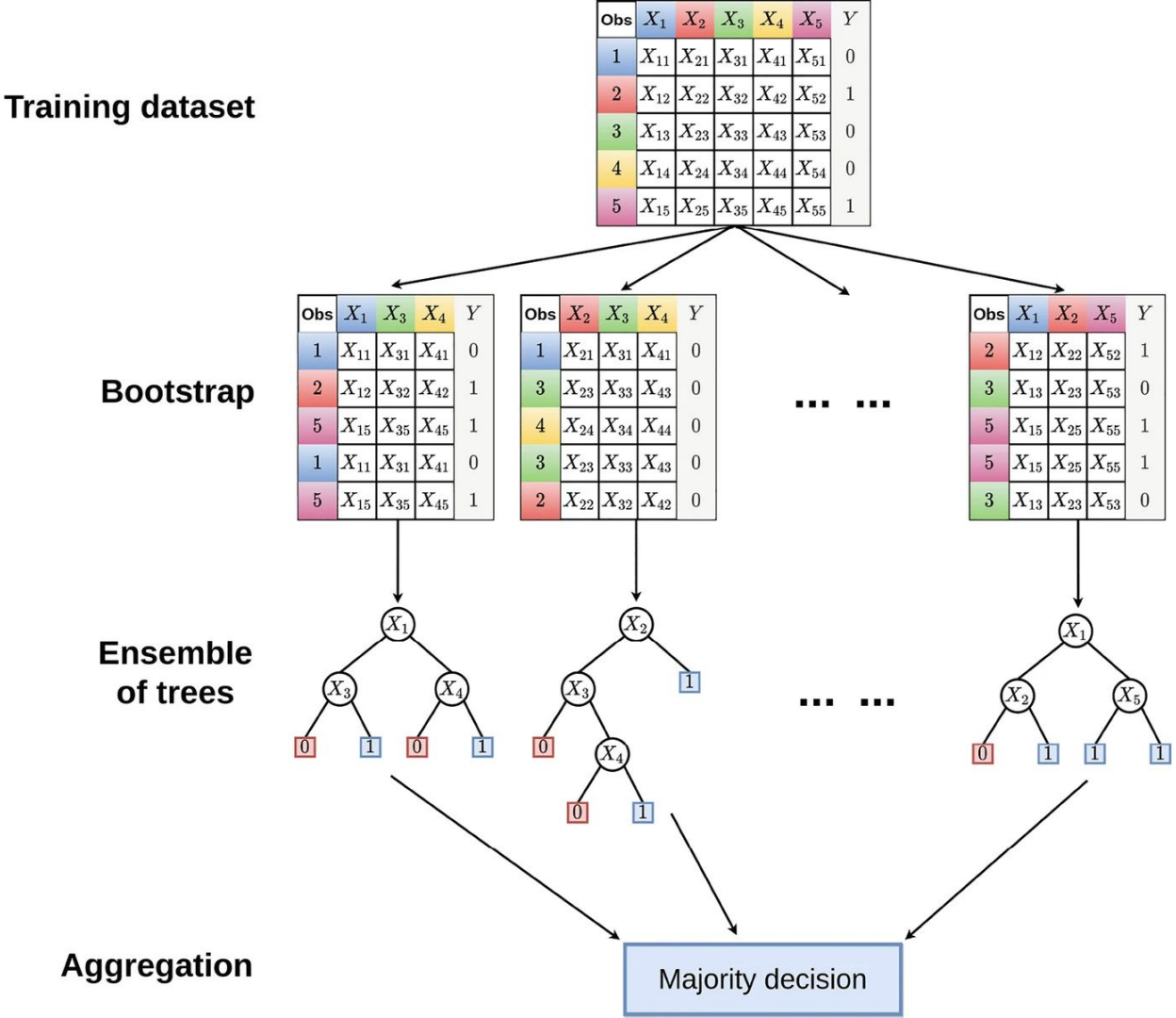
- Conviene combinar predictores débiles (sesgo bajo, varianza alta).
- Mayor independencia entre predictores base, menor error de predicción en la combinación.

