DigitalHouse > Coding School

DATA SCIENCE

Clase 45

Ensamble y Bagging

2017

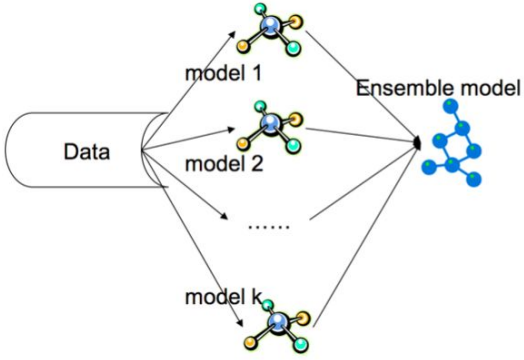
Ensamble y Bagging

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. Utilizar el clasificador bagging 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

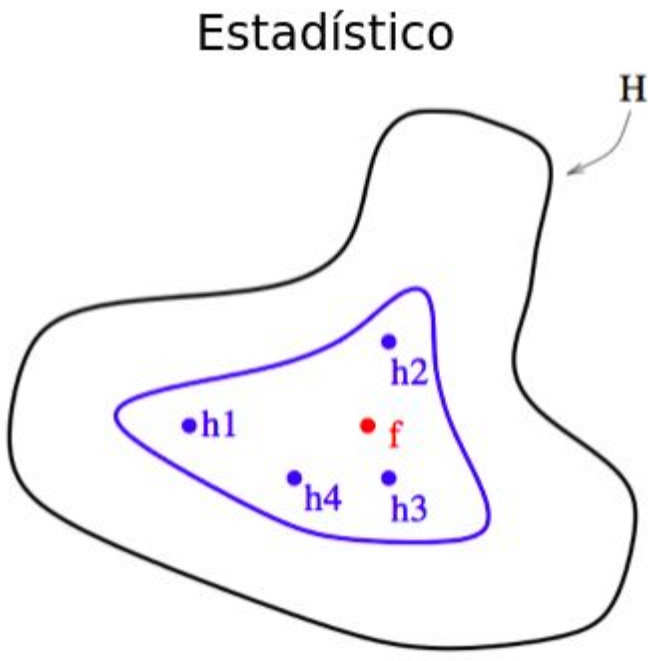
El problema estadístico

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*.

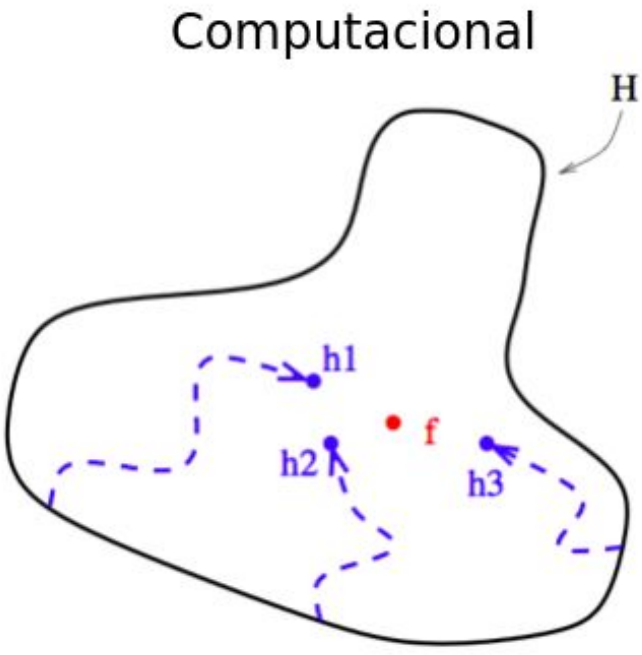


El Problema Computacional

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**.

