**Introducción al machine learning**

Bienvenido al curso de introducción al machine learning.

Machine learning (ML) es una herramienta para crear modelos predictivos a partir de conjuntos de datos. Cuantos más datos tengas, más complejos serán los programas que puedas escribir.

Los modelos de aprendizaje automático pueden producir información automatizada de los datos y mejorar sin ser programados explícitamente.

**Estructura del curso**

A lo largo de este curso intentarás resolver un ejercicio empresarial para desarrollar un sistema de precios inmobiliarios. Para realizar el ejercicio, entrenarás tu primer modelo y examinarás qué tan bien funciona. Aprenderás las métricas para evaluar su calidad, además de cómo prevenir errores comunes.

Posteriormente, trabajarás para mejorar la calidad de tu modelo. La forma más sencilla de hacerlo es seleccionar diferentes algoritmos y ajustar sus hiperparámetros. Explorar las posibilidades te ayudará a familiarizarte con un flujo de trabajo normal y te facilitará el proceso de aprendizaje.

Como prueba final de este curso, trabajarás en un proyecto por tu cuenta. Dicho proyecto consiste en usar tus habilidades de machine learning para crear un modelo para recomendar un nuevo plan móvil a los clientes de una compañía telefónica. Haz clic [aquí](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/moved_DS_7_Sprint_Proyecto.pdf) para ver la descripción del proyecto y lo que implica.

**Tus objetivos:**

* Entender la terminología básica de machine learning, por ejemplo, la diferencia entre clasificación y regresión, o la diferencia entre un modelo y un algoritmo de aprendizaje.
* Dominar la librería *scikit-learn* y aprender a medir las métricas de evaluación y a entrenar modelos.
* Aprender a examinar modelos y elegir el mejor.

En este sprint desarrollarás tus habilidades en:

Código QR

Descripción generada automáticamente

Imagen que contiene Código QR

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Código QR

Descripción generada automáticamente

**¿Cuánto tiempo llevará?**

Esta sección es un desafío moderado: no incluye muchos ejercicios, pero necesitarás comprender muchos términos y conceptos nuevos. Espera pasar entre 30 y 50 horas para completar este material, dependiendo de tus conocimientos previos y hábitos de estudio. Si sientes que te estás quedando atrás, no dudes en ponerte en contacto con nuestro equipo de orientación para informarles de esto. Como siempre, nos comprometemos a ayudarte en cada paso del camino.

Sprint 8

Capítulo 2/7

Entrenar tu primer modelo

**Introducción**

Ya conoces Python. Ahora sabes cómo preparar datos para el análisis, estudiar relaciones usando métodos estadísticos y trazar gráficos. Es hora de abordar el machine learning. Te daremos una tarea empresarial y entrenarás un modelo para resolverla.

**Empezarás por:**

* Aprender terminología básica de machine learning
* Descubrir qué es el aprendizaje supervisado y aprender los tipos de ejercicios que existen
* Explorar la librería Scikit-learn
* Entrenar tu primer modelo

**¿Cuánto tiempo tomará?**

Diez lecciones de 5-10 minutos cada una.

**Descripción del ejercicio**

Los posibles compradores de bienes raíces recurren con frecuencia a los servicios en línea porque desean encontrar una buena oferta. Por supuesto, al mismo tiempo, el objetivo de los vendedores es vender su propiedad al precio más alto posible. Además, el servicio en línea que ayuda a conectar a compradores y vendedores obviamente desea obtener ganancias (incluso si el margen no es alto). Uno de esos servicios notó que muchas veces los vendedores no fijan el precio de su propiedad de acuerdo con el valor de mercado, y que estos precios más altos frustraron a los posibles compradores. ¿Será posible eliminar el factor humano y fijar el precio de los artículos de forma automática mediante machine learning? Vamos a averiguarlo.

**Descripción del ejercicio empresarial**

Comprender el ejercicio es la mitad de la solución. Veamos lo que se debe hacer y descubramos cómo resolver el problema.

Imagina que estás trabajando en una plataforma de listas de bienes raíces. En lugar de utilizar los servicios de la agencia mobiliaria , los propietarios envían sus propias listas y los compradores pueden responderles directamente. Si una transacción se realiza con éxito, la plataforma se lleva una comisión.

Los análisis de sitios web mostraron que, con frecuencia, los propietarios no basan sus precios en el valor de mercado. Esta práctica siempre es mala para el sitio web; como resultado, las propiedades económicas se venden rápidamente, pero la comisión de la plataforma también es menor. Por otro lado, las propiedades sobrevaloradas rara vez se venden, lo que significa que no hay ninguna ganancia. El servicio necesita animar a los vendedores a fijar el precio de su propiedad de acuerdo con su valor de mercado. Necesitamos encontrar un algoritmo para ayudar a los propietarios a determinar el precio correcto.

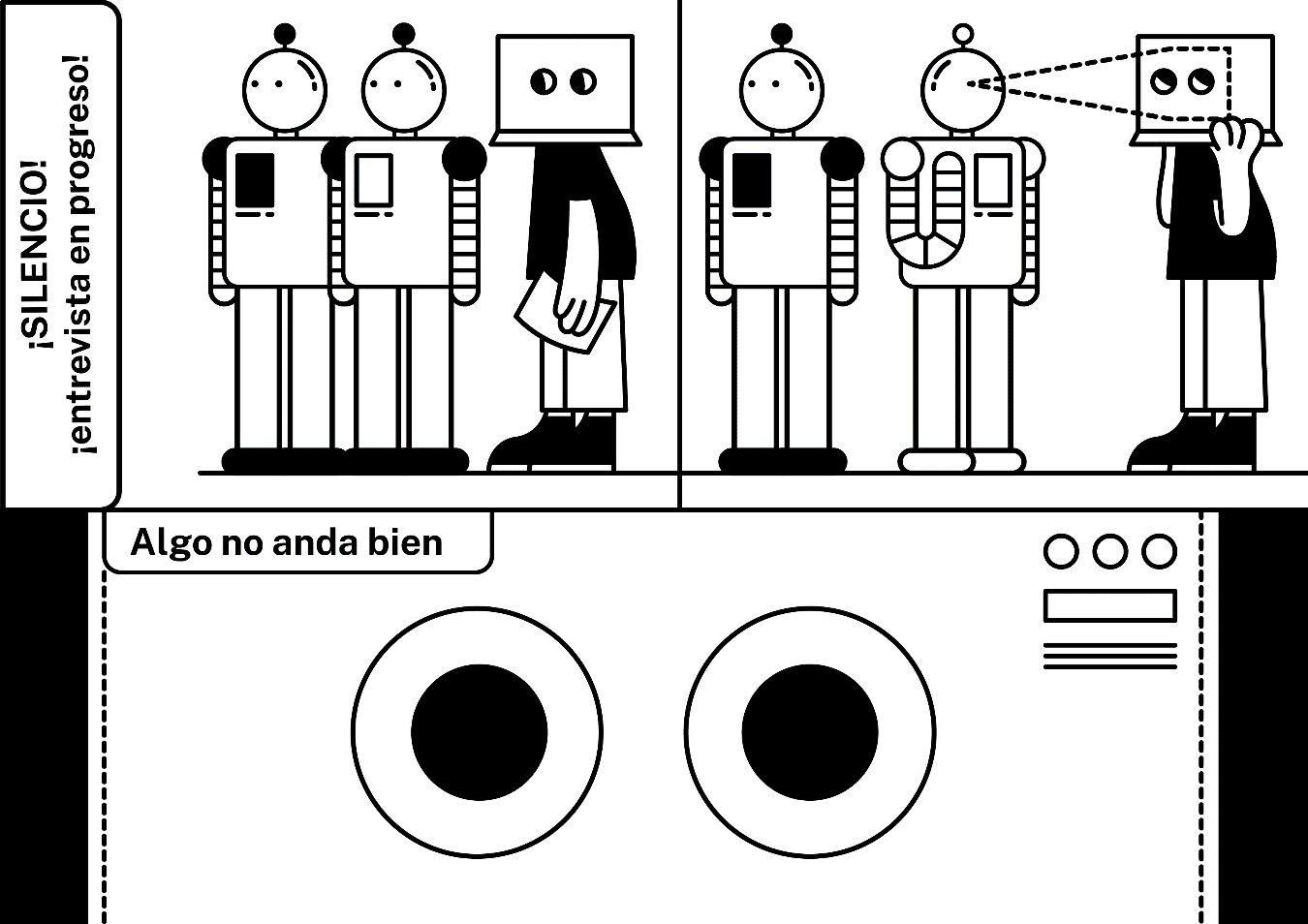
Para resolver este problema, deberás trabajar con los expertos para crear las reglas de precios. Por ejemplo:

* Multiplicar el área del apartamento por el precio promedio de la ciudad por m²
* Disminuir el precio en un 20 % si el apartamento se remodeló hace más de 3 años
* Incrementar el precio en un 30 % si hay una estación de metro cerca

El número de reglas posibles es ilimitado.

Pero eso no es todo. Un "algoritmo experto" podría ser una ventaja competitiva o una característica costosa e inútil. Las reglas también pueden ser difíciles de escalar (por ejemplo, al ingresar a otros mercados y regiones), por lo que con el tiempo pueden volverse irrelevantes.

El machine learning te ayudará a superar las deficiencias de las reglas expertas.



# **Dataset para el entrenamiento**

Descartaste a los expertos. El programa resolverá la tarea en cuanto terminemos de entrenarlo.

En realidad, los programas y los expertos se capacitan de manera similar: recopilan y clasifican conocimientos, descubren dependencias y adquieren experiencia. Tanto el aprendizaje humano como el machine learning incluyen alguna forma de estudiar la información. En el caso del machine learning, los modelos aprenden de conjuntos de datos de entrenamiento.

Para esta tarea, vamos a utilizar los datos presentados por un mercado inmobiliario online. Para que esta sea adecuada para entrenar un modelo, hemos eliminado las variables que no afectan el precio, así como los valores faltantes y las propiedades que están fuera de los límites de la ciudad. El nombre del archivo es /datasets/train\_ml.csv. ¡Vamos a descargarlo!

import pandas as pd

df = pd.read\_csv('/datasets/train\_data\_us.csv')

print(df.shape)

print(df.head(5))

(6495, 14)

last\_price bedrooms kitchen\_area living\_area total\_area balconies \

0 108000.0 2 6.6 31.5 59.0 31.5

1 264000.0 4 12.2 72.0 109.0 72.0

2 140000.0 3 10.8 49.0 74.5 49.0

3 64000‬.0 1 6.2 20.0 37.4 20.0

4 133000‬.0 3 10.4 41.9 64.9 41.9

ceiling\_height floors\_total floor bike\_parking is\_studio is\_open\_plan \

0 2 0.0 0.0 0.0 6.6 0.0

1 2 0.0 0.0 0.0 12.2 0.0

2 9 0.0 0.0 0.0 10.8 0.0

3 4 0.0 0.0 0.0 6.2 2.0

4 11 0.0 0.0 0.0 10.4 0.0

airport\_dist city\_center\_dist

0 20485.0 8180.0

1 42683.0 8643.0

2 14078.0 16670.0

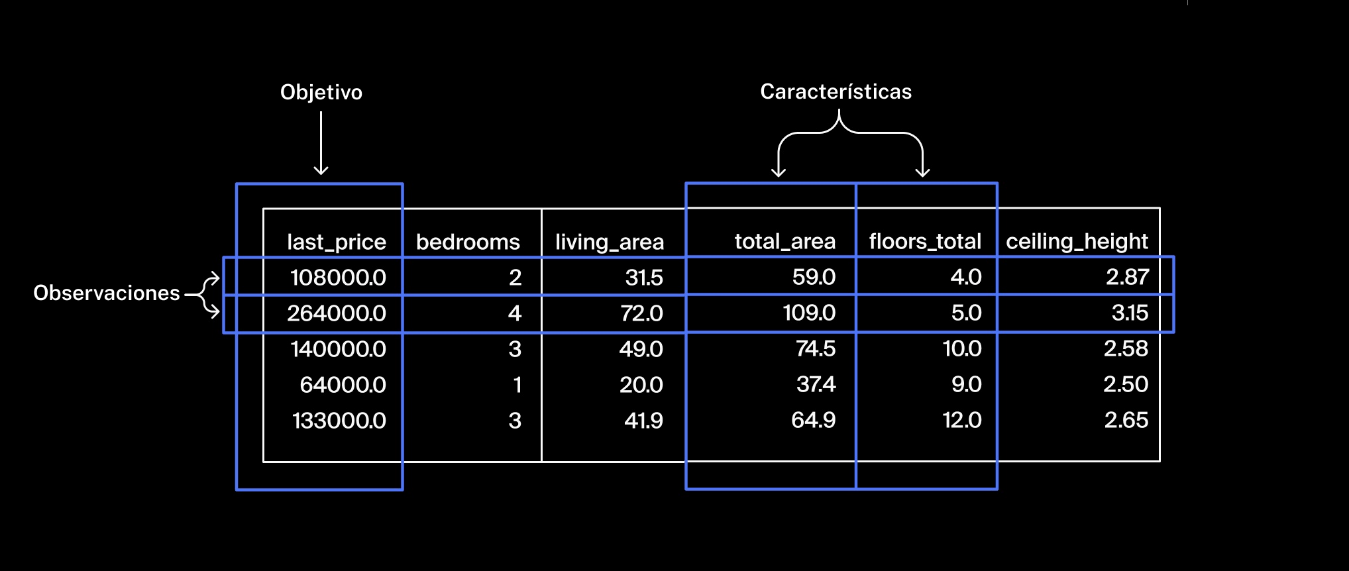
3 17792.0 17699.0

4 14767.0 10573.0

Veamos los diferentes tipos de datos en nuestro conjunto de datos.

* last\_price — precio al cierre (en dólares)
* total\_area — superficie del apartamento en metros cuadrados (m²)
* bedrooms — número de dormitorios
* ceiling\_height — altura del techo (m)
* floors\_total — número total de pisos en el edificio
* living\_area — superficie de sala de estar (m²)
* floor — piso
* bike\_parking — estacionamiento de bicicletas en el edificio (tipo de dato booleano)
* is\_studio — la propiedad es un estudio (tipo de dato booleano)
* is\_open\_plan — plan abierto (tipo de dato booleano)
* kitchen\_area — área de cocina (m²)
* balconies — número de balcones
* airport\_dist — distancia al aeropuerto más cercano en metros (m)
* city\_center\_dist — distancia al centro de la ciudad (m)

En el machine learning, las filas y columnas representan observaciones y características, respectivamente. La característica que necesitamos predecir se conoce como el objetivo. En esta tarea, el objetivo es last\_price.



# Aprendizaje supervisado

Existen distintas tareas de machine learning. Para desarrollar el algoritmo correcto, primero debes determinar el tipo de tarea.

Tienes un conjunto de datos de entrenamiento y una característica objetivo (precio de venta de la propiedad) que necesitas predecir usando el resto de las características. Esta es una tarea de aprendizaje supervisado. El "maestro" plantea preguntas (características) y da respuestas (el objetivo). No se da ninguna explicación sobre cómo las características conducen a la respuesta exactamente; la máquina tiene que resolverlo por sí misma. Con desafíos como estos, ¡no es de extrañar que en las películas ocurran levantamientos de IA todo el tiempo!

La mayor atención se centrará en el aprendizaje supervisado, ya que es adecuado para resolver muchas tareas comerciales. Más adelante, también abordaremos otras clases:

* Aprendizaje no supervisado: sin objetivo
* Aprendizaje semisupervisado: solo una parte de los datos de entrenamiento conoce el objetivo
* Recomendación: los usuarios y los elementos reemplazan las funciones y las observaciones (algo que puedas recomendar, por ejemplo, películas o vecindarios).

Pregunta

Identifica las tareas que pertenecen al aprendizaje supervisado:

Elige tantas como quieras

Determinar si el usuario hará clic o no en el banner publicitario en función del contenido del sitio web y el historial de clics.

El contenido del sitio corresponde a características; hacer clic/no hacer clic es el objetivo y el historial de clics es nuestro conjunto de datos de entrenamiento. Todo está claro. La máquina te lo agradecerá.

Dividir a los clientes del mercado online en grupos en función de su historial de compras.

Determinar el orden de relevancia de los sitios web mostrados en los resultados de búsqueda.

Identificar la edad de un usuario con base en el historial de compras. Tienes acceso al historial de compras de los clientes cuyas edades se conocen.

La lista de compras es una función, la edad es el objetivo y el historial de compras es el conjunto de datos de entrenamiento.

Bien, puedes quedarte con tu ropa, tus botas y tu motocicleta.

¡Excelente trabajo!

Veamos los tipos de aprendizaje supervisado.

Todas las variables y características son categóricas o numéricas, y el objetivo no es una excepción.

Las tareas de clasificación se ocupan de objetivos categóricos (por ejemplo, determinar especies de animales en una imagen). Si solo tenemos dos categorías, por ejemplo, si un cliente volverá a visitar el sitio web o no, es una clasificación binaria.

Si el objetivo es numérico, entonces es una tarea de regresión. Los datos se utilizan para encontrar relaciones entre las variables y hacer predicciones basadas en la información, como el pronóstico de volumen de precipitaciones o la predicción de los precios del mercado de valores para los próximos días.

Pregunta

Tus colegas del trabajo te pidieron que pronosticaras la cantidad de cortadoras de verduras que se venderán el próximo mes. ¿Esta tarea es de regresión o clasificación?

Regresión

No se necesita saber ingeniería aeroespacial para resolverlo. El objetivo es numérico.

Clasificación

¡Tu comprensión del material es impresionante!

Pregunta

¡Tienes spam! O un correo normal. Antes de hacer conjeturas, decidamos si se trata de una tarea de regresión o de clasificación.

Regresión

Clasificación

¡Felicidades! Tienes un objetivo categórico con solo dos opciones (spam/no spam), por lo que es una clasificación binaria.

¡Lo has entendido bien!

Pregunta

Debes identificar el dígito escrito a mano que se muestra en la imagen (del 0 al 9). ¿Es esta una tarea de regresión o de clasificación?

Regresión

Clasificación

Es un objetivo categórico. Intentamos engañarte, pero no funcionó.

¡Bien hecho!

Teoríatrain\_data\_us.csv

# Clasificación y regresión

Volvamos a nuestro problema de vivienda y decidamos qué es mejor usar: clasificación o regresión.

El precio del apartamento es un objetivo numérico, por lo que se trata de una tarea de regresión. La regresión suele implicar largos cálculos con muchas respuestas posibles, por lo que las tareas de regresión no son la forma más sencilla de familiarizarse con el machine learning. Para simplificar, dividiremos todos los precios en "altos" y "bajos" por ahora, convirtiendo efectivamente nuestra tarea en una tarea de clasificación binaria con solo dos respuestas posibles. Entonces todo lo que tenemos que hacer es predecir en qué clase cae cualquier lista dada. Nos ocuparemos de la regresión más tarde.

Entonces, ¿cómo se dividen exactamente los precios en altos y bajos? Es más fácil cuando hay un número casi igual de objetos en las categorías. ¿Por qué? ¡Porque es difícil distinguir las cornejas de los cuervos si solo estás viendo a las cornejas!

Averigüemos el precio medio (justo en el medio).

import pandas as pd

df = pd.read\_csv('/datasets/train\_data\_us.csv')

print(df['last\_price'].median())

113000.0

Los precios superiores a $113 000 son altos, mientras que cualquier valor inferior se considera bajo.

Ejercicio 1

Cambia la tarea original a una tarea de clasificación. Crea una nueva característica llamada price\_class. Para precios mayores a $113 000, asigna price\_class a 1. Para precios menores o iguales a $113 000, asigna 'price\_class' a 0. Imprime las primeras cinco filas de la tabla (que ya están en precodificación).

Usa el método loc() para acceder a elementos especificando su fila y columna.  
  
import pandas as pd

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear la nueva columna 'price\_class' basada en la condición dada

df['price\_class'] = (df['last\_price'] > 113000).astype(int)

print(df.head())  
  
Ejercicio 2  
Muchas librerías de machine learning requieren que las características se almacenen en variables separadas. Declara dos variables:

features para características

target para objetivo

Imprime los tamaños de estas variables en pantalla (que ya están en el precódigo).