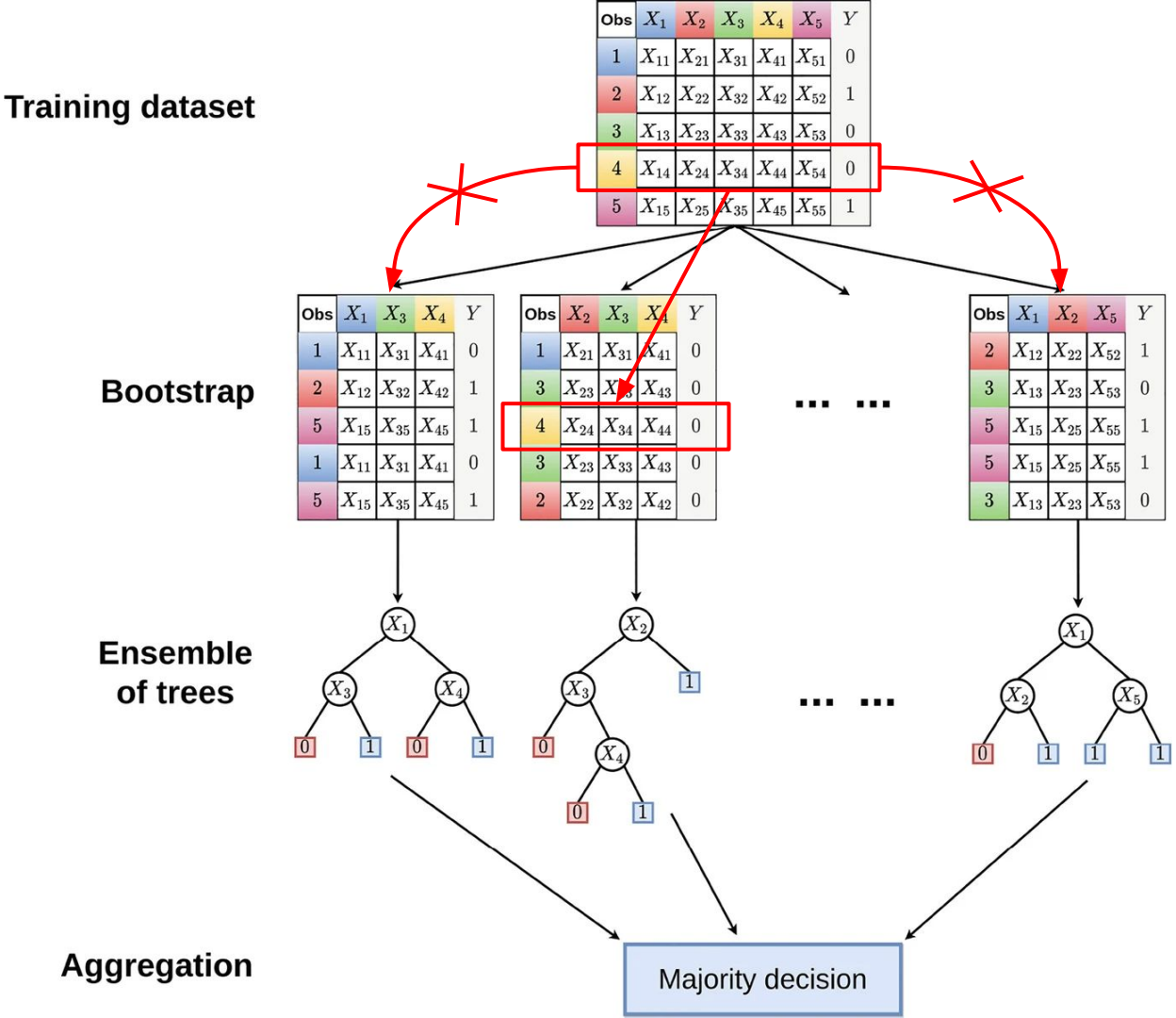
Random Forests*

- Bagging con árboles de decisión, modificado para reducir la correlación entre los árboles.
- Construyendo cada árbol sobre un subconjunto de características seleccionadas al azar.