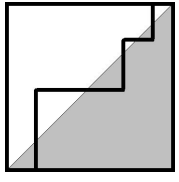
Un conjunto compuesto de varios clasificadores base con **diferentes puntos de partida** puede proporcionar una mejor aproximación de *f* que cualquier clasificador base individualmente. La verdadera función *f* es usualmente mejor aproximada usando varios puntos de partida para explorar el espacio de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



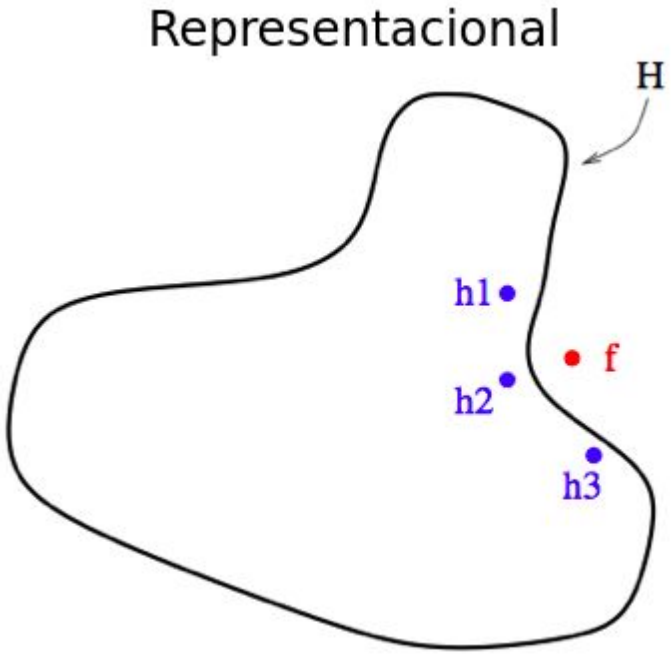
El Problema de representación

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

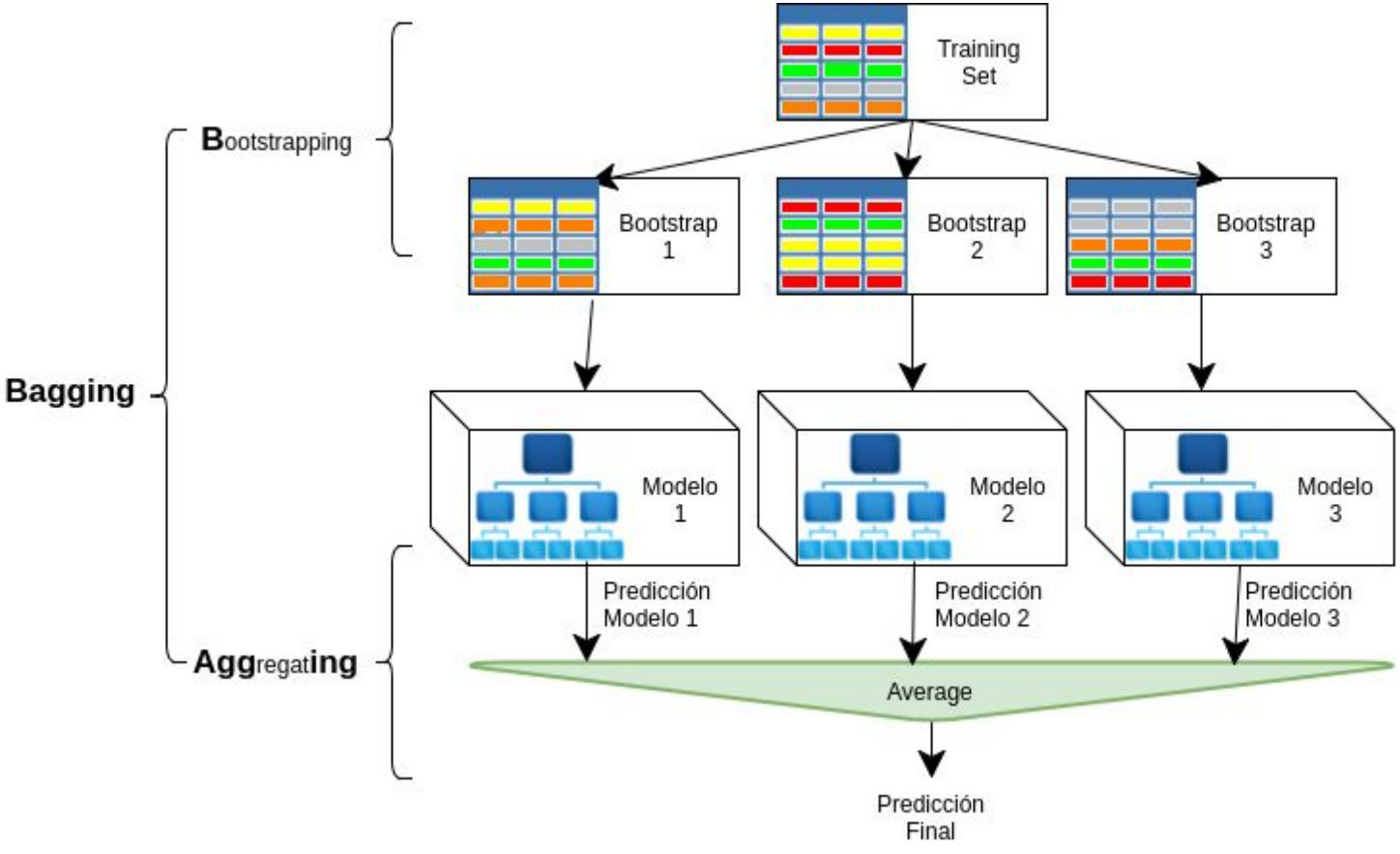
Original	1	2	3	4	5	6	7	8
set 1	2	7	8	3	7	6	3	1
set 2	7	8	5	6	4	2	7	1
set 3	3	6	2	7	5	6	2	2
set 4	4	5	1	4	6	4	3	8