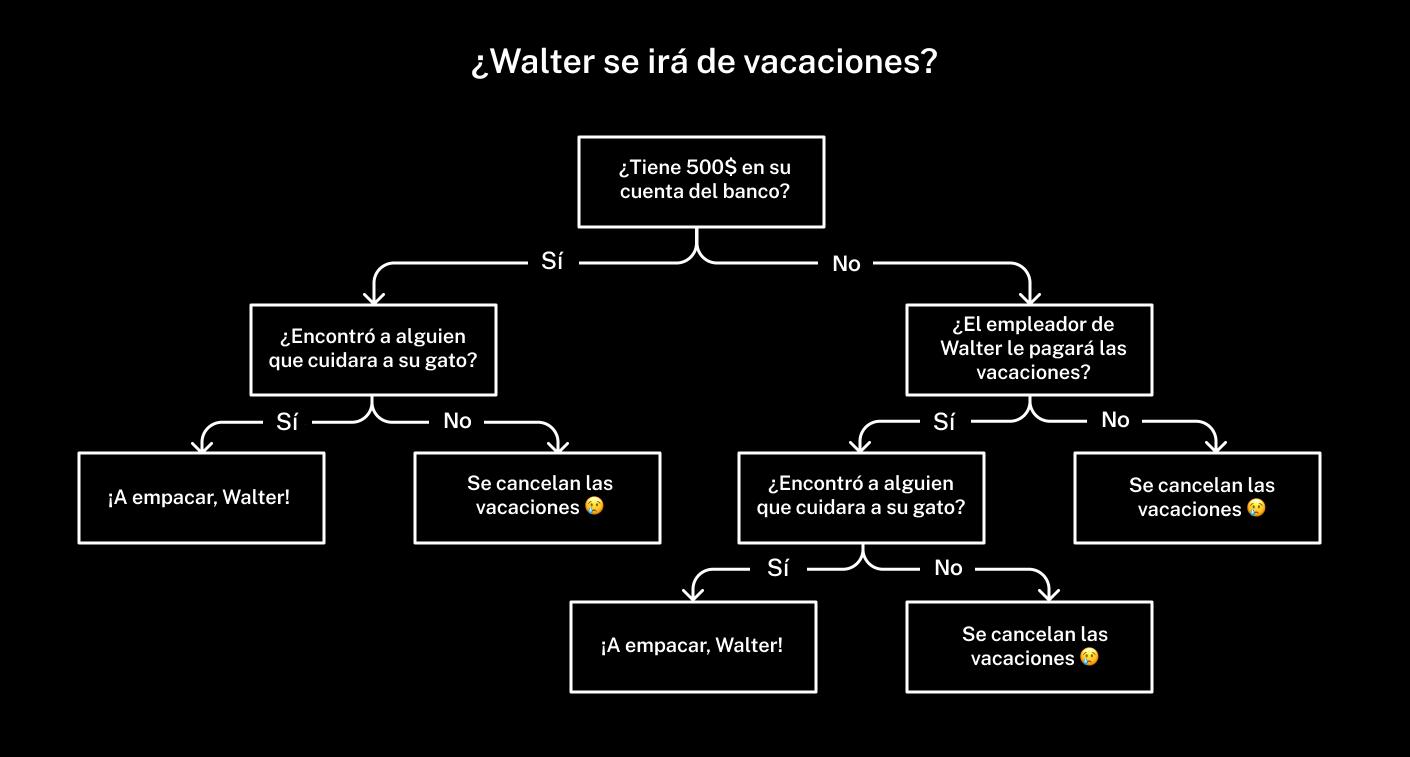
Todas las columnas, excepto last\_price y price\_class son características. price\_class es el objetivo. Ya no necesitamos la columna last\_price.

# Modelos y algoritmos

Veamos más de cerca los principios del machine learning, sus etapas y el propósito de los algoritmos.

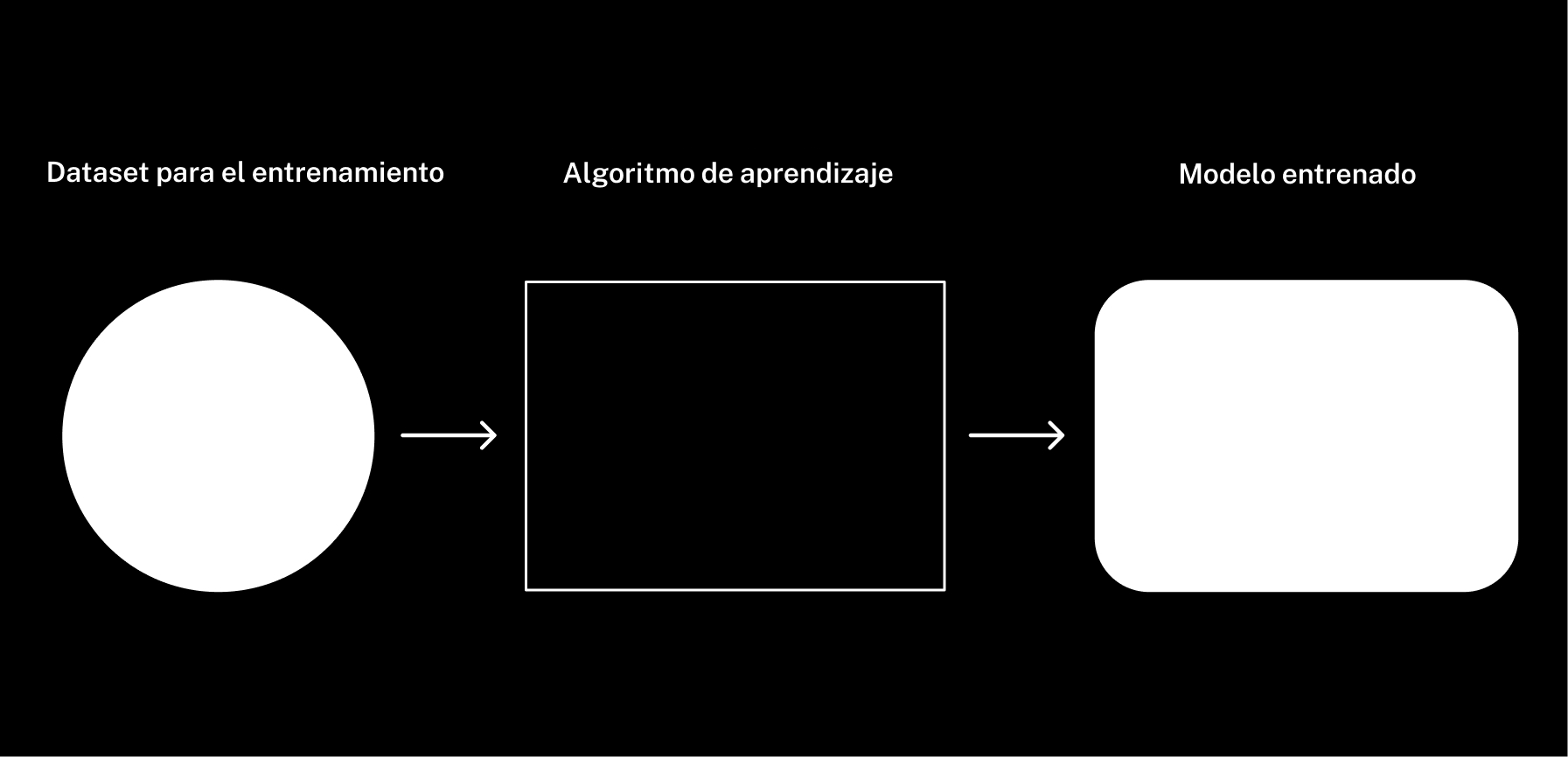
Para hacer predicciones debes comprender la relación entre las otras características de tu conjunto de datos y el objetivo. Los científicos de datos hacen suposiciones sobre esta relación y luego hacen predicciones basadas en dichas suposiciones. Si las predicciones son ciertas, la suposición es correcta. Este enfoque se denomina modelado. Las suposiciones y los métodos de predicción se denominan modelos de machine learning.

Un modelo popular se llama árbol de decisiones. Este puede describir el proceso de toma de decisiones en casi cualquier situación. Por ejemplo, determinar si Walter irá de vacaciones a Hawái o si Julia obtiene la aprobación de su préstamo para automóvil para un nuevo *Alfa Romeo Giulietta*. Veamos las condiciones. ¿Walter tiene suficiente dinero? ¿Encontró alguien que cuidara a su gato? Si se cumplen todas las condiciones, Walter puede ir a comprar un bañador. Con Julia no es tan simple. Julia tiene menos de 18 años, por lo que definitivamente no podrá obtener un préstamo. Así es como hacemos un árbol de decisiones con respuestas sí/no y diferentes escenarios.



Para nuestra tarea, supondremos que *el árbol de decisión determina el precio del apartamento*.

Cada árbol sale diferente. Entrenaremos el modelo para construir el más adecuado. Además del conjunto de datos, necesitaremos un algoritmo de aprendizaje. El conjunto de datos se procesa a través de nuestro algoritmo de aprendizaje, produciendo un modelo entrenado.



Pregunta

Elige el enunciado correcto:

Tanto los datos como el modelo entrenado son entradas para el algoritmo de aprendizaje.

El conjunto de datos de entrenamiento no incluye un objetivo.

Un árbol de decisión es un algoritmo de aprendizaje.

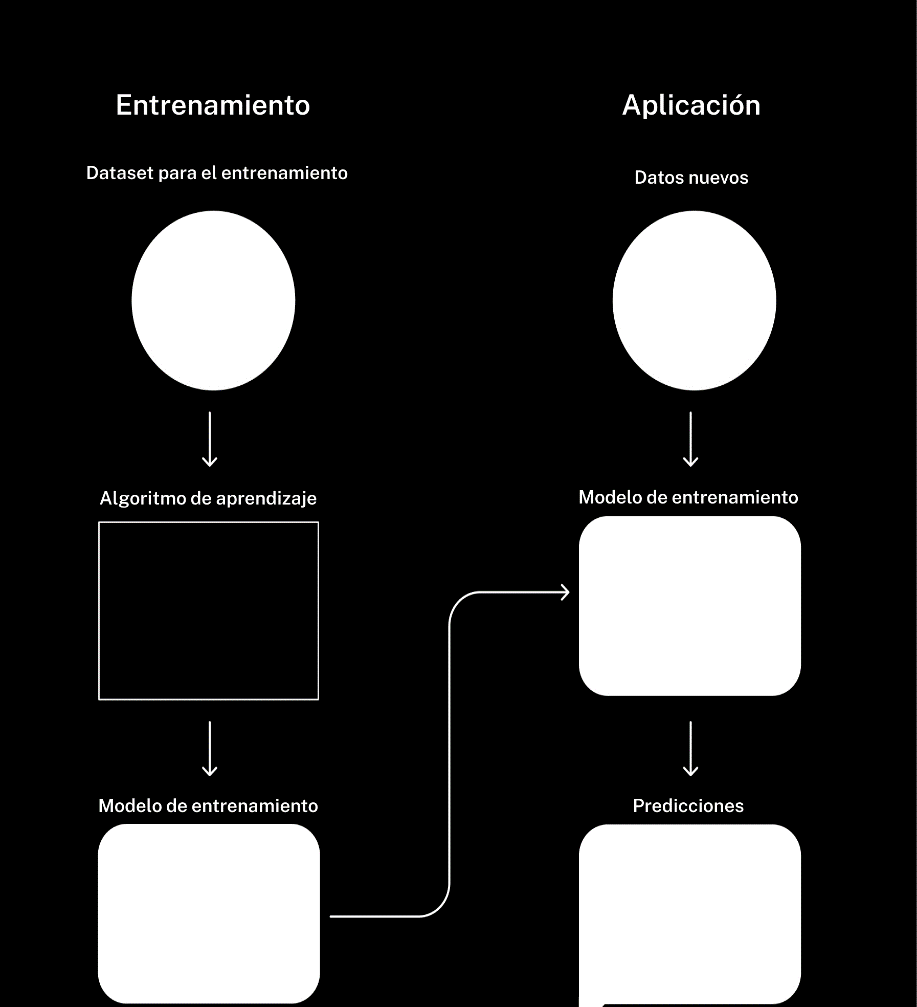
El modelo entrenado y el algoritmo de aprendizaje dependen de un conjunto de entrenamiento.

En el aprendizaje automático, no llegarás muy lejos sin un conjunto de entrenamiento. Lo necesitas para alimentar el algoritmo de aprendizaje con el fin de obtener tu modelo de entrenamiento.

¡Excelente trabajo!

Después del entrenamiento, el modelo está listo para hacer predicciones: para incorporar nuevas funciones como respuestas de entrada y salida (el objetivo), sin necesidad de aprender algoritmos ni conjuntos de datos de entrenamiento.

Es importante recordar que el proceso de aprendizaje automático consta de dos pasos: entrenamiento del modelo y aplicación del modelo.



Pregunta

Elige el enunciado correcto:

El modelo entrenado toma preguntas (características) y respuestas (el objetivo) como entrada.

El algoritmo de aprendizaje toma el conjunto de datos de entrenamiento y nuevos datos como entrada para predecir las respuestas.

Para que el modelo funcione bien, debe elegir el algoritmo de aprendizaje correcto.

Correcto. El modelo hace predicciones, pero el algoritmo de aprendizaje entrena al modelo. Si eliges un algoritmo inadecuado, obtendrás un modelo que no funciona.

Demasiados datos de entrenamiento ralentizan el modelo entrenado. Si hay demasiados por procesar, estos no ayudan en absoluto.

¡Perfecto!

# Librería Scikit-Learn

Exploremos las cajas negras y aprendamos a trabajar con una nueva librería antes de pasar a la programación.

Los algoritmos de aprendizaje suelen ser más complejos que los modelos. Entonces, por ahora, piensa en ellos como cajas negras y concéntrate en lo que debes usar como entrada y qué hacer con la salida, en lugar de enfocarte en lo que sucede dentro.

El concepto de caja negra se puede comparar con pedir pizza. Para pedir una pizza, todo lo que tienes que hacer es elegir los ingredientes, dar tu dirección y esperar. En realidad, no te importa lo que sucede entre el pedido de la pizza y el repartidor que llama a la puerta. Si alguna vez quisieras abrir tu propia pizzería (es decir, hacer un trabajo académico sobre la mejora de algoritmos), necesitarías conocer todo el funcionamiento interno. Por ahora, deja la cocina a los profesionales entrenados. ¿Entiendes? ¿Entrenados... como en "modelo entrenado"?

Pero ¿dónde podemos conseguir una de estas cajas negras para entrenar a nuestro modelo? Las librerías de Python ofrecen muchos algoritmos. En esta lección trabajaremos con la popular librería scikit-learn o sklearn (*kit científico para aprender*).

*Scikit-learn* es una gran fuente de herramientas para trabajar con datos y modelos. La librería se divide en módulos para mayor comodidad. Los árboles de decisión se almacenan en el módulo tree.

Cada modelo corresponde a una clase separada en *scikit-learn*. DecisionTreeClassifier es una clase para clasificaciones de árboles de decisión. Vamos a importarla desde la librería:

from sklearn.tree import DecisionTreeClassifier

Luego creamos una instancia de la clase:

model = DecisionTreeClassifier()

La variable model ahora almacena nuestro modelo, y tenemos que ejecutar un algoritmo de aprendizaje para entrenar el modelo para hacer predicciones.

Librería Scikit-Learn

Ejercicio 1

Comencemos con el entrenamiento del modelo. En la Lección 5 guardamos el conjunto de datos de entrenamiento en las variables features y target. Para iniciar el entrenamiento, llama al método fit() y pásale tus variables como argumento.

model.fit(features, target)

Finaliza el código e imprime la variable model en la pantalla (ya en el precódigo).

Copia y pega las tres strings de código que viste antes en esta lección.

import pandas as pd

from sklearn import set\_config

# no cambies estos parámetros de configuración

set\_config(print\_changed\_only=False)

# importa el árbol de decisión de la librería sklearn

from sklearn.tree import DecisionTreeClassifier

# < escribe tu código aquí >

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

model = DecisionTreeClassifier()# crea un modelo vacío y asígnalo a una variable

model.fit(features, target)# entrena un modelo llamando al método fit()

# < escribe tu código aquí >

print(model)

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',

max\_depth=None, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, random\_state=None,

splitter='best')

Ejercicio 2

Ahora tenemos un modelo entrenado en la variable model . Para predecir respuestas, llama al método predict() y pásale la tabla con las características de las nuevas observaciones.

answers = model.predict(new\_features)

Crea dos nuevas observaciones y verifica los resultados de la predicción. Recuerda que todo lo que está por encima y por debajo del precio mediano se etiqueta con las clases de precio 1 y 0, respectivamente. Las observaciones en nuestra tarea son apartamentos. Escribe los valores de las características para cada observación:

1. El primer apartamento tiene 12 dormitorios con una superficie total de 900 m². La superficie de la sala de estar es de 409.7 m² y la superficie de la cocina es de 112 m².
2. El segundo apartamento tiene 2 dormitorios con una superficie total de 109 m². La superficie de la sala de estar es de 32 m² y la de la cocina es de 40.5 m².

Casi no hay diferencia entre las características restantes. Ya las hemos incluido en el precódigo. No proporcionamos precio para estos dos casos experimentales, y se espera que nuestro clasificador entrenado adivine la clase correcta usando las características que especificamos. Predice la respuesta y muéstrala en la pantalla.

Las características que necesitas cambiar son: total\_area, bedrooms, living\_area y kitchen\_area.

Utiliza estas plantillas:

new\_features.loc[0, 'total\_area'] =....

new\_features.loc[1, 'total\_area'] =.... y etc.

Luego llama al método model.predict() y aplícalo al DataFrame new\_features. Guarda la predicción como answers. Muéstrala en la pantalla.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

model = DecisionTreeClassifier()

model.fit(features, target)

new\_features = pd.DataFrame(

[

[None, None, 2.8, 25, None, 25, 0, 0, 0, None, 0, 30706.0, 7877.0],

[None, None, 2.75, 25, None, 25, 0, 0, 0, None, 0, 36421.0, 9176.0]

],

columns=features.columns

)

# completa la tabla con las nuevas características total\_area, bedrooms, living\_area y kitchen\_area

new\_features.loc[0, 'total\_area'] = 900.0

new\_features.loc[0, 'bedrooms'] = 12

new\_features.loc[0, 'living\_area'] = 409.7

new\_features.loc[0, 'kitchen\_area'] = 112# < escribe tu código aquí >

new\_features.loc[1, 'total\_area'] = 109.0

new\_features.loc[1, 'bedrooms'] = 2

new\_features.loc[1, 'living\_area'] = 32

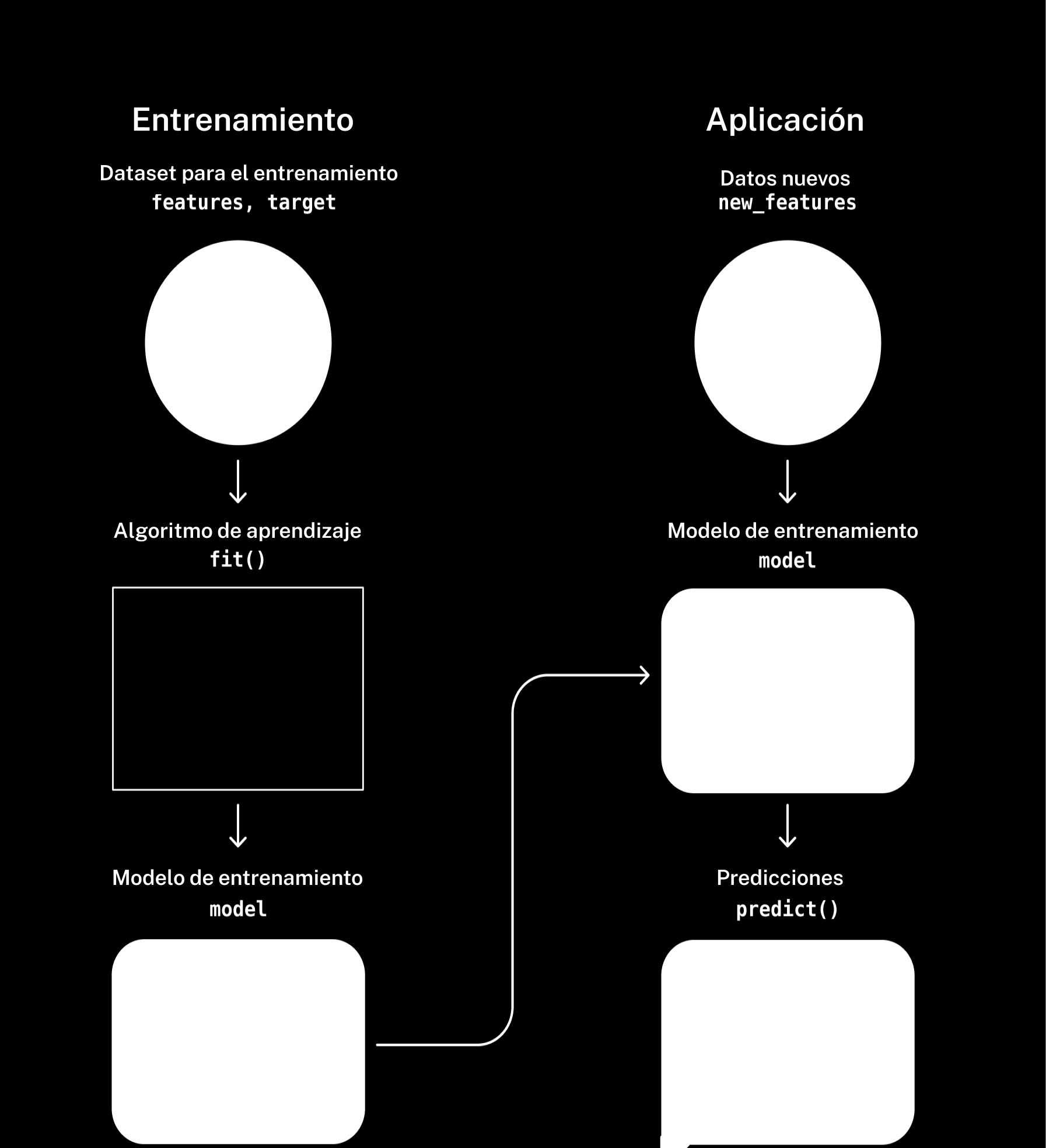
new\_features.loc[1, 'kitchen\_area'] = 40.5

answers = model.predict(new\_features) # predice respuestas y muestra el resultado en la pantalla

print(answers)# < escribe tu código aquí >  
[1. 0.]