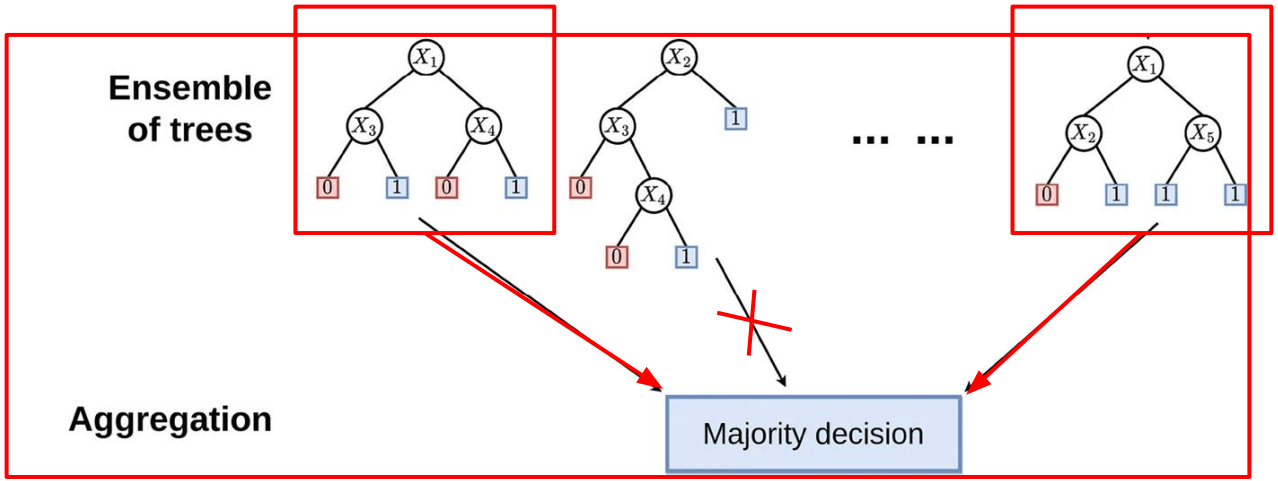
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

*L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, p. 5–32, Oct. 2001


Training dataset
Obs X_1 X_2 X_3 X_4 X_5 Y
1 X_{11} X_{21} X_{31} X_{41} X_{51} 0
2 X_{12} X_{22} X_{32} X_{42} X_{52} 1
3 X_{13} X_{23} X_{33} X_{43} X_{53} 0
4 X_{14} X_{24} X_{34} X_{44} X_{54} 0
5 X_{15} X_{25} X_{35} X_{45} X_{55} 1
Bootstrap
Obs X_1 X_3 X_4 Y
1 X_{11} X_{31} X_{41} 0
2 X_{12} X_{32} X_{42} 1
5 X_{15} X_{35} X_{45} 1
1 X_{11} X_{31} X_{41} 0
5 X_{15} X_{35} X_{45} 1
Obs X_2 X_3 X_4 Y
1 X_{21} X_{31} X_{41} 0
3 X_{23} X_{33} X_{43} 0
4 X_{24} X_{34} X_{44} 0
3 X_{23} X_{33} X_{43} 0
2 X_{22} X_{32} X_{42} 0
... ...
Obs X_1 X_2 X_5 Y
2 X_{12} X_{22} X_{52} 1
3 X_{13} X_{23} X_{53} 0
5 X_{15} X_{25} X_{55} 1
5 X_{15} X_{25} X_{55} 1
3 X_{13} X_{23} X_{53} 0
Ensemble of trees
X_1 X_3 X_4 0 1 0 1
X_2 X_3 1 0 X_4 0 1
... ...
X_1 X_2 X_5 0 1 1 1
Aggregation
Majority decision