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

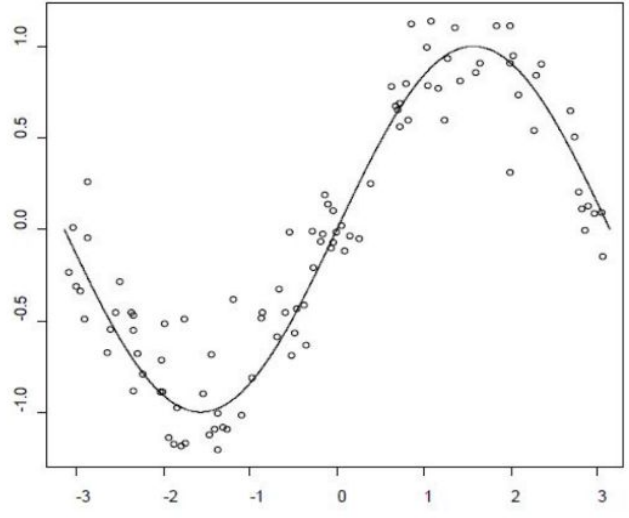
- El Bagging **reduce la varianza** del error de generalización al combinar múltiples clasificadores de base (siempre que estos satisfagan los requisitos anteriores).
- Si el clasificador **base es estable**, entonces el error del ensamble se debe principalmente al sesgo, y el bagging puede **no ser 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).


Training Set
Bootstrapping
Bootstrap 1
Bootstrap 2
Bootstrap 3
Bagging
Modelo 1
Modelo 2
Modelo 3
Predicción Modelo 1
Predicción Modelo 2
Predicción Modelo 3
Aggregating
Average
Predicción Final

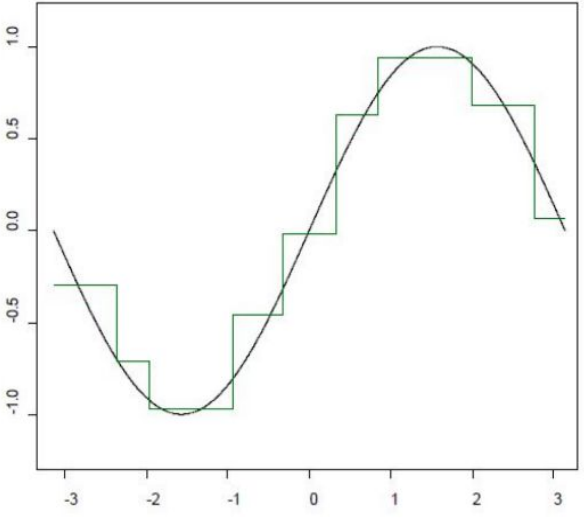
Bagging Predictors (Abstract del Paper Leo Breiman 1994)

— "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**."

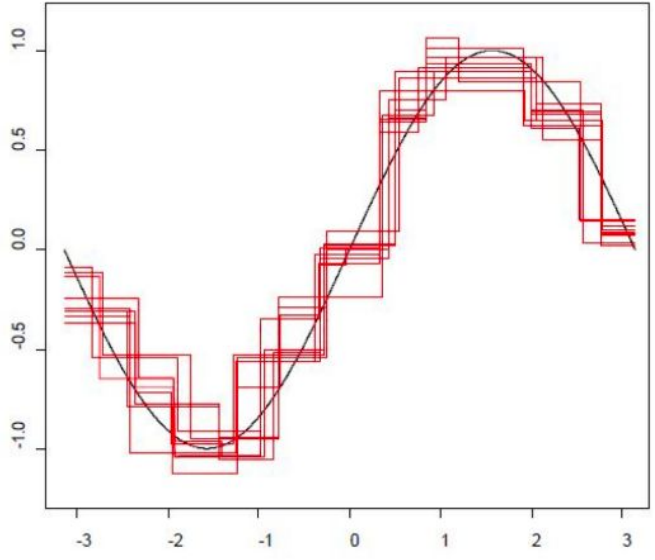
Función generadora



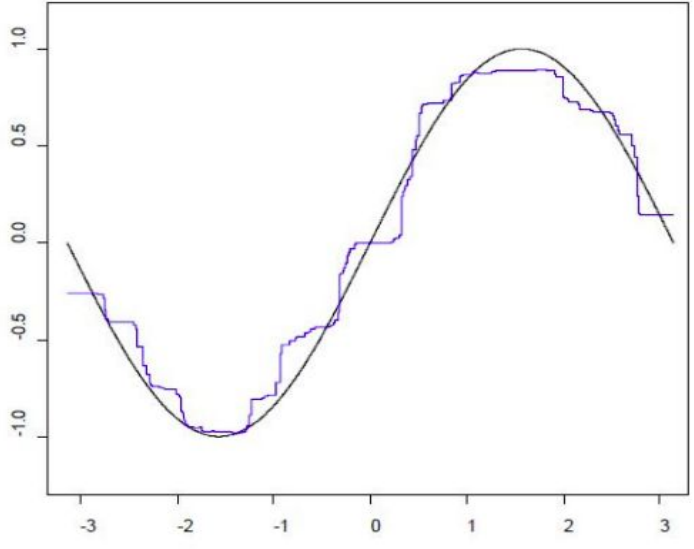
Un árbol de decisión



10 árboles de decisión



Promedio de 10 árboles de decisión



Práctica Guiada Ensambles y Bagging

El clasificador Bagging en Scikit Learn

En scikit-learn, los métodos de bagging se ofrecen como un meta-estimador unificado de BaggingClassifier (respectivamente 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*.**

Como ejemplo, vamos a comparar el rendimiento de un clasificador KNN simple versus el clasificador de Bagging en el conjunto de datos de aceptabilidad de autos.

El primer paso es leer los datos en Pandas.

```
import pandas as pd
df = pd.read_csv('./datasets/car.csv')
df.head()
```

	buying	maint	doors	persons	lug_boot	safety	acceptability
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Dado que la mayoría de los atributos son texto categórico, tendremos que codificarlos como números usando el LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
features = [c for c in df.columns if c != 'acceptability']
for c in df.columns:
    df[c] = le.fit_transform(df[c])

X = df[features]
y = df['acceptability']
X.head()
```

	buying	maint	doors	persons	lug_boot	safety
0	3	3	0	0	2	1
1	3	3	0	0	2	2
2	3	3	0	0	2	0
3	3	3	0	0	1	1
4	3	3	0	0	1	2

Aquí hemos sobrescrito los atributos originales por simplicidad, ya que no estamos interesados en hacer un estudio sobre la importancia de los mismos.

El siguiente paso es calcular el cross_val_score en los dos clasificadores:

```
from sklearn.cross_validation import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
bagging = BaggingClassifier(knn, bootstrap_features=True, random_state=1)

print "KNN Score:\t", cross_val_score(knn, X, y, cv=5, n_jobs=-1).mean()
print "Bagging Score:\t", cross_val_score(bagging, X, y, cv=5, n_jobs=-1).mean()
```

```
KNN Score:        0.643070305149
Bagging Score:    0.751220688265
```

Detalles del Clasificador Bagging

El meta clasificador BaggingClassifier tiene varios parámetros.

En pares, consulten la documentación para obtener una descripción detallada de cada uno y averiguar qué hacen max_samples, max_features y n_estimators.

LAB Ensambles y Bagging

CONCLUSIÓN

- Vimos los modelos Ensamble y Clasificadores Bagging

- 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.