# Diagrama de flujo de machine learning

Compara el diagrama de flujo de machine learning con el código. ¿Hay algo nuevo?



Esto es lo que tenemos:

* El conjunto de datos de entrenamiento se almacena en las variables features y target.
* Las características de las nuevas observaciones se registran en la variable new\_features. Ahora necesitamos encontrar el objetivo.
* El método fit() se usa para *entrenamiento*, y el método predict() se usa para *prueba.*
* El modelo se almacena en la variable model. Una vez que hayas terminado de entrenarlo, lo puedes utilizar para la predicción.

# Presentación del modelo

Averigüemos cómo se ve un modelo entrenado y qué hay dentro de él.

¿Habrías imaginado que un modelo contiene unos cuantos de miles de strings de código como estos?

|--- total\_area <= 60.75

| |--- total\_area <= 46.36

| | |--- city\_center\_dist <= 7959.50

| | | |--- kitchen\_area <= 8.10

| | | | |--- living\_area <= 21.45

| | | | | |--- living\_area <= 12.50

| | | | | | |--- kitchen\_area <= 7.50

| | | | | | | |--- class: 0.0

| | | | | | |--- kitchen\_area > 7.50

| | | | | | | |--- class: 1.0

| | | | | |--- living\_area > 12.50

| | | | | | |--- total\_area <= 33.75

...

Es demasiado complicado. Este programa fue escrito por otro programa, por ello, no debes esperar que sea legible para las personas.

Hay algoritmos dedicados que pueden ayudarnos a convertir modelos complicados (como este árbol de decisión) en algo más comprensible. Esto es lo que obtenemos:

|--- total\_area <= 60.75

| |--- total\_area <= 46.36

| | |--- class: 0.0

| |--- total\_area > 46.36

| | |--- ceiling\_height <= 2.69

| | | |--- class: 0.0

| | |--- ceiling\_height > 2.69

| | | |--- class: 1.0

|--- total\_area > 60.75

| |--- class: 1.0

En Python, el mismo texto se ve así:

if features['total\_area'] <= 60.75:

if features['total\_area'] <= 46.36:

answer = 0

else:

if features['ceiling\_height'] <= 2.69:

answer = 0

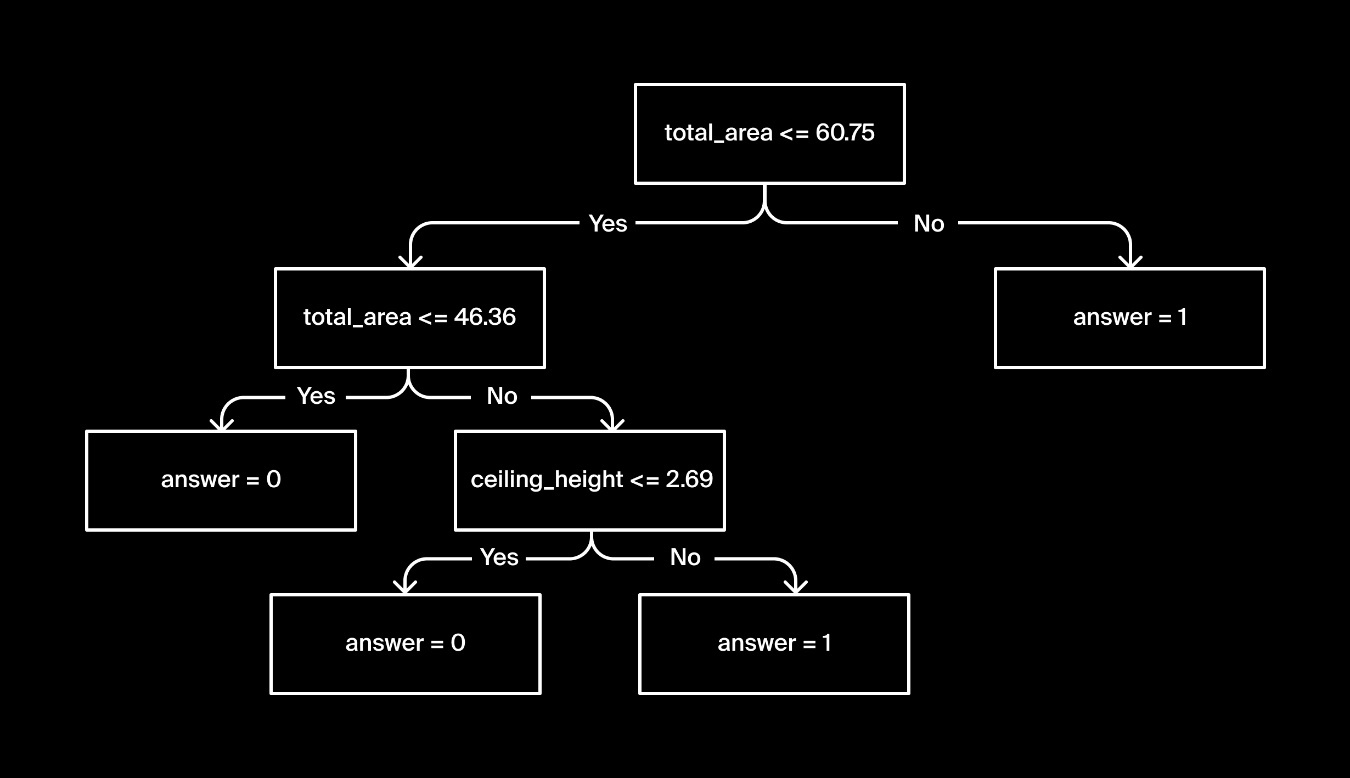
else:

answer = 1

else:

answer = 1

Así es como se ve como un gráfico:



¡Qué condiciones tan extrañas! Pero a la máquina no le importan los números bonitos.

En algunos casos, especialmente cuando tratamos con personas (por ejemplo, en diagnósticos clínicos o análisis de cuestionarios), la capacidad de mirar dentro del modelo y ver cómo aborda la tarea es crucial para prevenir escenarios no deseados, como la discriminación laboral. Sin embargo, para las tareas comerciales, la transparencia del modelo es opcional. Si funciona, no lo arregles.

# Conclusión

¡Felicidades por entrenar tu primer modelo!

En este capítulo aprendiste:

* Qué tareas puede resolver el machine learning.
* La diferencia entre tareas de regresión y clasificación.
* Qué son los modelos y algoritmos de aprendizaje.

A continuación, aprenderás cómo saber qué tan bien funciona un modelo y cómo usar las métricas.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Primer modelo entrenado](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Resumen_del_captulo_Primer_modelo_entrenado.pdf)
* [Hoja informativa: Primer modelo entrenado](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Hoja_informativa_Primer_modelo_entrenado.pdf)

Calidad del modelo

# Introducción

Has entrenado a tu primer modelo. Pero ¿funciona bien? Pongámoslo a prueba.

## Empezarás por:

* Aprender a trabajar con conjuntos de datos de prueba.
* Aprender acerca del sobreajuste y subajuste.

## ¿Cuánto tiempo tomará?

Diez lecciones de 5-10 minutos cada una

## Descripción del ejercicio

Seguiremos desarrollando un algoritmo para dar precios de apartamentos de forma automática.

Calidad del modelo

# Aleatoriedad en algoritmos de aprendizaje

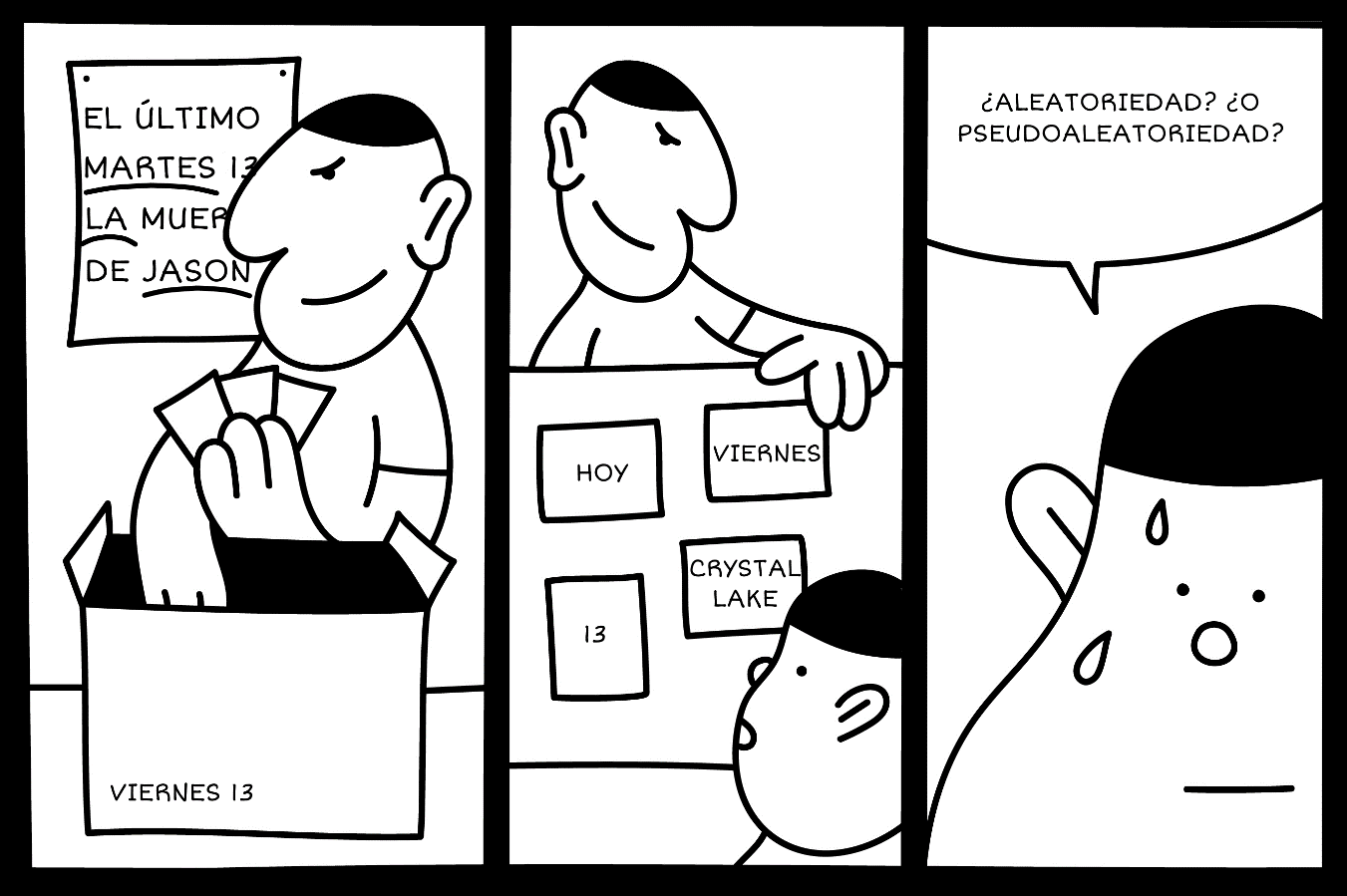
Obtenemos un nuevo modelo entrenado cada vez que entrenamos un árbol de decisión. ¿Cómo se supone que vamos a hacer una prueba si no sabemos quién la va a tomar? Analicemos este problema.

Agregar aleatoriedad a los algoritmos de aprendizaje es un gran impulso para la capacidad de un modelo para descubrir relaciones en los datos. Digamos que estás aprendiendo español. Tu plan es hacer veinte tarjetas con palabras nuevas y repasarlas de vez en cuando. Tu amigo sugiere barajar las tarjetas cada vez para memorizar mejor las palabras. El barajado introduce aleatoriedad en el algoritmo de aprendizaje.

Una computadora no puede generar números verdaderamente aleatorios. Utiliza generadores de números pseudoaleatorios que crean secuencias aparentemente aleatorias. Saber un cierto número no te permitirá adivinar el siguiente.

Tal vez te preguntes cómo es posible que los números sean aleatorios, pero siempre iguales. Nuevamente, es porque solo *parecen* aleatorios.

Volvamos a las tarjetas en español. Las tarjetas te ayudaron mucho, así que decidiste usarlas para ayudar a otros estudiantes de español. Crea una secuencia diferente de tarjetas para cada día para repartir a los estudiantes. De esta manera, sabrás exactamente qué tarjeta será, digamos, la octava en un viernes determinado. Esto no dañará al proceso de aprendizaje, ya que el orden de las tarjetas parecerá aleatorio para los estudiantes.



Para que el algoritmo de aprendizaje utilice siempre los mismos números pseudoaleatorios, simplemente especifica el parámetro random\_state:

*# especifica un estado aleatorio (número)*

model = DecisionTreeClassifier(random\_state=12345)

*# entrena al modelo de la misma manera que antes*

model.fit(features, target)

Puedes establecer cualquier valor para random\_state ("54321", "123", "0" o incluso un string). Lo importante es utilizar el mismo valor durante todo el tiempo que desees obtener exactamente el mismo modelo. Usemos el valor random\_state de "12345".

Si configuras random\_state=None (el valor predeterminado), la pseudoaleatoriedad siempre será diferente.

Pregunta

¿Por qué necesitamos el parámetro random\_state para algoritmos de aprendizaje?

A los programadores les gusta complicar las cosas.

Para obtener modelos ligeramente diferentes cada vez que ejecutamos el mismo código.

Este parámetro se especifica para duplicar un experimento exitoso

Perder resultados exitosos no es bueno.

¿Recuerdas haber jugado videojuegos en tu niñez? Luchar en un nivel difícil durante una semana, finalmente superarlo, pero olvidar guardar tu progreso... ¡Qué fastidio!

Si ves el código, puedes adivinar fácilmente qué modelo obtendrás.

¡Lo has entendido bien!

# Dataset de prueba

Ahora es momento poner el modelo a prueba. ¿Cómo comprobamos su conocimiento? Necesitaremos un nuevo conjunto de datos con respuestas conocidas.

Recordemos el código de entrenamiento del modelo:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features, target)

Para probar si nuestro modelo hace predicciones precisas incluso cuando se enfrenta a nuevos datos, vamos a utilizar un nuevo conjunto de datos. Ese será el conjunto de datos de prueba. Nombremos el archivo de datos test\_data\_us.csv y veamos cómo lo maneja el modelo.

Ejercicio 1  
  
  
Asigna las tres primeras observaciones del conjunto de prueba (/datasets/test\_data\_us.csv) a la variable test\_df. Guarda las funciones utilizadas para la clasificación en la variable test\_features. Haz una predicción de las respuestas.

Crea la nueva columna con las respuestas correctas (price\_class) y calcula sus valores de la misma manera que lo hiciste para el conjunto de datos principal. Copia esta columna en la variable test\_target.

Imprime las predicciones y las respuestas correctas en la pantalla de la siguiente manera:

Predicciones: [... ... ...]

Respuestas correctas: [... ... ...]

Las matrices de salida tienen que ser numpy.ndarray. El modelo genera este formato de manera predeterminada, así que deja test\_predictions como está. Usa el atributo values para tomar los valores de numpy.ndarray desde el objeto serie test\_target.

Averigüa cuántos errores ha cometido el modelo.

Procede como lo hiciste con el conjunto de entrenamiento. Carga los datos, convierte la regresión en clasificación, guarda las características y el objetivo en las variables test\_features y test\_target.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

# Cargar el conjunto de entrenamiento

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear la columna 'price\_class' en el conjunto de entrenamiento

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

# Definir las características y el objetivo del conjunto de entrenamiento

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

# Entrenar el modelo

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features, target)

# Cargar el conjunto de prueba

test\_df = pd.read\_csv('/datasets/test\_data\_us.csv')

# Asignar las tres primeras observaciones a test\_df

test\_df = test\_df.head(3)

# Crear la columna 'price\_class' en el conjunto de prueba

test\_df.loc[test\_df['last\_price'] > 113000, 'price\_class'] = 1

test\_df.loc[test\_df['last\_price'] <= 113000, 'price\_class'] = 0

# Definir las características y el objetivo del conjunto de prueba

test\_features = test\_df.drop(['last\_price', 'price\_class'], axis=1)

test\_target = test\_df['price\_class']

# Hacer predicciones en el conjunto de prueba

test\_predictions = model.predict(test\_features)

# Convertir test\_target a numpy.ndarray

test\_target\_values = test\_target.values

# Imprimir las predicciones y las respuestas correctas

print("Predicciones:", test\_predictions)

print("Respuestas correctas:", test\_target\_values)

# < escribe tu codigo aqui >

Respuesta ;

Predicciones: [1. 0. 0.]

Respuestas correctas: [0. 1. 0.]

¡Dos de las tres predicciones son incorrectas! Eso no nos servirá. Sin embargo, no podemos juzgar a nuestro modelo con base en solo estas. La muestra no es representativa. Necesitamos revisar todas las predicciones, y como hay cientos, esto debe automatizarse.

Ejercicio 2

Tres ejemplos no son suficientes para saber si el modelo funciona bien o no. Cuenta el número de errores para todo el conjunto de prueba.

Escribe la función error\_count(). El modelo toma las respuestas y predicciones correctas y devuelve el número de discrepancias. Muestra el resultado en pantalla de la siguiente manera (ya en el precódigo): Aquí está el algoritmo para la función error\_count():

— Declara una variable de contador y asígnale "0"

— Crea un bucle con una longitud igual al número de respuestas correctas:

Comprueba que cada respuesta coincida con cada predicción. Si no, agrega "1" al contador. Errores: ...

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features, target)

test\_df = pd.read\_csv('/datasets/test\_data\_us.csv')

test\_df.loc[test\_df['last\_price'] > 113000, 'price\_class'] = 1

test\_df.loc[test\_df['last\_price'] <= 113000, 'price\_class'] = 0

test\_features = test\_df.drop(['last\_price', 'price\_class'], axis=1)

test\_target = test\_df['price\_class']

test\_predictions = model.predict(test\_features)

def error\_count(answers, predictions):

count = 0

for answer, prediction in zip(answers, predictions):

if answer != prediction:

count += 1

return count# < función de código aquí >

print('Errores:', error\_count(test\_target, test\_predictions))

Errores: 275

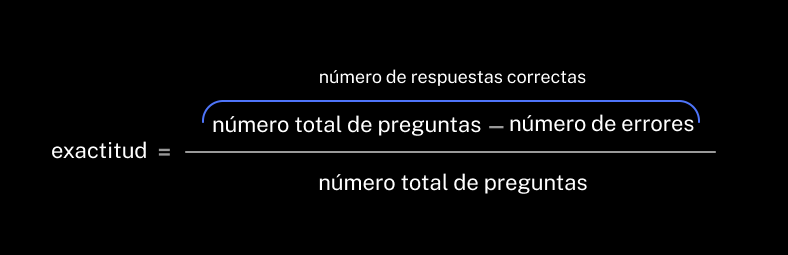
¡275 errores! ¿Es mucho? Comparemos la cantidad de errores con el tamaño del conjunto de prueba para ver si los errores representan un gran porcentaje.

# Exactitud

Nada es perfecto y todo es relativo. Incluso los mejores modelos cometen errores, pero el número de errores nos importa solo en relación con el número total de respuestas.

La relación entre el número de respuestas correctas y el número total de preguntas (es decir, el tamaño del conjunto de datos de prueba) se denomina exactitud (accuracy).

Para calcular la *exactitud*, utiliza esta fórmula:



Escribe la función accuracy(). Esta divide el número de respuestas correctas entre el número total de predicciones realizadas y devuelve la puntuación de exactitud.

Muestra la *exactitud* en la pantalla de la siguiente manera (ya en el precódigo): Exactitud: ...

Use la fórmula de exactitud dada en esta lección. Utiliza la función error\_count() para obtener el número de errores.

De forma alternativa, puedes reescribir la función error\_count() para contar las respuestas correctas. En ese caso, utiliza esta plantilla:

def accuracy(answers, predictions):

correct = 0

for i in range(len(answers)):

*# < escribe el código aquí >*

return correct / len(answers)

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 263000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 263000, 'price\_class'] = 0

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features, target)

test\_df = pd.read\_csv('/datasets/test\_data\_us.csv')

test\_df.loc[test\_df['last\_price'] > 263000, 'price\_class'] = 1

test\_df.loc[test\_df['last\_price'] <= 263000, 'price\_class'] = 0

test\_features = test\_df.drop(['last\_price', 'price\_class'], axis=1)

test\_target = test\_df['price\_class']

test\_predictions = model.predict(test\_features)

# Definir la función accuracy

def accuracy(answers, predictions):

correct = 0

for i in range(len(answers)):

if answers[i] == predictions[i]: # Contar las predicciones correctas

correct += 1

return correct / len(answers) # Calcular la exactitud

# Mostrar la exactitud

print('Exactitud:', accuracy(test\_target.values, test\_predictions))

Resultado

Exactitud: 0.8873456790123457

¡Es correcto!