Training dataset
Bootstrap
Ensemble of trees
Aggregation
Majority decision

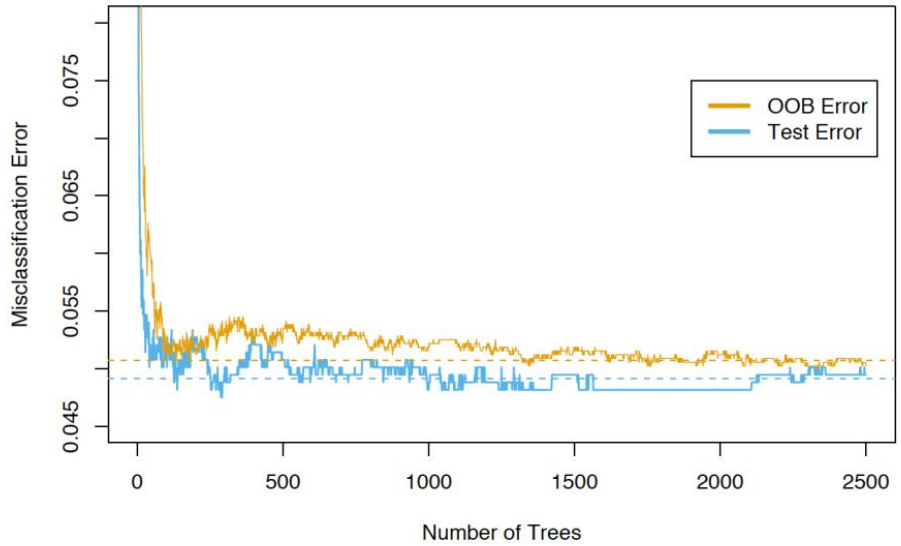
4	X_{14}	X_{24}	X_{34}	X_{44}	X_{54}



Estimación Out-of-bag (OOB)

El error de predicción se estima promediando los errores de predicción de las muestras OOB.

- Para cada muestra de entrenamiento (x_n, t_n), se construye su predictor RF promediando sólo los árboles de los muestreos bootstrap que no la contienen.




```
>>> bag_clf = BaggingClassifier(
...     DecisionTreeClassifier(), n_estimators=500,
...     bootstrap=True, n_jobs=-1, oob_score=True)
...
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.90133333333333332
```

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
>>> accuracy_score(y_test, y_pred)
0.91200000000000003
```

Feature importance

- Un subproducto de RF es la medida de la importancia de las características
- Para cada característica:
  - Se consideran los nodos donde se usa la característica
  - Se registra cuánto se reduce la impureza en ese nodo
  - Se calcula la importancia como un promedio de las reducciones de impureza ponderado por la cantidad de muestras de los nodos considerados
- Las importancias se normalizan de manera que sumen uno para el conjunto de características
- En Sklearn se calcula automáticamente y se accede con **feature_importances_**

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
>>> rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
>>> rnd_clf.fit(iris.data, iris.target)
>>> for score, name in zip(rnd_clf.feature_importances_, iris.data.columns):
...     print(round(score, 2), name)
...
0.11 sepal length (cm)
0.02 sepal width (cm)
0.44 petal length (cm)
0.42 petal width (cm)
```

Boosting

Boosting

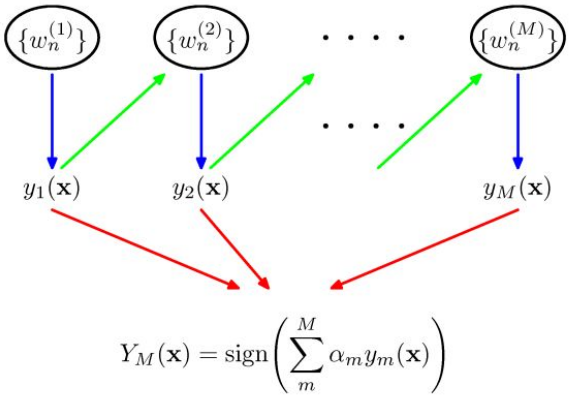
- Diferencia principal con métodos como bagging: los predictores base se entrenan secuencialmente o en cascada.

- Cada predictor se entrena condicionalmente a la performance de los predictores ya entrenados: fuerza a focalizarse en las muestras difíciles.