Casi 9 de 10. ¡No está mal! Tu pequeño modelo es casi un estudiante de 10.

# Métricas de evaluación

¿Cuál es la diferencia entre un buen modelo y uno malo? ¿Qué métricas debes elegir para evaluar un modelo?

Las métricas de evaluación se utilizan para medir la calidad de un modelo y se pueden expresar numéricamente. Ya has encontrado una de estas métricas: *exactitud*. Hay otras, por ejemplo:

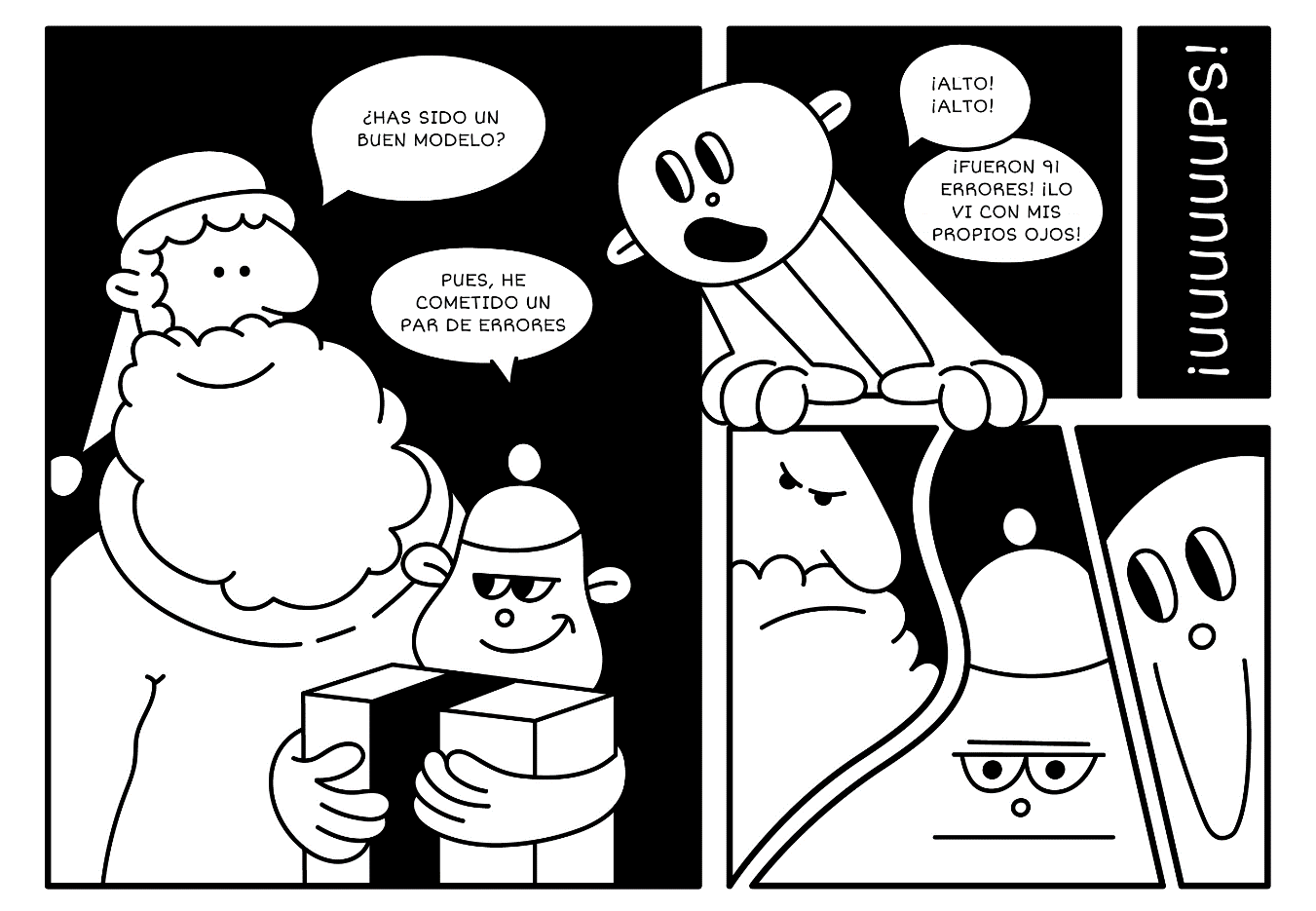
* precisión (precision) toma todos los apartamentos que el modelo consideró caros (están marcados como "1") y calcula qué fracción de ellos era costosa realmente. Los apartamentos no reconocidos por el modelo se ignoran.
* recall (sensibilidad) toma todos los apartamentos que son realmente caros y calcula qué fracción de ellos reconoció el modelo. Los apartamentos que fueron reconocidos por el modelo por error se ignoran.

Veamos el siguiente ejemplo para comprender mejor estas métricas:

Digamos que tienes un trabajo en un banco. El banco tiene 10 000 clientes. Sin embargo, 100 de ellos son estafadores, y tu trabajo es rastrearlos. El Clasificador A marca a 1000 clientes como posibles estafadores, pero solo 90 de ellos son estafadores reales. El Clasificador A tiene un recall alto (90%) porque logra encontrar a casi todos los estafadores, pero su precisión es baja (9%) ya que acusa erróneamente a cientos de ciudadanos honrados. Por otro lado, el Clasificador B marca a 10 clientes como estafadores, y 9 de ellos son realmente estafadores. Su recall es menor porque solo puede encontrar el 9% de ellos, pero la precisión es del 90% (¡9 de 10!).

En otras palabras, recall es una prioridad cuando es importante encontrar todas las observaciones requeridas, incluso a costa de cometer muchos errores, y la precisión es importante cuando tu objetivo es minimizar los errores, incluso si eso significa que muchos casos no se detectan.

Diferentes tareas requieren diferentes métricas de evaluación. ¿Por qué elegimos *exactitud* al determinar los precios de los apartamentos? Porque necesitamos tener en cuenta ambos tipos de errores (ceros que deberían ser unos, así como unos que deberían ser ceros), y eso es justo lo que hace la exactitud. Cada predicción incorrecta conduce a una recomendación incorrecta y a una posible pérdida de ganancias para el vendedor. A su vez, una mayor exactitud en la clasificación genera más ganancias.



La alta calidad del modelo es algo bueno, pero su aplicación debe justificarse. Establezcamos los límites obvios. Si todas las respuestas son incorrectas, la *exactitud* es cero. Si todas las respuestas son correctas, es igual a uno. Por lo tanto, la exactitud no puede ser menor que cero ni mayor que uno. Si nuestro modelo tuviera una exactitud de 0.4, ¿su calidad se consideraría alta o baja?

Pregunta

Supongamos que tenemos un modelo que asigna "0" o "1" al azar, con una probabilidad de 50/50. En otras palabras, el modelo "adivina" las respuestas en lugar de intentar hacer inferencias a partir de las características. ¿Cuál es la *exactitud* del modelo? Recordatorio: en esta tarea, el número de apartamentos en ambas clases es igual.

0

0.25

0.5

¡Bingo!

1

¡Bien hecho!

Consideremos las respuestas "1" (apartamentos caros) y "0" (apartamentos baratos) por separado:

# Métricas de evaluación en Scikit-Learn

*Sklearn* tiene muchas funciones para calcular métricas, por lo que no tenemos que usar una fórmula para encontrar la *exactitud*.

Las funciones métricas de la *librería sklearn* se encuentran en el módulo sklearn.metrics. Para calcular la exactitud utiliza la función accuracy\_score().

from sklearn.metrics import accuracy\_score

La función toma dos argumentos (las respuestas correctas y las predicciones del modelo) y devuelve el valor de *exactitud*.

accuracy = accuracy\_score(target, predictions)

exactitud=0.5⋅(𝑝𝑜𝑟𝑐𝑖𝑜ˊ𝑛 𝑑𝑒 𝑎𝑑𝑖𝑣𝑖𝑛𝑎𝑑𝑎𝑠 𝑐𝑜𝑟𝑟𝑒𝑐𝑡𝑎𝑚𝑒𝑛𝑡𝑒 1)+0.5⋅(𝑝𝑜𝑟𝑐𝑖𝑜ˊ𝑛 𝑑𝑒 𝑎𝑑𝑖𝑣𝑖𝑛𝑎𝑑𝑎𝑠 𝑐𝑜𝑟𝑟𝑒𝑐𝑡𝑎𝑚𝑒𝑛𝑡𝑒 0)exactitud=0.5⋅(*porcio*ˊ*n* *de* *adivinadas* *correctamente* 1)+0.5⋅(*porcio*ˊ*n* *de* *adivinadas* *correctamente* 0)

Las respuestas del modelo no están vinculadas con las respuestas correctas, por lo que la probabilidad de adivinar "1" es del 50 % (lo mismo para "0"), lo que significa que la *exactitud* es de 0.5.

exatitud=0.5⋅0.5+0.5⋅0.5=0.25+0.25=0.5exatitud=0.5⋅0.5+0.5⋅0.5=0.25+0.25=0.5

[Cálculo detallado](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/moved_Clculo_detallado_esp.pdf)

Si la *exactitud* es 0.4, ¿qué dice eso sobre la calidad del modelo? Es baja. Incluso adivinar al azar es mejor que eso.

Siempre asegúrate de que tu modelo funcione mejor que la casualidad, es decir, realiza una prueba de cordura.

## Pistas

La *exactitud* para el conjunto de entrenamiento se puede calcular de la siguiente manera:

train\_predictions = model.predict(features)

accuracy\_score(target, train\_predictions)

¿La puntuación de exactitud difiere entre el conjunto de entrenamiento y el conjunto de prueba? Calcula los valores y muéstralos en la pantalla de la siguiente manera:

Exactitud

Training set: ...

Test set: ...

Guarda las predicciones en las variables train\_predictions (para el conjunto de entrenamiento) y test\_predictions (para el conjunto de prueba).

La *exactitud* para el conjunto de entrenamiento se puede calcular de la siguiente manera:

train\_predictions = model.predict(features)

accuracy\_score(target, train\_predictions)

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Cargar el conjunto de datos de entrenamiento

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear la columna de clasificación del precio

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

# Separar las características y el objetivo

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

# Entrenar el modelo

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features, target)

# Realizar predicciones en el conjunto de entrenamiento

train\_predictions = model.predict(features)

train\_accuracy = accuracy\_score(target, train\_predictions)

# Cargar el conjunto de datos de prueba

test\_df = pd.read\_csv('/datasets/test\_data\_us.csv')

# Crear la columna de clasificación del precio en el conjunto de prueba

test\_df.loc[test\_df['last\_price'] > 113000, 'price\_class'] = 1

test\_df.loc[test\_df['last\_price'] <= 113000, 'price\_class'] = 0

# Separar las características y el objetivo en el conjunto de prueba

test\_features = test\_df.drop(['last\_price', 'price\_class'], axis=1)

test\_target = test\_df['price\_class']

# Realizar predicciones en el conjunto de prueba

test\_predictions = model.predict(test\_features)

test\_accuracy = accuracy\_score(test\_target, test\_predictions)

# Mostrar las exactitudes

print('Exactitud')

print(f'Training set: {train\_accuracy}')

print(f'Test set: {test\_accuracy}')

Resultado

Exactitud

Training set: 0.9998460354118552

Test set: 0.5756172839506173

Es correcto!

¡Mira eso! ¡Los resultados son diferentes! El modelo es más exacto cuando se trabaja con el conjunto de entrenamiento. El conjunto de prueba todavía es demasiado para él. ¡Pero no te preocupes! Mejoraremos el rendimiento de nuestro modelo.

Sprint 8

Capítulo 3/7

Calidad del modelo

# Subajuste y sobreajuste

Descubrimos que, en cuanto a la *exactitud*, nuestro modelo funciona mejor con el conjunto de entrenamiento, en comparación con el conjunto de prueba. Este suele ser el caso del machine learning. Pero, ¿por qué?

El modelo no tuvo problemas con las observaciones del conjunto de entrenamiento, pero sí con el conjunto de prueba. Este es un síntoma de sobreajuste.

Es posible que hayas enfrentado un problema similar. El sobreajuste es como estudiar demasiado para un examen. Puedes memorizar libros de texto completos, palabra por palabra, pero si te enfocas demasiado en el material del libro, el conocimiento será inútil en una prueba donde se te presenten preguntas que nunca antes habías visto. No serás capaz de responderlas si no puedes ver el contexto completo, de la misma forma que un modelo no podrá trabajar correctamente en el problema general si entrena demasiado para categorizar perfectamente un conjunto de datos particular.

Entrena demasiado un modelo y detectará las fluctuaciones y desviaciones más pequeñas que provienen del ruido aleatorio, en lugar de las relaciones de causa y efecto realmente existentes. Un modelo sobreajustado comenzará a ver dependencias que en realidad no existen. Intentará aplicar estas dependencias inexistentes a nuevos datos que podrían clasificarse mediante una regla más simple y general.

El subajuste es lo contrario. Ocurre cuando la *exactitud* es baja y aproximadamente igual tanto para el conjunto de entrenamiento como para el de prueba. Eso es como no tener suficiente tiempo para leer todo el material de estudio, y mucho menos para memorizar las respuestas. ¿Te suena familiar?

Pregunta

Elige el enunciado correcto:

La alta calidad del modelo en el conjunto de entrenamiento indica sobreajuste.

La baja calidad del modelo en el conjunto de prueba indica sobreajuste.

El sobreajuste significa que el modelo tiene una comprensión deficiente de las dependencias de datos.

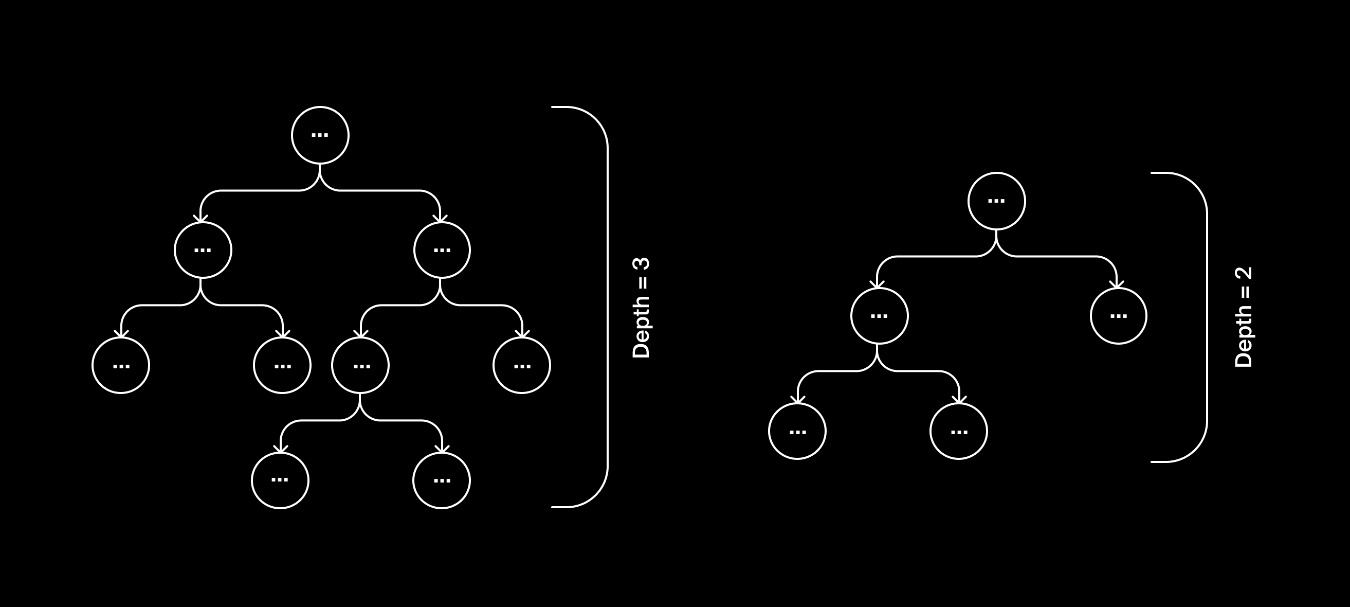
Correcto. Al igual que quemarse las pestañas para un examen: el modelo memoriza todas las respuestas correctas, pero no logra captar la idea general.

Si tu modelo funciona sin problema con nuevos datos, está sobreajustado.

¡Perfecto!

No siempre es posible evitar el sobreajuste o el subajuste. Cuando mejoras en uno de estos, aumenta el riesgo del otro.

Revisa este ejemplo de excelente ajuste de un algoritmo de entrenamiento. ¿Cómo afecta al equilibrio entre el sobreajuste y el subajuste? La profundidad del árbol (altura) es la cantidad máxima de condiciones desde la "parte superior" del árbol hasta la respuesta final, según la cantidad de transiciones de nodo a nodo.



Pregunta

¿Cómo afecta la altura (profundidad) del árbol al sobreajuste o subajuste?

No afecta a ninguno de los dos.

Una profundidad de árbol alta significa que el modelo tiende a sobreajustarse. Un árbol corto puede conducir a un subajuste.

Los árboles altos y antiguos han visto muchos veranos e inviernos, por lo que son mucho más grandes que los árboles jóvenes o los tocones (por cierto, un tocón es un árbol de decisión con una profundidad de uno). Así como los árboles reales son más pesados si son más altos, los árboles de decisión tienen más información si son más profundos.

Cuanta más información contenga un árbol, mayor será la posibilidad de que esté subajustado. Los árboles con menos información aumentan la posibilidad de sobreajuste.

¡Bien hecho!

Puedes establecer la profundidad de un árbol en *sklearn* con el parámetro max\_depth:

*# especificar la profundidad (ilimitado por defecto)*

model = DecisionTreeClassifier(random\_state=12345, max\_depth=3)

model.fit(features, target)

Lo intentaremos en la siguiente lección.

# Experimentos con árboles de decisión

### ¿Cómo afecta exactamente la profundidad del árbol a la exactitud de la predicción? Para averiguarlo, vamos a recorrer diferentes niveles de profundidad del árbol en el algoritmo de entrenamiento.

Ahora intentaremos determinar a qué nivel de profundidad del árbol el sobreajuste empieza a afectar a la *exactitud* de nuestros árboles. Vamos a aumentar la profundidad máxima del árbol en uno para cada nuevo modelo, comparar los valores de *exactitud* en los conjuntos de entrenamiento y prueba, y ver dónde empiezan a divergir.

Haz que el programa de entrenamiento del árbol de decisión pruebe varias configuraciones del parámetro de profundidad máxima del árbol max\_depth. El programa tiene que:

* Iterar sobre los valores del 1 al 10.
* Entrenar modelos en el conjunto de entrenamiento. No olvides especificar random\_state=54321 al inicializar el constructor del modelo.
* Imprimir la puntuación de *exactitud* de cada modelo en los conjuntos de entrenamiento y de prueba.

## Pistas

Termina el código del bucle:

for depth in range(1, 11):

model = *# <crea un modelo, especifica random\_state=54321 y max\_depth=depth >*

*# < entrena el modelo >*

train\_predictions = *# < predicciones obtenidas con el conjunto de entrenamiento >*

test\_predictions = *# < predicciones obtenidas con el conjunto de prueba>*

print("Exactitud de max\_depth igual a", depth)

print("Conjunto de entrenamiento:", accuracy\_score(target, train\_predictions))

print("Conjunto de prueba:", accuracy\_score(test\_target, test\_predictions))

print()

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Cargar los datos de entrenamiento

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear la columna 'price\_class' basada en 'last\_price'

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

# Separar características y objetivo

features = df.drop(['last\_price', 'price\_class'], axis=1)

target = df['price\_class']

# Cargar los datos de prueba

test\_df = pd.read\_csv('/datasets/test\_data\_us.csv')

# Crear la columna 'price\_class' basada en 'last\_price' en el conjunto de prueba

test\_df.loc[test\_df['last\_price'] > 113000, 'price\_class'] = 1

test\_df.loc[test\_df['last\_price'] <= 113000, 'price\_class'] = 0

# Separar características y objetivo en el conjunto de prueba

test\_features = test\_df.drop(['last\_price', 'price\_class'], axis=1)

test\_target = test\_df['price\_class']

# Iterar sobre los valores del parámetro max\_depth del 1 al 10

for depth in range(1, 11):

# Crear el modelo con la profundidad máxima especificada y random\_state=54321

model = DecisionTreeClassifier(max\_depth=depth, random\_state=54321)

# Entrenar el modelo

model.fit(features, target)

# Obtener las predicciones para el conjunto de entrenamiento

train\_predictions = model.predict(features)

# Obtener las predicciones para el conjunto de prueba

test\_predictions = model.predict(test\_features)

# Calcular las puntuaciones de exactitud

train\_accuracy = accuracy\_score(target, train\_predictions)

test\_accuracy = accuracy\_score(test\_target, test\_predictions)

# Imprimir las puntuaciones de exactitud

print("Exactitud de max\_depth igual a", depth)

print("Conjunto de entrenamiento:", train\_accuracy)

print("Conjunto de prueba:", test\_accuracy)

print()

Resultado

/usr/local/lib/python3.9/site-packages/joblib/\_multiprocessing\_helpers.py:46: UserWarning: [Errno 2] No such file or directory. joblib will operate in serial mode

warnings.warn('%s. joblib will operate in serial mode' % (e,))

Exactitud de max\_depth igual a 1

Conjunto de entrenamiento: 0.8435719784449577

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 2

Conjunto de entrenamiento: 0.8435719784449577

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 3

Conjunto de entrenamiento: 0.8665127020785219

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 4

Conjunto de entrenamiento: 0.8739030023094688

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 5

Conjunto de entrenamiento: 0.891301000769823

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 6

Conjunto de entrenamiento: 0.9023864511162433

Conjunto de prueba: 0.5092592592592593

Exactitud de max\_depth igual a 7

Conjunto de entrenamiento: 0.9140877598152425

Conjunto de prueba: 0.37808641975308643

Exactitud de max\_depth igual a 8

Conjunto de entrenamiento: 0.930715935334873

Conjunto de prueba: 0.3950617283950617

Exactitud de max\_depth igual a 9

Conjunto de entrenamiento: 0.9441108545034642

Conjunto de prueba: 0.4367283950617284

Exactitud de max\_depth igual a 10

Conjunto de entrenamiento: 0.957197844495766

Conjunto de prueba: 0.3950617283950617

¡Es correcto!

Como puedes ver, a medida que aumenta la profundidad máxima del árbol, también lo hace la *exactitud* en el conjunto de entrenamiento. Sin embargo, la *exactitud* en el conjunto de prueba deja de mejorar después de que la profundidad de max\_depth alcance 5, ¡y de hecho empeora a medida que aumenta la profundidad! El modelo con max\_depth igual a 5 es también el último modelo en el que las puntuaciones de "exactitud" en ambos conjuntos son similares, por lo que podemos concluir que los modelos más profundos se ven afectados por el sobreajuste. Por eso no podemos confiar únicamente en el conjunto de entrenamiento a la hora de evaluar la calidad de nuestros modelos.

Sprint 8

Capítulo 3/7 · Última lección

Calidad del modelo

# Conclusión

En este capítulo aprendiste:

* Por qué necesitamos conjuntos de prueba
* Qué es el sobreajuste y cómo la profundidad del árbol lo afecta
* Cómo evaluar la calidad del modelo

En el próximo capítulo exploraremos técnicas para mejorar la calidad del modelo. Aprenderemos a medir la calidad sin un conjunto de prueba, además de usar hiperparámetros y nuevos algoritmos para elegir el mejor modelo.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Calidad del modelo](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Resumen_del_captulo_Calidad_del_modelo.pdf)
* [Hoja informativa: Calidad del modelo](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Hoja_informativa_Calidad_del_modelo_1.pdf)

Sprint 8

Capítulo 4/7

Mejora del modelo

# Introducción

Ya sabes cómo evaluar la calidad del modelo. Es hora de aprender a ajustar los algoritmos para mejorarlo.

## Empezarás por:

* Compensar el sobreajuste de un modelo dividiendo los datos en conjuntos de entrenamiento y validación.
* Cambiar los hiperparámetros del modelo.
* Entrenar modelos de regresión de bosque aleatorio y logística.
* Comparar la calidad de los modelos resultantes para escoger el más adecuado.

## ¿Cuánto tiempo tomará?

9 lecciones de 5-10 minutos cada una

## Descripción del ejercicio

Seguiremos desarrollando un algoritmo para dar precios de apartamentos de forma automática.

Sprint 8

Capítulo 4/7

Mejora del modelo

# Datasets de validación

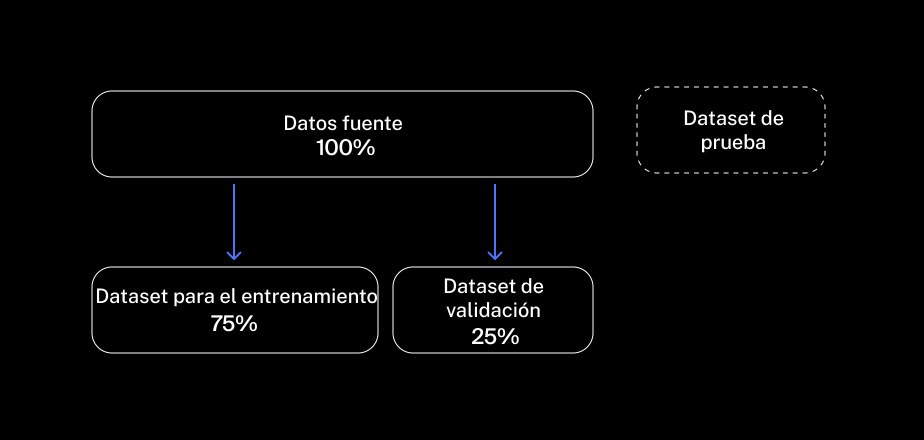
Necesitamos evaluar la precisión de predicciones del modelo pero ¿el conjunto de prueba está escondido?

*Imagina que estás usando pruebas de años anteriores para estudiar para un examen final. No te apresures en resolver todo de una vez. No quieres leer todas las respuestas porque quieres resolver algunas de las pruebas por tu cuenta para comprobar qué tan bien entiendes el tema. La misma estrategia se usa en el aprendizaje automático. Para que el control de calidad sea fiable necesitamos un dataset de validación.*

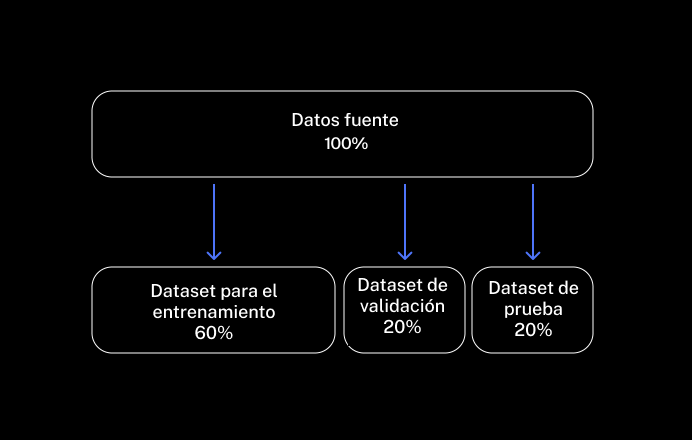
El dataset de validación se separa del dataset fuente antes de que se entrene el modelo. De otro modo, el modelo sabría todas las respuestas antes de aprender del conjunto de entrenamiento. La validación muestra cómo se comporta el modelo en el campo y ayuda a revelar si hay sobreajuste.

La parte de los datos que se va a asignar al conjunto de validación depende del número de observaciones y características, así como de la variación de los datos. Estos son los dos escenarios más comunes:

1) El conjunto de prueba existe (o existirá en el futuro cercano) pero no está disponible por el momento. La proporción ideal es 3:1. Esto significa que un 75 % es para el conjunto de entrenamiento y un 25 % es para el conjunto de validación. Este escenario se usará en nuestra plataforma de entrenamiento en línea.



2) El conjunto de prueba no existe. En ese caso los datos fuente deben dividirse en tres partes: entrenamiento, validación y prueba. Usualmente, el tamaño del conjunto de validación y del de prueba son iguales. Esto da como resultado una proporción de datos fuente de 3:1:1



Pregunta

Elige el enunciado correcto:

La validación ayuda a identificar modelos sobreajustados.

La validación muestra cómo lidia el modelo con datos desconocidos. Si la exactitud de la predicción cae dramáticamente, entonces el modelo está sobreajustado.

El modelo solo se valida una vez.

La validación te ayudará a identificar errores del código que podrían ocurrir durante el proceso de entrenamiento.

La validación es imposible sin un validador.

¡Buen trabajo!

Pregunta

Elige el enunciado correcto:

El conjunto de validación comprende hasta el 25 % del conjunto de prueba.

El conjunto de validación se usa para entrenar el modelo.

Se necesita el conjunto de validación para la evaluación final del modelo.

Se usa el conjunto de prueba para la evaluación final del modelo entrenado.

¡Así es!Mientras el modelo aprende, el conjunto de prueba espera, espera y espera... ¡Y, por fin, llega su momento de brillar en la evaluación final!

¡Perfecto!

# Separar datos en dos conjuntos

El conjunto de validación constituye el 25 % de los datos fuente. Entonces, ¿cómo debemos extraerlo?

En *sklearn* hay una función llamada train\_test\_split, con la que se puede separar cualquier conjunto de datos en dos: entrenamiento y prueba. Pero nosotros vamos a usar esta función para obtener un conjunto de validación y uno de entrenamiento.

Importa train\_test\_split desde el módulo model\_selection de scikit-learn:

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

Antes de separar, necesitamos establecer dos parámetros:

* Nombre del dataset que vamos a separar.
* Tamaño del conjunto de validación (test\_size). El tamaño se expresa con un decimal entre 0 y 1 que representa una fracción del dataset fuente. En este caso, tenemos test\_size=0.25 porque queremos trabajar con el 25 % del conjunto fuente.

La función train\_test\_split() devuelve dos conjuntos de datos: entrenamiento y validación.

df\_train, df\_valid = train\_test\_split(df, test\_size=0.25, random\_state=12345)

Nota: podemos asignar cualquier valor a random\_state excepto None.

Separa el dataset en dos conjuntos:

* conjunto de entrenamiento (df\_train);
* conjunto de validación (df\_valid) — 25 % de los datos fuente

Declara cuatro variables y pásalas de la siguiente manera:

* características: features\_train y features\_valid;
* objetivo: target\_train y target\_valid

Imprime los tamaños de las tablas que están almacenadas en cuatro variables (hecho en el precódigo).

## Pistas

Declara variables para el conjunto de validación así:

features\_valid = df\_valid.drop(['last\_price', 'price\_class'], axis=1)

target\_valid = df\_valid['price\_class']

import pandas as pd

from sklearn.model\_selection import train\_test\_split# < importa la función train\_test\_split desde la librería sklearn >

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

df\_train, df\_valid = train\_test\_split(df, test\_size=0.25, random\_state=12345)

# < separa los datos en entrenamiento y validación >

# < declara variables para las características y para la característica objetivo >

features\_train = df\_train.drop(['last\_price', 'price\_class'], axis=1)

target\_train = df\_train['price\_class']

features\_valid = df\_valid.drop(['last\_price', 'price\_class'], axis=1)# < escribe tu código aquí >

target\_valid = df\_valid['price\_class']# < escribe el código aquí

print(features\_train.shape)

print(target\_train.shape)

print(features\_valid.shape)

print(target\_valid.shape)

Resultado

(4871, 13)

(4871,)

(1624, 13)

(1624,)

¡Es correcto!

Puedes dividir muchas cosas: un partido político, una pantalla de televisión para un juego multijugador o, lo que más nos importa ahora, un dataset. Se requiere el enfoque correcto para separar adecuadamente. En este caso, necesitamos la función train\_test\_split()

Capítulo 4/7

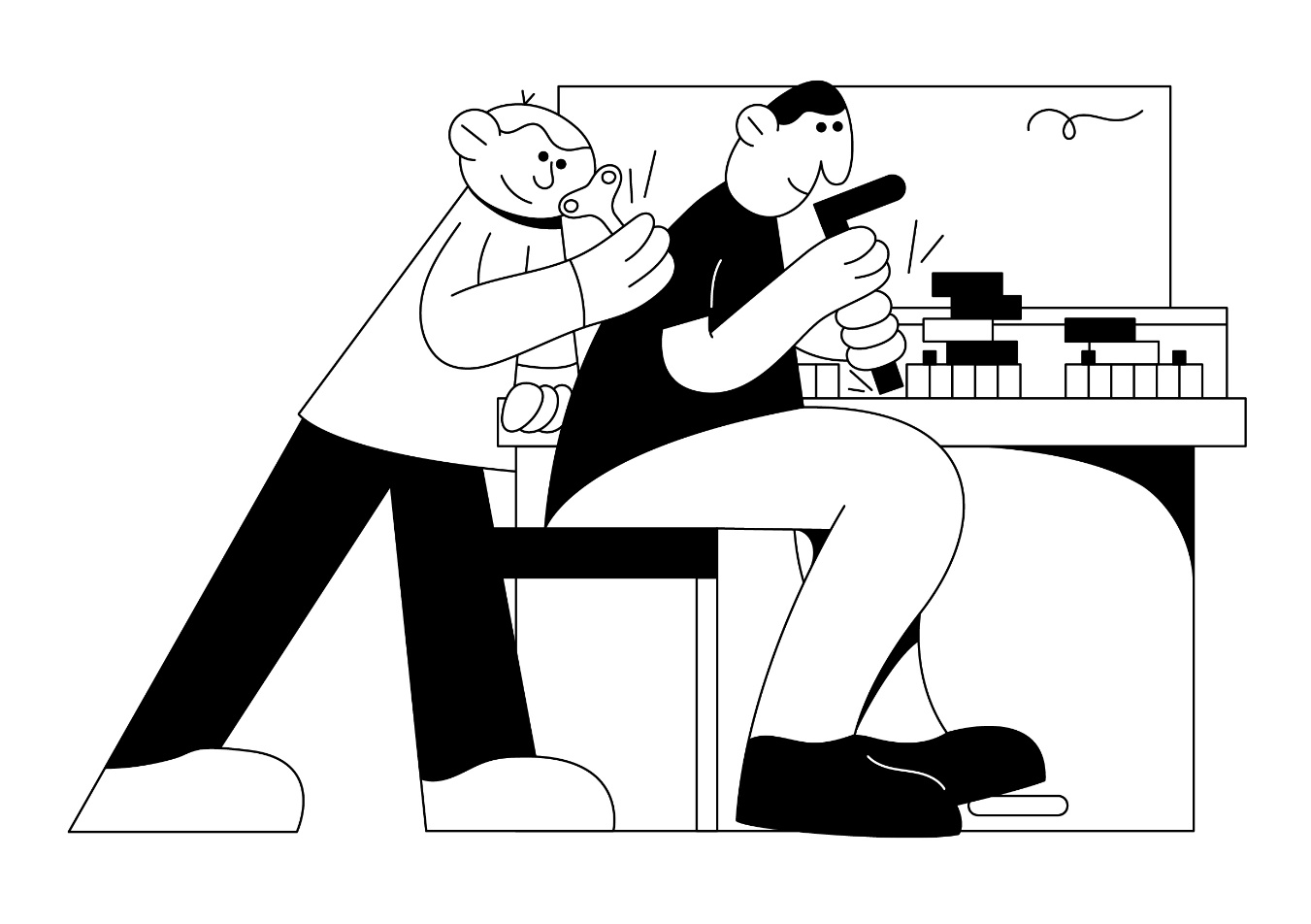
Mejora del modelo

# Hiperparámetros

¿Qué son los hiperparámetros y cómo pueden ayudarnos a escoger el mejor modelo? ¿En qué se diferencian los hiperparámetros y los parámetros regulares?

¿De qué depende la exactitud de las predicciones del árbol de decisión? Del número y de las posiciones relativas de los nodos (estructura), de la pregunta en la parte superior y de las respuestas en los nodos inferiores. El modelo adopta todos estos parámetros del conjunto de entrenamiento (no confundir con parámetros en Python).

Además de los parámetros de modelo regulares, tenemos hiperparámetros. Estos son configuraciones para algoritmos de aprendizaje. Por ejemplo, en el árbol de decisión, uno de los ejemplos es el parámetro de profundidad máxima. Otro son las opciones de criterio: Gini/entropía (más tarde veremos más a fondo el criterio de entropía). Los hiperparámetros también ayudan a mejorar el modelo. Se pueden ajustar antes del entrenamiento.



Pregunta

Selecciona los enunciados correctos:

Elige tantas como quieras

Los parámetros de un modelo determinan la manera en que un modelo hace predicciones.

Los parámetros de un modelo son configuraciones que determinan la forma en que funciona un modelo.

Los valores de los hiperparámetros dependen del dataset utilizado durante el entrenamiento.

Los parámetros de un modelo se configuran manualmente.

Los datos de entrenamiento afectan a los valores de los parámetros de un modelo.

Los parámetros de un modelo están vinculados a los datos porque se generan durante el entrenamiento.

¡Perfecto!

Echa un vistazo a un código ya conocido:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,

splitter='best')

Nuestra "caja negra" ya no es tan negra. Todos los hiperparámetros del algoritmo están dentro de los paréntesis. Ya conoces los criterios de profundidad máxima y Gini. También usaremos:

— min\_samples\_split: Este hiperparámetro determina el número mínimo de observaciones que se deben tener en un nodo antes de que pueda dividirse. Su objetivo es evitar la creación de nodos que contengan un número insuficiente de observaciones del conjunto de entrenamiento.

— min\_samples\_leaf: Este hiperparámetro establece el número mínimo de observaciones que debe tener una hoja, es decir, un nodo final sin divisiones. Su propósito es prevenir que el algoritmo genere nodos hoja con un número insuficiente de observaciones del conjunto de entrenamiento.

Los valores de los hiperparámetros se establecen de forma predeterminada. No necesitas especificarlos pero puedes cambiarlos. Imagina que quieres preparar la receta de macarrones con queso de la abuela. Todos los ingredientes y las porciones están en una lista pero quieres sorprender a la abuela y hacerlos más cremosos, así que añades más queso. No sabes si a tu familia le gustará pero tienes la certeza de que has cambiado los hiperparámetros del plato.

Para cada algoritmo, hay un conjunto específico de configuraciones que necesitas especificar antes del entrenamiento.

Pregunta

¿Qué significa "ajustar los hiperparámetros"? Elige el enunciado correcto:

Adaptar las configuraciones del modelo entrenado, metiéndote en el modelo y modificando los resultados.

Cambiar los datos de la caja negra antes del entrenamiento.

Especificar manualmente las características del modelo antes del entrenamiento.

¡Buen trabajo! ¡Qué difícil es engañarte!

¡Buen trabajo!

Pregunta

¿Cuáles de estos son parámetros de un modelo? Elige las respuestas correctas:

Elige tantas como quieras

La profundidad del árbol de decisión

Tu conocimiento de la naturaleza del árbol de decisión es bastante profundo. Es una parte de la estructura del árbol entrenado.

La profundidad máxima del árbol

El número mínimo de observaciones necesarias para hacer un nodo

La característica que se usa en la condición del primer nodo.

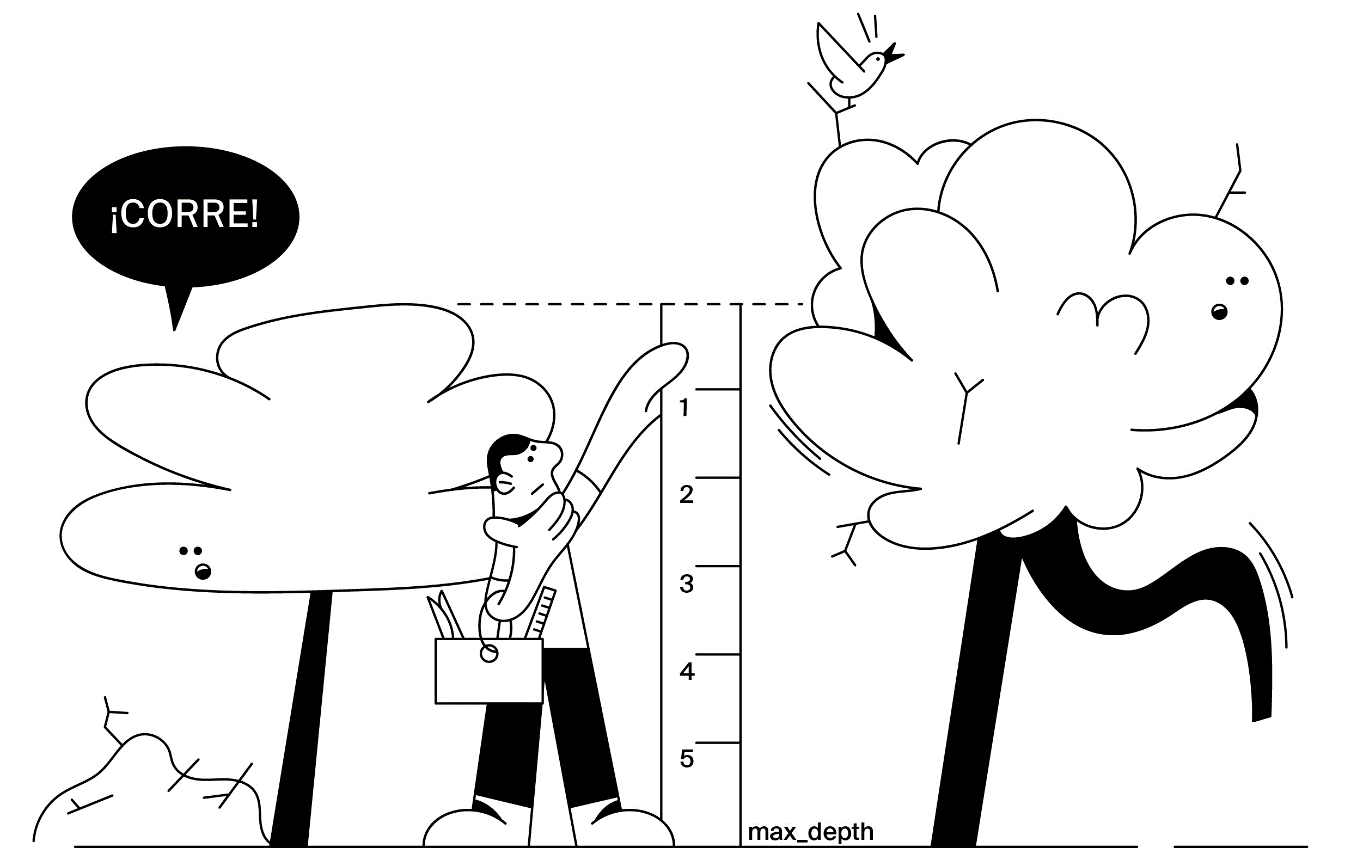
Esta es parte de un árbol entrenado.

¡Perfecto!

# Ajuste de hiperparámetros

Vamos a ajustar los hiperparámetros de nuestro árbol.

El hiperparámetro más importante de un árbol de decisión es max\_depth. Este determina qué obtendremos al final: un tocón con una pregunta o un arce con una enorme copa.



¿Cómo podemos encontrar el mejor valor para el hiperparámetro max\_depthsi queremos mejorar el modelo? No lo sabemos de antemano. Así que iteraremos diferentes valores con un bucle y compararemos la calidad de las diferentes versiones del modelo. Lo comprobaremos automáticamente sin el conjunto de prueba.

Cambia el hiperparámetro max\_depthen el bucle de 1 a 5. Por cada valor, imprime la calidad para el conjunto de validación. Imprime esto en la pantalla:

max\_depth = 1 : ...

max\_depth = 2 : ...

...

max\_depth = 5 : ...

Aún no necesitamos probar nuestros modelos con el dataset de prueba. Primero seleccionaremos el mejor modelo.

## Pistas

Termina el código del bucle:

for depth in range(1, 6):

model = *# < crea un modelo, especificar max\_depth=depth >*

*# < entrena el modelo >*

predictions\_valid = *# < encuentra las predicciones usando el conjunto de validación >*

print("max\_depth =", depth, ": ", end='')

print(accuracy\_score(target\_valid, predictions\_valid))

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

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

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

df\_train, df\_valid = train\_test\_split(df, test\_size=0.25, random\_state=12345)

features\_train = df\_train.drop(['last\_price', 'price\_class'], axis=1)

target\_train = df\_train['price\_class']

features\_valid = df\_valid.drop(['last\_price', 'price\_class'], axis=1)

target\_valid = df\_valid['price\_class']

# < crea un bucle para max\_depth de 1 a 5 >

# Crear un bucle para max\_depth de 1 a 5

for depth in range(1, 6):

# Crear el modelo, especificando max\_depth y random\_state

model = DecisionTreeClassifier(random\_state=12345, max\_depth=depth)

# Entrenar el modelo

model.fit(features\_train, target\_train)

# Encontrar las predicciones usando el conjunto de validación

predictions\_valid = model.predict(features\_valid)

# Imprimir la profundidad y la exactitud

print("max\_depth =", depth, ":", accuracy\_score(target\_valid, predictions\_valid))

Resultado

max\_depth = 1 : 0.8522167487684729

max\_depth = 2 : 0.8522167487684729

max\_depth = 3 : 0.8466748768472906

max\_depth = 4 : 0.8725369458128078

max\_depth = 5 : 0.8663793103448276

¡Es correcto!

Hay tres cosas que podrías ver por siempre: una chimenea ardiendo, agua caer e hiperparámetros cambiar la calidad del modelo.

Sin embargo, si el modelo no ha mejorado puede que sea buena idea dejarlo e intentar con otro.

# Nuevos modelos: bosque aleatorio

Hemos cambiado los hiperparámetros pero los resultados aún dejan mucho que desear. Claramente, un árbol no es suficiente. ¡Necesitamos un bosque!

Probemos con un nuevo algoritmo de aprendizaje llamado bosque aleatorio. Este algoritmo entrena una gran cantidad de árboles independientes y toma una decisión mediante el voto. Un bosque aleatorio ayuda a mejorar los resultados y a evitar el sobreajuste.

Sabes por qué la gente vota cuando hay que tomar decisiones importantes, ¿verdad? De esta forma puedes obtener una valoración promedio que anule el sesgo personal y los errores. El bosque aleatorio se basa en el mismo principio.

Entonces, ¿cómo lo entrenamos? En la librería scikit-learn, puedes encontrar RandomForestClassifier que es un algoritmo de bosque aleatorio. Impórtalo desde el módulo ensemble:

from sklearn.ensemble import RandomForestClassifier

Usaremos el hiperparámetro n\_estimators (significa "número de estimadores") para establecer el número de árboles en el bosque. El aumento en la cantidad de estimadores siempre disminuye la varianza de la predicción, por lo que cuantos más árboles uses, mejores resultados obtendrás. Los bosques no pueden sobreajustarse debido a que tienen demasiados árboles. Si bien el sobreajuste de un bosque aún puede ocurrir debido al sobreajuste de sus árboles individuales, este efecto generalmente se ve compensado por el beneficio de tener muchos árboles. En casos raros donde no lo es, la poda lo arregla, pero en la mayoría de los casos los beneficios de la poda son insignificantes.

Aunque el número de estimadores nunca provoca un sobreajuste, sigue siendo necesario limitarlo, aunque por una razón diferente. El uso de más y más árboles incurre en un costo computacional cada vez mayor y sufre de rendimientos decrecientes. Eventualmente, la métrica de calidad del modelo alcanza una meseta y deja de mejorar, mientras que el tiempo de ejecución sigue aumentando.

Scikit-learn establece n\_estimators en 100 de forma predeterminada. Pero por ahora, vamos a establecer el valor de n\_estimators en 3. Y no olvides hacer que la pseudoaleatoriedad sea estática con el parámetro random\_state.

model = RandomForestClassifier(random\_state=54321, n\_estimators=3)

Como en las lecciones anteriores, vamos a entrenar el modelo usando el método fit().

model.fit(features, target)

Hasta este punto usamos la función accuracy\_score() para comparar las etiquetas predichas con las respuestas reales y cuantificar las discordancias. Sin embargo, si todo lo que queremos hacer es evaluar la calidad del modelo, y no nos importan las etiquetas predichas en sí mismas, en lugar de usar el método predict() con la función accuracy\_score(), podemos usar un método que llama a ambos de manera interna: el método score(). De esta manera, el paso intermedio de convertir características en predicciones está oculto para nosotros y, en cambio, obtenemos la puntuación de *exactitud* de inmediato. Usarlo hace que el código sea más claro y más corto. Así es como se le llama:

model.score(features, target)

Completa el precódigo para hacer un bucle que pruebe modelos de bosque aleatorio con varios números de estimadores (árboles).

1. Divide los datos en conjuntos de entrenamiento y validación.
2. Elige el rango para que sea lo suficientemente grande para obtener una puntuación lo suficientemente buena, pero lo suficientemente pequeño para que tu programa no sea innecesariamente lento.
3. Configura el número de árboles para que sea igual a la variable de bucle est en el constructor del modelo.
4. Entrena modelos en el conjunto de entrenamiento. Calcula *accuracy* en el conjunto de validación para cada modelo.
5. Imprime la mejor puntuación de *accuracy* junto con el número correspondiente de estimadores.
6. Divide el 25 % de los datos para hacer el conjunto de validación.
7. Será suficiente con 10 estimadores, así que escribe range(1, 11).
8. Pasa n\_estimators=est al constructor del modelo.
9. Entrena modelos en el conjunto de entrenamiento: features\_train, target\_train. Calcula los valores de *accuracy* para el conjunto de validación, features\_valid, target\_valid.
10. Cada vez que encuentres el mejor score, (valor), actualiza best\_score y guarda el est actual en best\_est.

Consejo

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

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

# Cargar el dataset

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear la clase de precios

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

# Dividir el dataset en entrenamiento y validación (25% para validación)

df\_train, df\_valid = train\_test\_split(df, test\_size=0.25, random\_state=54321)

# Separar características y objetivo para el conjunto de entrenamiento

features\_train = df\_train.drop(['last\_price', 'price\_class'], axis=1)

target\_train = df\_train['price\_class']

# Separar características y objetivo para el conjunto de validación

features\_valid = df\_valid.drop(['last\_price', 'price\_class'], axis=1)

target\_valid = df\_valid['price\_class']

# Inicializar variables para el mejor score y el número de estimadores

best\_score = 0

best\_est = 0

# Probar modelos de Random Forest con diferentes números de estimadores

for est in range(1, 11): # Seleccionar el rango del hiperparámetro

model = RandomForestClassifier(random\_state=54321, n\_estimators=est) # Configurar el número de árboles

model.fit(features\_train, target\_train) # Entrenar el modelo en el conjunto de entrenamiento

score = model.score(features\_valid, target\_valid) # Calcular la puntuación de accuracy en el conjunto de validación

if score > best\_score:

best\_score = score # Guardar la mejor puntuación de accuracy en el conjunto de validación

best\_est = est # Guardar el número de estimadores que corresponden a la mejor puntuación de exactitud

# Imprimir la puntuación de exactitud para cada número de estimadores

print("La exactitud del mejor modelo en el conjunto de validación (n\_estimators = {}): {}".format(best\_est, best\_score))

Resultado

La exactitud del mejor modelo en el conjunto de validación (n\_estimators = 10): 0.8860837438423645

¡Es correcto!

Dicen que los bosques tienen oídos y definitivamente ese es el caso con el bosque aleatorio. ¡Nos escucha y nos da excelentes respuestas!

Si jugaste con el rango, podrías notar que si bien puede ser tan grande como desees, probar más de 30 modelos con diferentes números de estimadores uno por uno dará como resultado retrasos notables sin mejorar mucho el valor. Si bien siempre existe la posibilidad de mejorar, esa mejora rápidamente deja de valer el tiempo de cálculo que requiere.

# Regresión logística

El aprendizaje de algoritmos no solo requiere árboles de decisión, sino también otros enfoques para realizar tareas de clasificación.

Si incrementamos el valor del hiperparámetro n\_estimators el modelo se vuelve voluminoso y el proceso de entrenamiento se hace lento, lo cual no es bueno. Pero si mantenemos el número de árboles bajo, los resultados no mejorarán, así que eso también es malo. Quizás sea momento de que cambiemos los árboles por otra cosa. Probemos con la regresión logística. Aunque el nombre sugiere un problema de regresión, sigue siendo un algoritmo de clasificación.

A mediados del siglo XIX, el matemático belga Pierre François Verhulst presentó la función logística. Originalmente, la usó para predecir la dinámica de crecimiento de la población, pero en el siglo XX sus ecuaciones resultaron útiles para una gran variedad de aplicaciones. Al ponerla en un gráfico, la función logística tiene la forma de una curva sigmoide (que tiene forma de s). Por lo general, el eje Y tiene un rango de 0 a 1 y representa la probabilidad de un evento. El eje X traza el gráfico de una variable que afecta a la probabilidad.

Si fuéramos a usar una curva logística para intentar predecir una etiqueta binaria (precio alto/bajo) con base en solo una característica (digamos, el área del apartamento), entonces el modelo entrenado podría representarse visualmente como se muestra a continuación. Agrega puntos al gráfico haciendo clic en ellos y observa cómo la regresión logística los categorizará en precios altos y bajos.

Aquí vemos apartamentos con la etiqueta "1" (costosos) en la parte superior y apartamentos con la etiqueta "0" (baratos) en la inferior. Está claro que, en términos generales, los apartamentos costosos suelen ser más grandes, pero la distinción no está tan marcada. Hay algo de traslape de valores de área donde la etiqueta no es definitiva. La curva representa la probabilidad de que un determinado valor de área corresponda con la etiqueta "1" y el trabajo del algoritmo es encontrar la posición óptima para esta curva. Luego podemos usar la curva resultante para etiquetar nuevos datos. Si la curva y el área determinada nos dan una probabilidad de más de 0.5 (o, en otras palabras, más del 50%), entonces el apartamento lleva la leyenda "1"; de lo contrario, lleva la leyenda "0".

La fórmula de la función logística que respalda a esta curva se puede escribir así:

𝑙𝑛(𝑝1−𝑝)=𝑏0+𝑏1⋅𝑥1*ln*(1−*pp*​)=*b*0​+*b*1​⋅*x*1​

Para obtener la verdadera forma de la función logística necesitarías despejar *p* (donde *p* es la probabilidad de la etiqueta "1", mostrada en un eje vertical), pero esta forma alternativa es más ilustrativa. A la derecha, *x1* es la característica (en este caso, el área). Su coeficiente *b1* es el coeficiente angular y *b0* es el coeficiente lineal. Puedes verlos como la “inclinación” y el “desplazamiento” de una curva, respectivamente. Típicamente, estos parámetros se inicializan como ceros, pero se ajustarán hacia valores positivos o negativos al ejecutar el algoritmo. En este caso, ya que las áreas siempre son positivas y su correlación con los precios también es positiva, el coeficiente lineal será negativo para compensarlo.

El lado izquierdo de la ecuación se llama “proximidad de clase” y se representa con una función log-odds, un logaritmo de las probabilidades. Fíjate que debido a que *p* es la probabilidad de que la etiqueta adecuada sea "1", eso significa que *1-p* es la probabilidad de que la etiqueta adecuada sea "0". Si ambas probabilidades son iguales a 0.5, entonces la proximidad de clase es *ln(1) = 0* y ninguna etiqueta puede ser asignada con confianza. Esto supone dos cosas:

1) Si la proximidad de clase es positiva, el modelo escoge la clase "1" (precio alto); si es negativa, el modelo escoge la clase "0" (precio bajo).

2) Cuanto más lejos esté la proximidad de clase de cero, más seguro está el modelo sobre una de las leyendas. Los valores de proximidad de clase de 1, 2 y 3 corresponden con los valores *p* de 0.73, 0.88 y 0.95. Incluso valores más altos de proximidad de clase harán que *p* tienda a 1 infinitamente. En cambio, valores de proximidad de clase de -1, -2 y -3 corresponden con valores *p* de 0.27, 0.12 y 0.05. Incluso valores más bajos de proximidad de clase harán que *p* tienda a 0 infinitamente.

Durante el entrenamiento, el modelo calcula la proximidad de clase para cada observación en los datos de entrenamiento y les asigna etiquetas. Luego compara las etiquetas que asignó con las reales y ajusta repetidamente los parámetros *b* en la dirección que mejore los resultados de la clasificación.

La regresión logística puede usarse con más de una característica a la vez. Intentemos usar dos: área y distancia. Esta vez, la regresión logística tiene que aprender tres parámetros: dos coeficientes angulares y uno lineal. Después de un determinado número de mejoras iterativas, el modelo puede resultar así:

𝑝𝑟𝑜𝑥𝑖𝑚𝑖𝑑𝑎𝑑 𝑑𝑒 𝑐𝑙𝑎𝑠𝑒=−7+10⋅𝑎ˊ𝑟𝑒𝑎/𝑚2−𝑑𝑖𝑠𝑡𝑎𝑛𝑐𝑖𝑎 𝑑𝑒𝑠𝑑𝑒 𝑒𝑙 𝑐𝑒𝑛𝑡𝑟𝑜 𝑐𝑒𝑛𝑡𝑒𝑟/𝑚*proximidad* *de* *clase*=−7+10⋅*a*ˊ*rea*/*m*2−*distancia* *desde* *el* *centro* *center*/*m*

El precio de un apartamento es directamente proporcional al área e inversamente proporcional a la distancia desde el centro de la ciudad. Cada metro cuadrado del área afecta al precio diez veces más que cada metro de distancia del centro de la ciudad. Aparentemente, un desplazamiento del coeficiente lineal negativo también es necesario.

Se puede incrementar más el número de características, así que, en la práctica, siempre vamos a usar todas las disponibles a la vez. Si una característica no es relevante para la predicción, lo más probable es que el modelo aprenda un coeficiente angular cercano a cero para esa característica, con lo que esencialmente estará ignorándola.

La regresión logística tiene una estructura mucho más sencilla que los árboles de decisión, así que si bien es un tanto rígida, no es tan propensa al sobreajuste.

El modelo LogisticRegression se puede encontrar en el módulo linear\_model de la librería scikit-learn. Impórtalo:

from sklearn.linear\_model import LogisticRegression

Almacena el modelo en una variable y especifica los hiperparámetros. Para obtener uniformidad en los resultados, establece random\_state en 54321.

También necesitamos especificar un solver, una versión del algoritmo que determine qué tan ajustada está exactamente la curva. Por lo general, producen resultados similares. Usaremos el solver 'liblinear' porque es el más general. Funciona bien para conjuntos de datos pequeños con muchas características.

model = LogisticRegression(random\_state=54321, solver='liblinear')

Entrena el modelo llamando al método \*\*fit().

model.fit(features, target)

Llama al método score() para mostrar la *exactitud* del modelo:

model.score(features, target)

Entrena un modelo de regresión logística en el conjunto de entrenamiento, luego calcula el valor de *accuracy* tanto en el conjunto de entrenamiento como en el conjunto de validación.

Especifica random\_state=54321 y solver='liblinear' al inicializar el constructor de modelos.

import pandas as pd

from sklearn.linear\_model import LogisticRegression

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

df = pd.read\_csv('/datasets/train\_data\_us.csv')

df.loc[df['last\_price'] > 113000, 'price\_class'] = 1

df.loc[df['last\_price'] <= 113000, 'price\_class'] = 0

df\_train, df\_valid = train\_test\_split(df, test\_size=0.25, random\_state=54321) # segmenta el 25% de los datos para hacer el conjunto de validación

features\_train = df\_train.drop(['last\_price', 'price\_class'], axis=1)

target\_train = df\_train['price\_class']

features\_valid = df\_valid.drop(['last\_price', 'price\_class'], axis=1)

target\_valid = df\_valid['price\_class']

model = LogisticRegression(random\_state=54321, solver='liblinear')# inicializa el constructor de regresión logística con los parámetros random\_state=54321 y solver='liblinear'

model.fit(features\_train, target\_train) # entrena el modelo en el conjunto de entrenamiento

score\_train = model.score(features\_train, target\_train) # calcula la puntuación de accuracy en el conjunto de entrenamiento

score\_valid = model.score(features\_valid, target\_valid) # calcula la puntuación de accuracy en el conjunto de validación

print("Accuracy del modelo de regresión logística en el conjunto de entrenamiento:", score\_train)

print("Accuracy del modelo de regresión logística en el conjunto de validación:", score\_valid)

Resultado

Accuracy del modelo de regresión logística en el conjunto de entrenamiento: 0.8786696776842537

Accuracy del modelo de regresión logística en el co

Es correcto!

El entrenamiento del modelo es muy rápido. ¡Ahora nuestro modelo puede competir en una carrera de la F1!

*Accuracy* es un poco más baja en comparación con los modelos de bosques aleatorios; pero en el lado positivo, considera cómo el conjunto de entrenamiento *accuracy* no "se adelanta" a los valores de *accuracy* en los conjuntos de validación y prueba. Esto indica la falta de sobreajuste y demuestra cómo la regresión logística es más resistente a él.

Imagen que contiene Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Capítulo 4/7 · Faltan 2 lecciones

Mejora del modelo

# Comparación de modelos

Quienes se dedican a la ciencia de los datos no solo entrenan modelos, sino que también le explican las ventajas y desventajas al cliente. Comparemos nuestros modelos.

En las lecciones anteriores has entrenado a los siguientes modelos:

* árbol de decisión
* bosque aleatorio
* regresión logística

No necesitamos trabajar con los tres modelos al mismo tiempo. Cada uno tiene sus ventajas y desventajas. Vamos a evaluar la *exactitud* y la velocidad de ejecución de los modelos.

1. Exactitud. Este es un criterio clave para las empresas. Una exactitud alta se traduce a ingresos altos.

Un bosque aleatorio tiene la exactitud más alta porque usa un conjunto de árboles en vez de solo uno.

En segundo lugar está la regresión logística. El modelo no es complicado así que no habrá sobreajuste.

Los árboles de decisión tienen la calidad más baja de predicción. Si la profundidad del árbol es menos de 4, el árbol está subajustado; si es mayor a 4, está sobreajustado.

2. Velocidad de ejecución. También es un criterio importante: si el servicio es lento, será inevitable perder usuarios.

La regresión logística es la más rápida porque tiene el menor número de parámetros.

La velocidad de los árboles de decisión también es alta y depende de la profundidad del árbol. Durante nuestros experimentos se obtuvo la mejor calidad de modelo a una profundidad de 4. El modelo obtuvo la respuesta en apenas cuatro pruebas de valor de característica. ¡Bastante rápido!

El bosque aleatorio es el más lento: cuantos más árboles haya, más lento trabaja este modelo.

Vamos a poner los criterios en una tabla:

| **Nombre** | **Exactitud** | **Velocidad** |
| --- | --- | --- |
| árbol de decisión | Bajo | Alto |
| bosque aleatorio | Alto | Bajo |
| regresión logística | Medio | Alto |

¡Usar tablas comparativas como esta puede ayudarte en tu trabajo diario!

Sprint 8

Capítulo 4/7 · Última lección

Mejora del modelo

# Conclusión

En este capítulo aprendiste:

* Cómo verificar la calidad de un modelo
* Qué son los hiperparámetros y cómo afectan a la calidad de predicción
* Cómo "plantar" un bosque aleatorio y entrenar una regresión logística

En el próximo capítulo conocerás una nueva tarea de aprendizaje automático (la regresión), la métrica de evaluación *ECM* y el árbol de regresión.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Mejora del modelo](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Resumen_del_captulo_Mejora_del_modelo.pdf)
* [Hoja informativa: Mejora del modelo](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Hoja_informaiva_Mejora_del_modelo.pdf)

Capítulo 5/7

Pasar a la regresión

# Introducción

Ahora conoces tres algoritmos de clasificación. Dividiste los precios en alto y bajo y predijiste la clase del apartamento. Es momento de volver a ese ejercicio de regresión.

## Empezarás por:

* Aprender a resolver tareas de regresión.
* Aprender a evaluar modelos de regresión usando el *ECM* (error cuadrático medio).
* Entrenar un árbol de decisión, un bosque aleatorio y un modelo de regresión lineal para la tarea de regresión.

## ¿Cuánto tiempo tomará?

10 lecciones de 5-10 minutos cada una

## Descripción del ejercicio

Ya no necesitamos dividir los apartamentos en costosos y baratos. Entrenaremos un modelo que podrá recomendar un precio de apartamento en dólares.

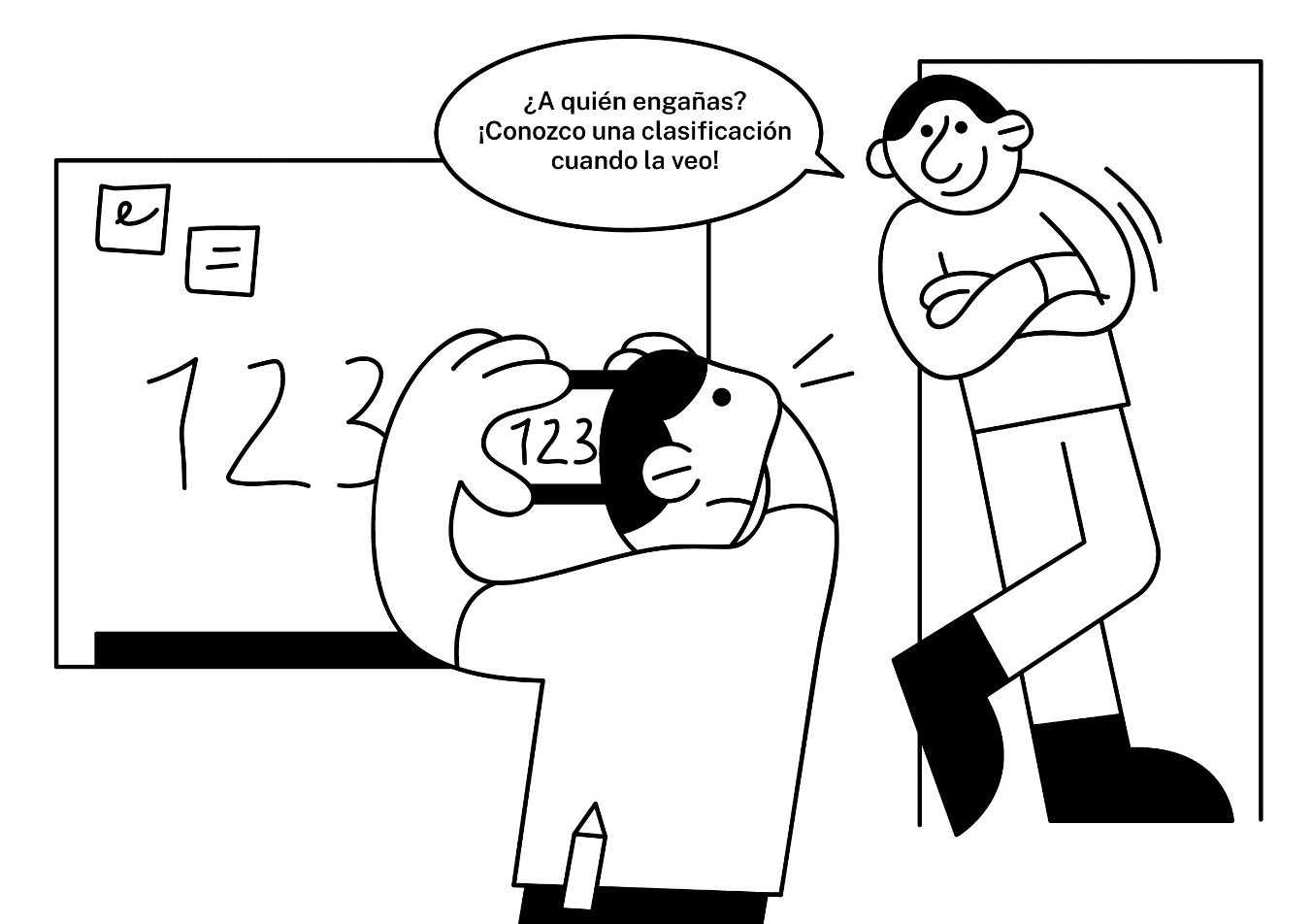
Capítulo 5/7

Pasar a la regresión

# Regresión

¿En qué se diferencian la clasificación y la regresión? Vamos a repasar antes de entrenar un nuevo modelo.

Un objetivo (respuesta) puede ser categórico o numérico. Un objetivo ctaegórico indica un problema de tipo clasificación. Si el objetivo es numérico, la tarea es regresión. Si el objetivo es numérico, la tarea es regresión.



Pregunta

¿Qué tareas pertenecen a la regresión? Selecciona todas las opciones que correspondan:

Elige tantas como quieras

Predecir el precio más alto que las personas están dispuestas a pagar por un taxi.

El objetivo es numérico así que es una tarea de regresión.

Pronosticar cuántos boletos se venderán para las proyección de una película en la mañana y en la tarde.

El objetivo es numérico así que es una tarea de regresión.

Determinar el grupo de edad de una persona: menor de 18, menor de 35, mayor de 35.

Determinar cuántos días tomará tramitar una visa con base en el país, el tipo de permiso de entrada y la información de quien solicita.

El objetivo es numérico así que la tarea es regresión.

¡Excelente trabajo!

El precio de un apartamento es un objetivo numérico. Entonces, vayamos a la regresión.

Capítulo 5/7

Pasar a la regresión

# Error cuadrático medio

¿Qué métrica es la más utilizada en una tarea de regresión? ¡El error cuadrático medio!

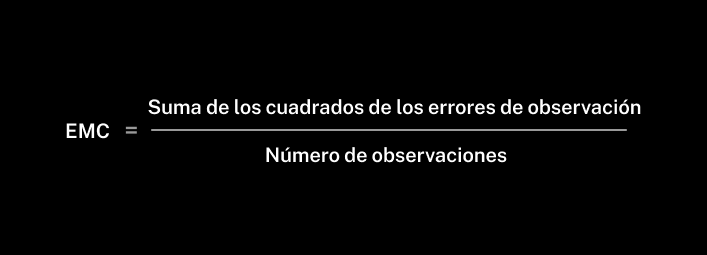
¿Cómo debemos definir la "respuesta correcta" para nuestra tarea? ¿Debería ser una coincidencia exacta de precio? ¿Qué pasaría si el modelo errara por 10 centavos? ¿Sería un fracaso? Obviamente, la métrica de *exactitud* ya no es válida.

La métrica de evaluación más comúnmente usada para las tareas de regresión es el error cuadrático medio o *EMC* (en inglés *MSE*).

Para encontrar el *EMC*, primero debes calcular el error de cada observación:



Calcula el *EMC* usando esta fórmula.



Vamos a analizar estos cálculos:

1. El error de observación muestra el grado de discrepancia entre la respuesta correcta y la predicción. Si el error es mucho más grande que cero, el modelo ha sobrevalorado el apartamento; si es mucho menor que cero, entonces el modelo le ha dado un precio reducido.
2. No tendría caso sumar los errores como están, ya que los positivos anularían los negativos. Para hacer que todos cuenten, necesitamos deshacernos de los signos elevándolos al cuadrado.
3. Calculamos la media para obtener un valor representativo del conjunto completo de observaciones.

En las lecciones anteriores intentamos obtener la mayor *exactitud*. Por otro lado, el *EMC* debe ser lo más bajo posible.

Recuerda que al elevar al cuadrado el *EMC* no tendrá la misma unidad de medida que nuestra variable objetivo. Probablemente recuerdes que nos enfrentamos a un problema similar en el curso de Estadística cuando intentábamos medir la dispersión con varianza. ¿Recuerdas qué hicimos para sacar las unidades originales? ¡Sacamos una raíz cuadrada! Si tomas la raíz cuadrada del *EMC* obtendrás una métrica llamada *RECM* (en inglés *RMSE*), que significa *raíz del error cuadrático medio*. Tiene las mismas unidades de medida que la variable objetivo.

Imagina que tu startup se lanza al mercado latinoamericano. Creaste un modelo de regresión que predice los presupuestos para campañas de publicidad. La tabla muestra la predicción y el costo real.

| **País** | **Predicción** | **Precio real** |
| --- | --- | --- |
| Brasil | 623 | 649 |
| Perú | 253 | 253 |
| México | 150 | 370 |
| Colombia | 237 | 148 |

Pregunta

Con la ayuda de una calculadora o de EMC Python, calcula el EMC y selecciona el valor correcto:

-157

-39.25

39.25

157

6162.25

14249.25

¡Correcto! ¡Machu Picchu, ahí vamos! Eh... hipotéticamente.

24649

56997

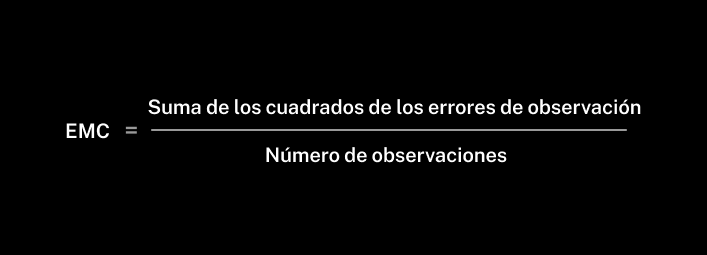
¡Lo has entendido bien!

# Calcular el EMC

Vamos a automatizar el cálculo del *EMC*.

¿Cómo calculaste el *EMC* en el ejercicio pasado? ¿Lo hiciste manualmente con una calculadora o programaste un sencillo programa en Python?

Automatiza este cálculo:



1.

Escribe una función mse(). Debe tomar respuestas correctas y predicciones y devolver el valor del error cuadrático medio.

Tomamos las respuestas (costos reales) y las predicciones de la tabla. Imprime en la pantalla el valor del *EMC*.

def mse(answers, predictions):

total = 0

for i in range(len(answers)):

total += (answers[i] - predictions[i]) \*\* 2 # Agrega el error cuadrático

result = total / len(answers) # Divide la suma entre el número de observaciones

return result  
  
Resultado

14249.25

¡Es correcto!

¡Buen trabajo! La máquina y tú obtuvieron la misma respuesta.

2.

También hay una función para calcular el *EMC* en *sklearn*.

Busca en la documentación el nombre de la función y cómo opera. Impórtala y resuelve el mismo problema. Imprime en la pantalla el valor del *EMC*.

Aquí puedes encontrar la documentación de la librería *sklearn*: <https://scikit-learn.org/stable/modules/classes.html>. Si pierdes el vínculo, haz una búsqueda con "*sklearn reference"* como frase clave.

*EMC* (mean\_squared\_error) está almacenada en el módulo sklearn.metrics.

from sklearn.metrics import mean\_squared\_error

answers = [623, 253, 150, 237]

predictions = [649, 253, 370, 148]

result = mean\_squared\_error(answers, predictions)

print(result)

Resultado

14249.25

Es correcto!

El mismo valor del EMC pero casi sin esfuerzo.

# Interpretación del ECM

Para hacer la prueba de cordura al modelo para buscar problemas de clasificación necesitamos comparar sus predicciones con posibilidad aleatoria. ¿Cómo se hace una prueba de cordura para un modelo de regresión?

Darle la misma respuesta a todas las observaciones es un método simple de predicción de regresión. Vamos a usar el valor promedio del precio del apartamento para estar más cerca de la realidad.

1.

Prepara los datos y encuentra el precio promedio.

* Declara la variable *features* con todas las características excepto last\_price.
* Declara la variable *target* con *last\_price* como objetivo.
* Calcula el valor promedio para los elementos de la variable *target*.

Imprime el resultado como se muestra:

Average price: ...

## Pistas

Este es el código para declarar las variables:

features = df.drop(['last\_price'], axis=1)

target = df['last\_price']

import pandas as pd

# Cargar los datos

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear las variables features y target

features = df.drop(['last\_price'], axis=1)

target = df['last\_price']

# Calcular el promedio del precio

average\_price = target.mean()

# Imprimir el resultado

print(f"Average price: {average\_price}")

Average price: 161005.67427559663

¡Es correcto!

¡Encontraste el precio promedio!

2.

Calcula el *ECM* para el conjunto de entrenamiento usando el precio promedio como valor de predicción.

Imprime el resultado como se muestra:

MSE: ...

La función *mean\_squared\_error()* en scikit-learn tiene su maña. Tendrás que pasar unos minutos leyendo la documentación o consultar *Stack Overflow*.

En el precódigo, dividimos los precios entre 100 000 con el propósito de evitar números grandes.

## Pistas

La función mean\_squared\_error() solo toma objetos de tipo matriz como sus argumentos. No acepta valores escalares así que necesitas enviar la media al tamaño del objetivo. Una manera de hacerlo es crear una serie así:

predictions = pd.Series(target.mean(), index=target.index)

import pandas as pd

from sklearn.metrics import mean\_squared\_error

# Cargar los datos

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Crear las variables features y target

features = df.drop(['last\_price'], axis=1)

target = df['last\_price'] / 100000

# Calcular la media del target

mean\_price = target.mean()

# Crear una serie de predicciones utilizando la media

predictions = pd.Series(mean\_price, index=target.index)

# Calcular el ECM

mse = mean\_squared\_error(target, predictions)

# Imprimir el resultado

print('MSE:', mse)

Resultado

MSE: 5.529775874409322

¡Es correcto!

Si elevas las diferencias al cuadrado, obtendrás el resultado en "dólares cuadrados". Recibir dinero en esa divisa sería buenísimo. Pero para propósitos de nuestro ejercicio tendremos que arreglarlo.

3.

No necesitamos "dólares cuadrados". Para obtener dólares normales encuentra la *RECM* sacando la raíz cuadrada del *ECM*. Imprime en la pantalla los resultados como se muestra:

RMSE: ...

Para encontrar la raíz cuadrada de un número usa el operador de exponente \*\*. Eleva el número a la potencia 0.5. Por ejemplo:

print(25 \*\* 0.5)

5

Con la ayuda del operador \*\* eleva el ECM a la potencia 0.5.

import pandas as pd

from sklearn.metrics import mean\_squared\_error

df = pd.read\_csv('/datasets/train\_data\_us.csv')

features = df.drop(['last\_price'], axis=1)

target = df['last\_price'] / 100000

predictions = pd.Series(target.mean(), index=target.index)

mse = mean\_squared\_error(target, predictions)

rmse = mse \*\* 0.5# < encuentra la raíz cuadrada de ECM >

print('RMSE:', rmse)

Resultado

RMSE: 2.3515475488302

¡Es correcto!

La *RECM* nos indica que, en promedio, nuestras predicciones erraron por aproximadamente 235 000 dólares (no cuadrados). Parece mucho pero es aceptable para un modelo que siempre responde con el valor medio, dado que los precios de apartamentos más caros en nuestro conjunto llegan hasta 8 millones de dólares.

# rbol de decisión de regresión

### Se puede usar un árbol de decisión para la regresión y también para la clasificación.

Para tareas de regresión, los árboles de decisión se entrenan de manera similar a la clasificación pero no predicen una clase, sino un número.

Por ejemplo, Gabriel llegó por fin a Hawái. ¿Cuánto le costará un boleto para el Museo de Arte de Caricaturas?

Diagrama

Descripción generada automáticamente

1. Destina 25% de los datos al conjunto de validación y el resto para el de entrenamiento.
2. Entrena modelos de árbol de decisión para un problema de regresión con diferentes valores de profundidad entre 1 y 6.
3. Calcula el valor de *RECM* en el conjunto de validación para cada modelo.
4. Almacena en la variable best\_model el modelo con el mejor valor de *RECM* en el conjunto de validación. Para calcular la métrica *RECM*, toma el valor de la raíz cuadrada del *ECM*:

¿Es un buen resultado? Comparémoslo con otros modelos. Por lo menos podemos tener la certeza de que nuestro modelo pasó la prueba de cordura.

Para calcular la métrica *RECM*, toma el valor de la raíz cuadrada del *ECM*:

mean\_squared\_error(target\_valid, predictions\_valid)\*\*0.5

import pandas as pd

from sklearn.tree import DecisionTreeRegressor

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

from sklearn.metrics import mean\_squared\_error

# Cargar los datos

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Separar las características y el objetivo

features = df.drop(['last\_price'], axis=1)

target = df['last\_price'] / 1000000

# Dividir los datos en conjuntos de entrenamiento y validación

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345)

best\_model = None

best\_result = 10000

best\_depth = 0

# Entrenar modelos con diferentes profundidades

for depth in range(1, 6): # selecciona el rango del hiperparámetro

model = DecisionTreeRegressor(max\_depth=depth, random\_state=12345) # define el modelo con la profundidad actual

model.fit(features\_train, target\_train) # entrena el modelo en el conjunto de entrenamiento

predictions\_valid = model.predict(features\_valid) # obtén las predicciones del modelo en el conjunto de validación

result = mean\_squared\_error(target\_valid, predictions\_valid) \*\* 0.5 # calcula la RECM en el conjunto de validación

if result < best\_result:

best\_model = model

best\_result = result

best\_depth = depth

print(f"RECM del mejor modelo en el conjunto de validación (max\_depth = {best\_depth}): {best\_result}")

Resultado

RECM del mejor modelo en el conjunto de validación (max\_depth = 5): 0.1308258643787925

¡Es correcto!

¿Es un buen resultado? Comparémoslo con otros modelos. Por lo menos podemos tener la certeza de que nuestro modelo pasó la prueba de cordura.

# Bosque aleatorio de regresión

Si lo piensas, bosque aleatorio es un nombre lindo... En fin, sigamos con el entrenamiento de un modelo de bosque aleatorio para regresión.

El bosque aleatorio de regresión no difiere mucho del de clasificación. Lo que hace es entrenar a un grupo de árboles independientes y luego promedia sus respuestas para tomar una decisión.

1. Selecciona 25% de los datos para la muestra de validación (prueba), lo demás será para el entrenamiento.
2. Extrae características y objetivos para el entrenamiento y la validación. Para conseguir objetivos, selecciona la columna 'last\_price' y divídela entre 100000. La característica son todas las columnas excepto por la columna 'last\_price'.
3. Entrena modelos de bosque aleatorio para el problema de regresión:
   * con el número de árboles: de 10 a 50, en intervalos de 10,
   * con una profundidad máxima de 1 a 10.
4. Para cada modelo, calcula la *RECM* en el conjunto de validación y guárdalo en la variable error.

Para calcular la métrica *RECM*, toma el valor de la raíz cuadrada del *ECM*:

mean\_squared\_error(target\_valid, predictions\_valid)\*\*0.5

El código se puede ejecutar durante cerca de un minuto. Esto es normal porque estás entrenando 50 modelos.

[Dataset para este ejercicio](https://practicum-content.s3.us-west-1.amazonaws.com/datasets/moved_train_data_us.csv)

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.metrics import mean\_squared\_error

# Cargar los datos

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Dividir los datos en conjuntos de entrenamiento y validación

df\_train, df\_valid = train\_test\_split(df, random\_state=54321, test\_size=0.25)

# Extraer las características y los objetivos para el entrenamiento

features\_train = df\_train.drop(['last\_price'], axis=1)

target\_train = df\_train['last\_price'] / 100000

# Extraer las características y los objetivos para la validación

features\_valid = df\_valid.drop(['last\_price'], axis=1)

target\_valid = df\_valid['last\_price'] / 100000

# Configuración inicial para encontrar el mejor modelo

best\_error = 10000

best\_est = 0

best\_depth = 0

# Entrenar modelos con diferentes números de árboles y profundidades

for est in range(10, 51, 10):

for depth in range(1, 11):

model = RandomForestRegressor(random\_state=54321, n\_estimators=est, max\_depth=depth)

model.fit(features\_train, target\_train)

predictions\_valid = model.predict(features\_valid)

error = mean\_squared\_error(target\_valid, predictions\_valid) \*\* 0.5

print(f"Validación RECM para n\_estimators={est}, max\_depth={depth} es {error}")

if error < best\_error:

best\_error = error

best\_est = est

best\_depth = depth

print(f"RECM del mejor modelo en el conjunto de validación: {best\_error}, n\_estimators: {best\_est}, max\_depth: {best\_depth}")

Resultado

Validación RECM para n\_estimators=10, max\_depth=1 es 2.3960212238796124

Validación RECM para n\_estimators=10, max\_depth=2 es 2.1122446639688435

Validación RECM para n\_estimators=10, max\_depth=3 es 1.9740959499997468

Validación RECM para n\_estimators=10, max\_depth=4 es 1.8427098576557455

Validación RECM para n\_estimators=10, max\_depth=5 es 1.784417875686694

Validación RECM para n\_estimators=10, max\_depth=6 es 1.8702209720173095

Validación RECM para n\_estimators=10, max\_depth=7 es 1.793159121509078

Validación RECM para n\_estimators=10, max\_depth=8 es 1.7468096998889795

Validación RECM para n\_estimators=10, max\_depth=9 es 1.7909475486091944

Validación RECM para n\_estimators=10, max\_depth=10 es 1.730814097537693

Validación RECM para n\_estimators=20, max\_depth=1 es 2.385864337512238

Validación RECM para n\_estimators=20, max\_depth=2 es 2.114680438966696

Validación RECM para n\_estimators=20, max\_depth=3 es 1.9852473762330756

Validación RECM para n\_estimators=20, max\_depth=4 es 1.9033912628930718

Validación RECM para n\_estimators=20, max\_depth=5 es 1.8566404726957315

Validación RECM para n\_estimators=20, max\_depth=6 es 1.8217357628413762

Validación RECM para n\_estimators=20, max\_depth=7 es 1.734104553475037

Validación RECM para n\_estimators=20, max\_depth=8 es 1.76963432774949

Validación RECM para n\_estimators=20, max\_depth=9 es 1.7537621632785976

Validación RECM para n\_estimators=20, max\_depth=10 es 1.7354093982135819

Validación RECM para n\_estimators=30, max\_depth=1 es 2.3699493863231695

Validación RECM para n\_estimators=30, max\_depth=2 es 2.0992551483640454

Validación RECM para n\_estimators=30, max\_depth=3 es 1.9781820690170426

Validación RECM para n\_estimators=30, max\_depth=4 es 1.917102290123685

Validación RECM para n\_estimators=30, max\_depth=5 es 1.8736445688363852

Validación RECM para n\_estimators=30, max\_depth=6 es 1.8220494163346614

Validación RECM para n\_estimators=30, max\_depth=7 es 1.7716174496210542

Validación RECM para n\_estimators=30, max\_depth=8 es 1.7625178458445354

Validación RECM para n\_estimators=30, max\_depth=9 es 1.7672430407213024

Validación RECM para n\_estimators=30, max\_depth=10 es 1.7264573220445658

Validación RECM para n\_estimators=40, max\_depth=1 es 2.3982305588099453

Validación RECM para n\_estimators=40, max\_depth=2 es 2.1016265917767525

Validación RECM para n\_estimators=40, max\_depth=3 es 1.988383643555704

Validación RECM para n\_estimators=40, max\_depth=4 es 1.931586227717903

Validación RECM para n\_estimators=40, max\_depth=5 es 1.873157638874597

Validación RECM para n\_estimators=40, max\_depth=6 es 1.8312503349110927

Validación RECM para n\_estimators=40, max\_depth=7 es 1.7877778529360793

Validación RECM para n\_estimators=40, max\_depth=8 es 1.7882013173154068

Validación RECM para n\_estimators=40, max\_depth=9 es 1.7736774076707327

Validación RECM para n\_estimators=40, max\_depth=10 es 1.729298708232932

Validación RECM para n\_estimators=50, max\_depth=1 es 2.3714687109562695

Validación RECM para n\_estimators=50, max\_depth=2 es 2.100286153891176

Validación RECM para n\_estimators=50, max\_depth=3 es 1.9822059638620555

Validación RECM para n\_estimators=50, max\_depth=4 es 1.9311502876246265

Validación RECM para n\_estimators=50, max\_depth=5 es 1.8375155830929193

Validación RECM para n\_estimators=50, max\_depth=6 es 1.8289930496830193

Validación RECM para n\_estimators=50, max\_depth=7 es 1.774507099117585

Validación RECM para n\_estimators=50, max\_depth=8 es 1.7620920773763233

Validación RECM para n\_estimators=50, max\_depth=9 es 1.7596642410297794

Validación RECM para n\_estimators=50, max\_depth=10 es 1.718843030335708

RECM del mejor modelo en el conjunto de validación: 1.718843030335708, n\_estimators:

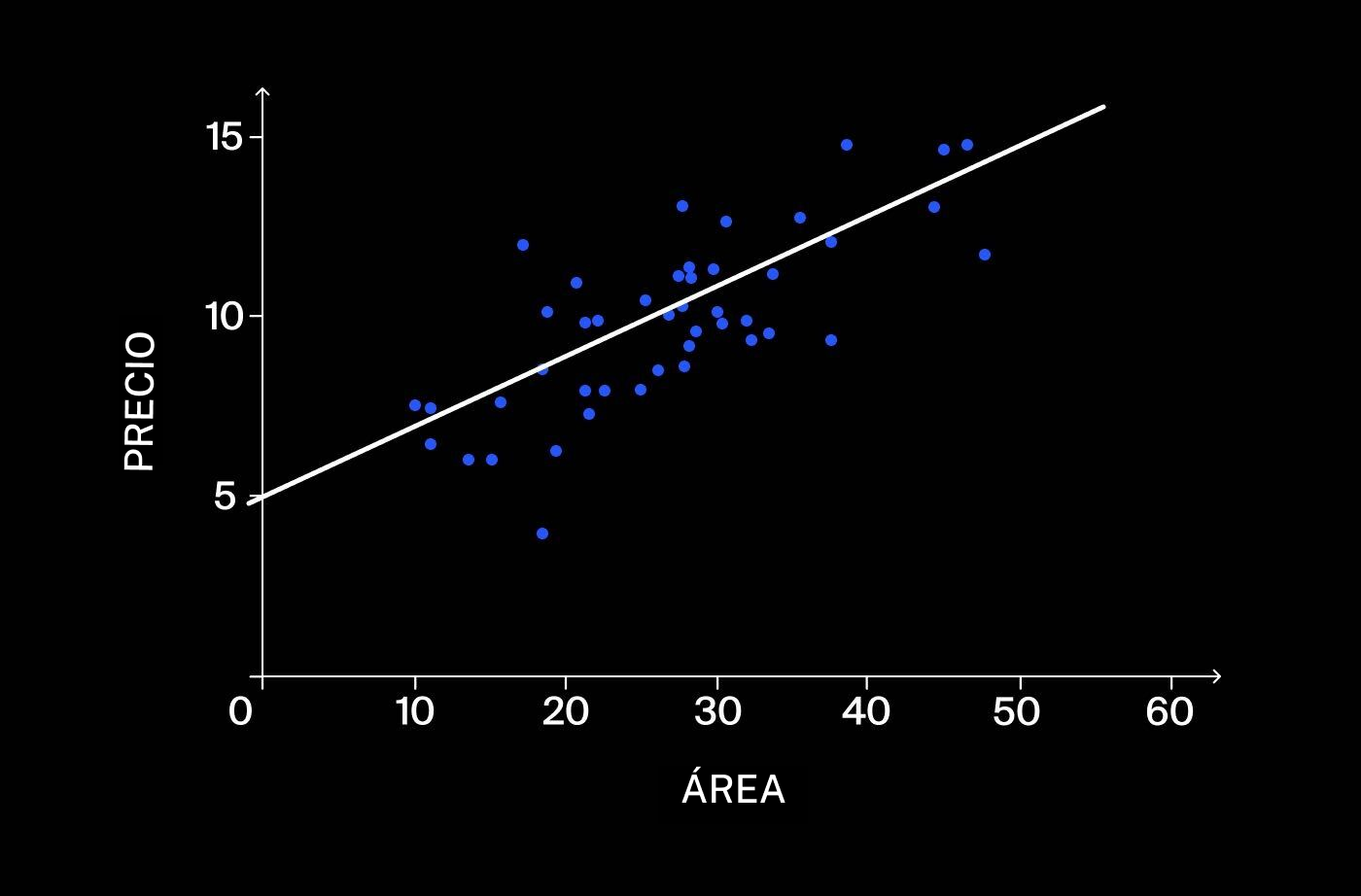
¡Es correcto!

Un árbol está bien, pero no puedes vencer a un bosque: la *RECM* para el conjunto de prueba es menor. Y no te preocupes si es mayor para el conjunto de entrenamiento, eso se debe a que el árbol de decisión está sobreajustado.

# Regresión lineal

### ¿Qué algoritmo vamos a usar en vez de regresión logística? ¡Así es! Regresión lineal.

La idea general de la regresión lineal es que hay una sencilla ecuación lineal que describe cómo un determinado parámetro define el output. Primero, imaginemos que la imagen de abajo nos muestra cómo se relaciona el precio con el área:



La primera línea muestra la relación aproximada entre ambos. La recta puede describirse mediante una de las ecuaciones algebraicas más sencillas: la ecuación lineal y = ax + b. Veámoslo más detenidamente:

*y* es el objetivo, una variable dependiente que queremos encontrar. En este caso, es el precio. *x* es la característica, una variable independiente que define a la dependiente. Aquí es el área. Fíjate que, a diferencia de la regresión logística, el lado derecho de la ecuación afecta directamente a la variable dependiente en vez de mediante una curva log-odds. Por eso es una línea recta.

*a* representa la pendiente y muestra dos cosas. En primer lugar, si es *positiva* el precio aumenta junto con el área; si es negativa, el precio baja conforme el área aumenta. En segundo lugar, muestra cuánto cambia el precio si el área aumenta por cada unidad. En el gráfico de arriba, el precio aumenta aproximadamente 5 unidades por cada 30 unidades del área. Eso significa que la pendiente es 5/30 o 1/6.

*b* se llama coeficiente lineal. Es un parámetro independiente que desplaza la línea a lo largo del eje y. Revela la cantidad fija añadida o sustraída del precio, independiente del tamaño del área. Puedes verlo en un gráfico donde el área es 0 y la línea cruza el eje y. En este caso es 5.

El algoritmo de regresión lineal determina los valores óptimos para *a* y *b* de manera que traza la línea lo más cerca posible de los puntos (para eso, minimiza el ECM). En última instancia, ajustar un modelo lineal sirve para determinar estos parámetros.

Los nombres de términos como a y b pueden variar en diferentes contextos. Sin embargo, el principio fundamental de la regresión lineal se mantiene.

Ahora vamos a usar dos características: el área y la distancia desde el centro de la ciudad. Eso significa que la regresión lineal tiene que aprender tres parámetros: dos coeficientes de pendiente y un término independiente. La ecuación se ve algo así:

𝑝𝑟𝑒𝑐𝑖𝑜=𝑎ˊ𝑟𝑒𝑎⋅3 000 𝑑𝑜𝑙./𝑚2−(𝑑𝑖𝑠𝑡𝑎𝑛𝑐𝑖𝑎 𝑑𝑒𝑠𝑑𝑒 𝑒𝑙 𝑐𝑒𝑛𝑡𝑟𝑜 𝑑𝑒 𝑙𝑎 𝑐𝑖𝑢𝑑𝑎𝑑)⋅4 𝑑𝑜𝑙./𝑚+99*precio*=*a*ˊ*rea*⋅3 000 *dol*./*m*2−(*distancia* *desde* *el* *centro* *de* *la* *ciudad*)⋅4 *dol*./*m*+99

El modelo ha determinado que el valor del apartamento se incrementa en 3000 dólares por cada metro cuadrado adicional, mientras que disminuye en 4 dólares por cada metro alejado del centro de la ciudad. Adicionalmente, indica que, en promedio, el precio base de los apartamentos incluye un suplemento de 99 dólares. Aunque es teóricamente improbable encontrar un apartamento sin superficie y ubicado exactamente en el centro, nuestro modelo sugiere que tal apartamento tendría un costo inicial de 99 dólares.

Para representar la relación entre dos variables independientes y una dependiente necesitamos usar un espacio tridimensional. La ecuación de arriba representa un plano que divide el espacio en dos. Pero no vamos a representar eso, ni tú tampoco necesitas hacerlo. Con cada característica que añadimos las dimensiones de nuestro modelo aumentan en 1 y, de todos modos, no habrá un método directo para representar más de 3 dimensiones. Por fortuna, eso no obstaculiza la efectividad de las predicciones del modelo. Las matemáticas pueden trabajar con cualquier número de dimensiones aunque nuestro cerebro no las pueda visualizar.

La regresión lineal es un tanto rígida, no logrará ajustarse perfectamente a todos los puntos de datos. Por un lado, esto significa que algunas configuraciones de puntos podrían ser demasiado complejas para este modelo simple. Necesitaríamos usar otra cosa en estos casos. Pero, por el otro, la ventaja de la rigidez es que el sobreajuste no afecta a la regresión lineal tanto como a los árboles de decisión.

Si esta explicación de la regresión lineal suena muy sosa, ¡es porque lo es! Afortunadamente, pronto habrá un curso entero sobre ello. Por ahora, crearemos y entrenaremos un modelo de regresión lineal simple para que puedas ver cómo funciona.

Lleva las características de entrenamiento a la variable features y la característica last\_price del objetivo a la variable target. Divide el valor de la característica target entre 1000000.

Inicializa un modelo de regresión lineal y entrénalo. Calcula el valor de la métrica *RECM* en el conjunto de validación y almacénalo en la variable result.

Usa el método drop para extraer todas las características, excepto last\_price.

Para calcular la métrica RECM, toma la raíz cuadrada del valor de ECM.

import pandas as pd

from sklearn.linear\_model import LinearRegression

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

from sklearn.metrics import mean\_squared\_error

# Cargar los datos

df = pd.read\_csv('/datasets/train\_data\_us.csv')

# Extraer las características y el objetivo

features = df.drop(['last\_price'], axis=1)

target = df['last\_price'] / 1000000

# Dividir los datos en conjuntos de entrenamiento y validación

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(features, target, test\_size=0.25, random\_state=12345)

# Inicializar el modelo de regresión lineal

model = LinearRegression()

# Entrenar el modelo en el conjunto de entrenamiento

model.fit(features\_train, target\_train)

# Obtener las predicciones del modelo en el conjunto de validación

predictions\_valid = model.predict(features\_valid)

# Calcular la RECM en el conjunto de validación

result = mean\_squared\_error(target\_valid, predictions\_valid) \*\* 0.5

# Imprimir el resultado

print("RECM del modelo de regresión lineal en el conjunto de validación:", result)

Resultado

RECM del modelo de regresión lineal en el conjunto de validación: 0.15452013394017108

¡Es correcto!

La regresión lineal no sobreajusta. Los valores de la *RECM* para el conjunto de entrenamiento y para el de prueba son casi los mismos.

# Seleccionar el mejor modelo

### Hemos entrenado un árbol de decisión, un bosque aleatorio y una regresión lineal. ¿Cuál es el mejor modelo?

A estas alturas, probablemente ya tengas una idea de cuál de los tres modelos de regresión funciona mejor para nuestro ejercicio. Si aún no lo tienes claro, repasa las tres lecciones anteriores y compara los valores del RMSE para el conjunto de validación de los modelos que mejor desempeño mostraron. Escoge un modelo, configura sus hiperparámetros con valores que demostraron ser óptimos y entrénalo para una prueba final.

Usa todo el dataset fuente para entrenar el modelo que escogiste. Cuantos más datos haya, mejor. Ya no necesitamos mantener separado el conjunto de validación porque ya escogimos el modelo más adecuado y ajustamos sus hiperparámetros.

Para aprobar el ejercicio, entrena el mejor modelo con hiperparámetros óptimos en todo el dataset. Si no tienes la seguridad de cuál es el mejor, tómate la libertad de intentarlo con los tres.

Algunos de los modelos que entrenamos lograron obtener valores de *RECM* de prueba tan bajos como 1.45. Ahora que estamos usando todo el dataset fuente para el entrenamiento podemos esperar que el desempeño del modelo mejore aún más. Intenta romper este récord: a ver si puedes bajar la *RECM* de prueba a menos de 1.44.

No olvides dividir la variable target entre 100 000 y usar random\_state=54321 como en los ejercicios anteriores.

import pandas as pd

from sklearn.metrics import mean\_squared\_error

from sklearn.ensemble import RandomForestRegressor

df = pd.read\_csv('/datasets/train\_data\_us.csv')

features = df.drop(['last\_price'], axis=1)

target = df['last\_price'] / 100000

final\_model = RandomForestRegressor(random\_state=54321, n\_estimators=40)

final\_model.fit(features, target)

predictions = final\_model.predict(features)

result = mean\_squared\_error(target, predictions)\*\*0.5

print(result)

Resultado

0.6199495797913459

¡Es correcto!

En efecto, sí hay una pequeña mejoría gracias a que usaste un poco más de datos para el entrenamiento. Y este resultado se puede mejorar más si usas más estimadores, aunque no por mucho.

¿Por qué ganó este modelo específico con estos hiperparámetros específicos? ¡Buena pregunta! Pero no tiene respuesta. A veces solo necesitas dar vueltas entre diferentes opciones, revisar los resultados y escoger el mejor modelo.

# Conclusión

Ahora sabes:

* Entrenar un árbol de decisión, un bosque aleatorio y un modelo de regresión lineal para tareas de regresión;
* Calcular el *ECM*;
* Comparar la calidad de modelos de regresión.

Hiciste tu investigación y preparaste un prototipo para resolver el ejercicio. No hemos logrado obtener cero errores, pero no te preocupes, esto es prácticamente imposible en el aprendizaje automático. Ahora sigue el proyecto.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Pasar a la regresión](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Resumen_del_captulo_Pasar_a_la_regresin.pdf)
* [Hoja informativa: Pasar a la regresión](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/moved_Hoja_informativa_Pasar_a_la_regresin.pdf)

¡Felicidades! Terminaste Introducción al machine learning. Es hora de usar todo el conocimiento y habilidades que adquiriste en un proyecto: un estudio de caso real que harás por tu cuenta.

Cuando finalices el proyecto envía tu trabajo al revisor de proyecto para su evaluación. Te dará su opinión en 48 horas. Utiliza los comentarios para realizar cambios y luego envía la nueva versión al revisor del proyecto.

Por lo general, este proceso se repetirá varias veces hasta que recibas el visto bueno de la revisión y se aprueben todas las correcciones. Todo eso es parte del trabajo.

**Descripción del proyecto**

La compañía móvil Megaline no está satisfecha al ver que muchos de sus clientes utilizan planes heredados. Quieren desarrollar un modelo que pueda analizar el comportamiento de los clientes y recomendar uno de los nuevos planes de Megaline: Smart o Ultra.

Tienes acceso a los datos de comportamiento de los suscriptores que ya se han cambiado a los planes nuevos (del proyecto del sprint de Análisis estadístico de datos). Para esta tarea de clasificación debes crear un modelo que escoja el plan correcto. Como ya hiciste el paso de procesar los datos, puedes lanzarte directo a crear el modelo.

Desarrolla un modelo con la mayor *exactitud* posible. En este proyecto, el umbral de *exactitud* es 0.75. Usa el dataset para comprobar la *exactitud*.

**Instrucciones del proyecto.**

1. Abre y examina el archivo de datos. Dirección al archivo:datasets/users\_behavior.csv [Descarga el dataset](https://practicum-content.s3.us-west-1.amazonaws.com/datasets/users_behavior.csv)
2. Segmenta los datos fuente en un conjunto de entrenamiento, uno de validación y uno de prueba.
3. Investiga la calidad de diferentes modelos cambiando los hiperparámetros. Describe brevemente los hallazgos del estudio.
4. Comprueba la calidad del modelo usando el conjunto de prueba.
5. Tarea adicional: haz una prueba de cordura al modelo. Estos datos son más complejos que los que habías usado antes así que no será una tarea fácil. Más adelante lo veremos con más detalle.

**Descripción de datos**

Cada observación en el dataset contiene información del comportamiento mensual sobre un usuario. La información dada es la siguiente:

* сalls — número de llamadas,
* minutes — duración total de la llamada en minutos,
* messages — número de mensajes de texto,
* mb\_used — Tráfico de Internet utilizado en MB,
* is\_ultra — plan para el mes actual (Ultra - 1, Smart - 0).

**Evaluación del proyecto**

Hemos definido los criterios de evaluación para el proyecto. Lee esto con atención antes de pasar al ejercicio.

Esto es lo que los revisores buscarán cuando evalúen tu proyecto:

* ¿Cómo leíste los datos después de descargarlos?
* ¿Segmentaste correctamente los datos en conjuntos de entrenamiento, validación y prueba?
* ¿Cómo escogiste el tamaño de los conjuntos?
* ¿Evaluaste correctamente la calidad del modelo?
* ¿Qué modelos e hiperparámentros usaste?
* ¿Cuáles fueron tus hallazgos?
* ¿Probaste los modelos correctamente?
* ¿Cuál es tu puntuación de exactitud?
* ¿Te ceñiste a la estructura del proyecto y mantuviste limpio el código?

Tienes tus hojas informativas y los resúmenes de los capítulos así que ya puedes continuar con el proyecto.

¡Buena suerte!

# Conclusión

¡Felicidades! Has terminado el curso introductorio sobre machine learning.

Aprendiste a:

* Distinguir la clasificación de la regresión; un modelo para un algoritmo de aprendizaje
* Entrenar un modelo y medir su calidad con la librería *sklearn*
* Evaluar y hacer pruebas de modelos

El siguiente curso también estará dedicado al aprendizaje supervisado. Aprenderás nuevas métricas de evaluación y explorarás nuevas formas de mejorar los modelos.