- El algoritmo de boosting más popular es Adaboost.*
  - Cada predictor base se entrena usando una forma ponderada de las muestras: los pesos dependen de la performance de los predictores previos en clasificar las muestras.
  - A la muestra mal predicha por un predictor base se le asignan un peso mayor a la hora de entrenar el predictor base siguiente.
  - Una vez entrenados todos los predictores de la cascada, sus predicciones se combinan mediante mayoría ponderada.

L. Breiman, “Bagging predictors,” Mach. Learn., vol. 24, p. 123–140, Aug. 1996
B. Efron, “Bootstrap methods: Another look at the jackknife,” The Annals of Statistics, vol. 7, pp. 1–26, Jan. 1979

Adaboost




```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5)
ada_clf.fit(X_train, y_train)
```

Gradient boosting

Gradient boosting

Gradient boosting* nace de la siguiente observación (más intuitiva en regresión):

- Consideremos un predictor $y = f(\mathrm{x})$, y la función de costo cuadrática,

$$L(f) = \sum_{n=1}^{N} L(y_n, f(\mathrm{x}_n)) = \sum_{n=1}^{N} (y_n - f(\mathrm{x}_n))^2 .$$

- Supongamos que queremos optimizar f iterativamente, y llamemos $f^{(m)}$ el modelo en la iteración m. Para esto se busca un nuevo estimador h que refine el predictor:

$$f^{(m+1)}(\mathrm{x}) = f^{(m)}(\mathrm{x}) + h(\mathrm{x}).$$

- Se busca que h se ajuste lo mejor posible al residuo $y - f^{(m)}(\mathrm{x})$.
- Esto se puede implementar como una cascada de predictores, donde el predictor f_m ajusta el error de aproximación obtenido por el predictor f_{m-1}.

---

* J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001

GB en Scikit-Learn

1. Entrenamos un primer árbol de decisión:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

2. Entrenamos el segundo árbol sobre los residuos del primero:

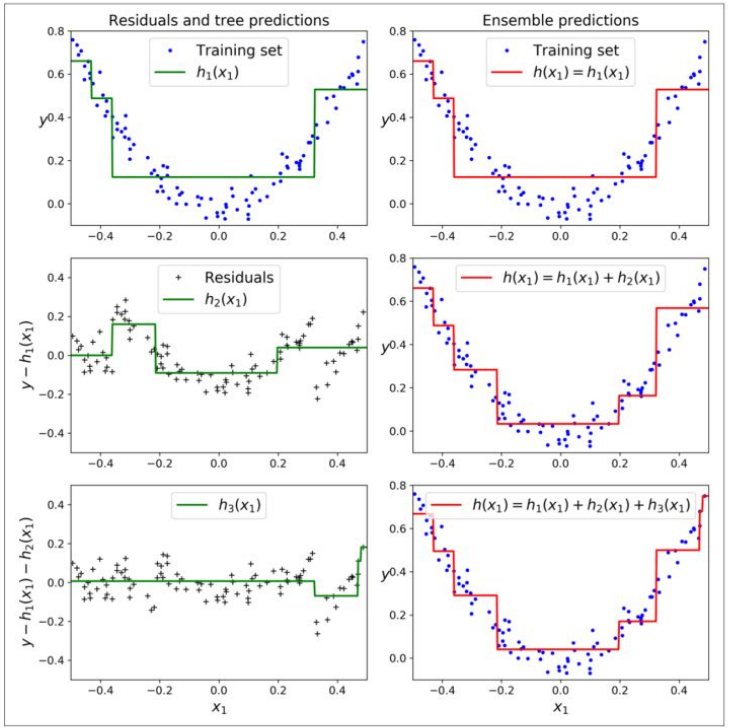
```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

3. Entrenamos el tercer árbol sobre los residuos del segundo:

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Inferencia sobre el ensamble:

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



Extreme Gradient Boosting (XGBoost)*

Librería Python de referencia†, que implementa una versión optimizada de Gradient Boosting

```
import xgboost

xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)
```

Ofrece varias funcionalidades útiles, como una implementación automática de early stopping:

```
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)], early_stopping_rounds=2)
y_pred = xgb_reg.predict(X_val)
```

---

† https://xgboost.readthedocs.io

* T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016


FACULTAD DE
INGENIERÍA
UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY