- El aprendizaje automático consiste en hacer que las máquinas mejoren en alguna tarea aprendiendo de los datos, en lugar de tener que codificar reglas explícitamente.

- Hay muchos tipos diferentes de sistemas de AA: supervisados o no, por lotes o en línea, basados en instancias o basados en modelos, etc.

- En un proyecto de AA, usted recopila datos en un conjunto de entrenamiento y alimenta el conjunto de entrenamiento a un algoritmo de aprendizaje. Si el algoritmo está basado en modelos, sintoniza algunos parámetros para ajustar el modelo al conjunto de entrenamiento (es decir, para hacer buenas predicciones sobre el conjunto de entrenamiento), y luego, con suerte, también podrá hacer buenas predicciones en nuevos casos. Si el algoritmo está basado en instancias, simplemente aprende los ejemplos de memoria y usa una medida de similitud para generalizar a nuevas instancias.

- El sistema no funcionará bien si su conjunto de entrenamiento es demasiado pequeño, o si los datos no son representativos, ruidosos o están contaminados con características irrelevantes (basura dentro, basura fuera). Por último, su modelo no debe ser ni demasiado simple (en cuyo caso no se ajustará) ni demasiado complejo (en cuyo caso se sobreajustará).

Solo hay un último tema importante que cubrir: una vez que haya entrenado un modelo, no querrá simplemente "esperar" que se generalice a nuevos casos. Quiere evaluarlo y ajustarlo si es necesario. Veamos cómo.

## Prueba y validación

La única forma de saber qué tan bien se generalizará un modelo a casos nuevos es probarlo en casos nuevos. Una forma de hacerlo es poner su modelo en producción y controlar qué tan bien funciona. Esto funciona bien, pero si su modelo es terriblemente malo, sus usuarios se quejarán, no es la mejor idea.

Una mejor opción es dividir sus datos en dos conjuntos: el *conjunto de entrenamiento* y el *equipo de prueba.* Como indican estos nombres, entrena su modelo con el conjunto de entrenamiento y lo prueba con el conjunto de prueba. La tasa de error en casos nuevos se denomina *error de generalización* o *error fuera de muestra),* y al evaluar su modelo en el conjunto de prueba, obtiene una estimación de este error. Este valor le indica qué tan bien funcionará su modelo en instancias que nunca antes había visto.

Si el error de entrenamiento es bajo (es decir, su modelo comete pocos errores en el conjunto de entrenamiento) pero el error de generalización es alto, significa que su modelo está sobreajustando los datos de entrenamiento.

Es común utilizar el 80% de los datos para entrenamiento y *resistir* 20% para pruebas.

Entonces, evaluar un modelo es bastante simple: solo use un conjunto de prueba. Ahora suponga que está dudando entre dos modelos (digamos un modelo lineal y un modelo polinomial): ¿cómo puede decidir? Una opción es entrenar a ambos y comparar qué tan bien generalizan usando el conjunto de prueba.

Ahora suponga que el modelo lineal se generaliza mejor, pero desea aplicar cierta regularización para evitar el sobreajuste. La pregunta es: ¿cómo se elige el valor del hiperparámetro de regularización? Una opción es entrenar 100 modelos diferentes usando 100 valores diferentes para este hiperparámetro. Suponga que encuentra el mejor valor de hiperparámetro que produce un modelo con el error de generalización más bajo, digamos solo un error del 5%.

Así que lanza este modelo a producción, pero desafortunadamente no funciona tan bien como se esperaba y produce un 15% de errores. ¿Lo que acaba de suceder?

El problema es que midió el error de generalización varias veces en el conjunto de prueba y adaptó el modelo y los hiperparámetros para producir el mejor modelo *para ese set*. Esto significa que es poco probable que el modelo funcione tan bien con datos nuevos.

Una solución común a este problema es tener un segundo conjunto de reserva llamado *conjunto de validación.* Entrena varios modelos con varios hiperparámetros utilizando el conjunto de entrenamiento, selecciona el modelo y los hiperparámetros que funcionan mejor en el conjunto de validación y, cuando está satisfecho con su modelo, ejecuta una única prueba final contra el conjunto de prueba para obtener una estimación de el error de generalización.

Para evitar "desperdiciar" demasiados datos de entrenamiento en conjuntos de validación, una técnica común es utilizar *validación cruzada:* el conjunto de entrenamiento se divide en subconjuntos complementarios y cada modelo se entrena con una combinación diferente de estos subconjuntos y se valida con las partes restantes. Una vez que se han seleccionado el tipo de modelo y los hiperparámetros, se entrena un modelo final usando estos hiperparámetros en el conjunto de entrenamiento completo, y el error generalizado se mide en el conjunto de prueba.

### Teorema sin almuerzo gratis

Un modelo es una versión simplificada de las observaciones. Las simplificaciones están destinadas a descartar los detalles superfluos que es poco probable que se generalicen a nuevas instancias. Sin embargo, para decidir qué datos descartar y qué datos conservar, debe hacer *suposiciones.* Por ejemplo, un modelo lineal supone que los datos son fundamentalmente lineales y que la distancia entre las instancias y la línea recta es solo ruido, que se puede ignorar con seguridad.

en un famoso papel de 1996 , [11] David Wolpert demostró que si no hace absolutamente ninguna suposición sobre los datos, entonces no hay razón para preferir un modelo sobre cualquier otro. Esto se llama *No hay almuerzo gratis (* Teorema de la NFL). Para algunos conjuntos de datos, lo mejor

11 “La falta de distinciones a priori entre algoritmos de aprendizaje”, D. Wolperts (1996).

modelo es un modelo lineal, mientras que para otros conjuntos de datos es una red neuronal. No hay modelo que sea *a priori* garantizado para funcionar mejor (de ahí el nombre del teorema). La única forma de saber con certeza qué modelo es el mejor es evaluarlos todos. Dado que esto no es posible, en la práctica usted hace algunas suposiciones razonables sobre los datos y evalúa solo algunos modelos razonables. Por ejemplo, para tareas simples puede evaluar modelos lineales con varios niveles de regularización, y para un problema complejo puede evaluar varias redes neuronales.

## Ejercicios

En este capítulo hemos cubierto algunos de los conceptos más importantes en Machine Learning. En los próximos capítulos profundizaremos y escribiremos más código, pero antes de hacerlo, asegúrese de saber cómo responder las siguientes preguntas:

1. ¿Cómo definiría el aprendizaje automático?
2. ¿Puedes nombrar cuatro tipos de problemas donde brilla?
3. ¿Qué es un conjunto de entrenamiento etiquetado?
4. ¿Cuáles son las dos tareas supervisadas más comunes?
5. ¿Puede nombrar cuatro tareas comunes sin supervisión?
6. ¿Qué tipo de algoritmo de aprendizaje automático usaría para permitir que un robot camine en varios terrenos desconocidos?
7. ¿Qué tipo de algoritmo utilizaría para segmentar a sus clientes en varios grupos?
8. ¿Enmarcaría el problema de la detección de spam como un problema de aprendizaje supervisado o un problema de aprendizaje no supervisado?
9. ¿Qué es un sistema de aprendizaje en línea?
10. ¿Qué es el aprendizaje fuera del núcleo?
11. ¿Qué tipo de algoritmo de aprendizaje se basa en una medida de similitud para hacer predicciones?
12. ¿Cuál es la diferencia entre un parámetro de modelo y el hiperparámetro de un algoritmo de aprendizaje?
13. ¿Qué buscan los algoritmos de aprendizaje basados en modelos? ¿Cuál es la estrategia más común que utilizan para tener éxito? ¿Cómo hacen predicciones?
14. ¿Puede nombrar cuatro de los principales desafíos del aprendizaje automático?
15. Si su modelo tiene un gran rendimiento en los datos de entrenamiento, pero se generaliza mal a instancias nuevas, ¿qué está sucediendo? ¿Puedes nombrar tres posibles soluciones?
16. ¿Qué es un equipo de prueba y por qué desearía utilizarlo?

17. ¿Cuál es el propósito de un conjunto de validación?

18. ¿Qué puede salir mal si ajusta los hiperparámetros con el conjunto de prueba?

19. ¿Qué es la validación cruzada y por qué la prefiere a un conjunto de validación?

Las soluciones a estos ejercicios están disponibles en Apéndice A .

CAPITULO 2

# Proyecto de aprendizaje automático de extremo a ext

En este capítulo, analizará un proyecto de ejemplo de principio a fin, pretendiendo ser un científico de datos contratado recientemente en una empresa inmobiliaria. [1] Estos son los pasos principales por los que pasará:

1. Mire el panorama general.
2. Obtenga los datos.
3. Descubra y visualice los datos para obtener información.
4. Prepare los datos para los algoritmos de Machine Learning.
5. Seleccione un modelo y entrénelo.
6. Ajuste su modelo.
7. Presenta tu solución.
8. Inicie, supervise y mantenga su sistema.

## Trabajar con datos reales

Cuando esté aprendiendo sobre Machine Learning, es mejor experimentar realmente con datos del mundo real, no solo con conjuntos de datos artificiales. Afortunadamente, hay miles de conjuntos de datos abiertos para elegir, que abarcan todo tipo de dominios. A continuación, se muestran algunos lugares en los que puede buscar para obtener datos:

- Repositorios de datos abiertos populares:

1 El proyecto de ejemplo es completamente ficticio; el objetivo es solo ilustrar los pasos principales de un proyecto de aprendizaje automático, no aprender nada sobre el negocio inmobiliario.

- Repositorio de aprendizaje automático de UC Irvine
- Conjuntos de datos de Kaggle
- Conjuntos de datos de AWS de Amazon

• Metaportales (enumeran repositorios de datos abiertos):

- *http://dataportals.org/*
- *http://opendatamonitor.eu/*
- *http://quandl.com/*

• Otras páginas que enumeran muchos repositorios de datos abiertos populares:

- Lista de Wikipedia de conjuntos de datos de aprendizaje automático
- Pregunta de Quora.com
- Subreddit de conjuntos de datos

En este capítulo, elegimos el conjunto de datos de precios de la vivienda de California del repositorio StatLib[2] ( ver Figura 2-1 ). Este conjunto de datos se basó en datos del censo de California de 1990. No es exactamente reciente (todavía podría pagar una bonita casa en el Área de la Bahía en ese momento), pero tiene muchas cualidades para aprender, así que pretendemos que son datos recientes. También agregamos un atributo categórico y eliminamos algunas características con fines didácticos.

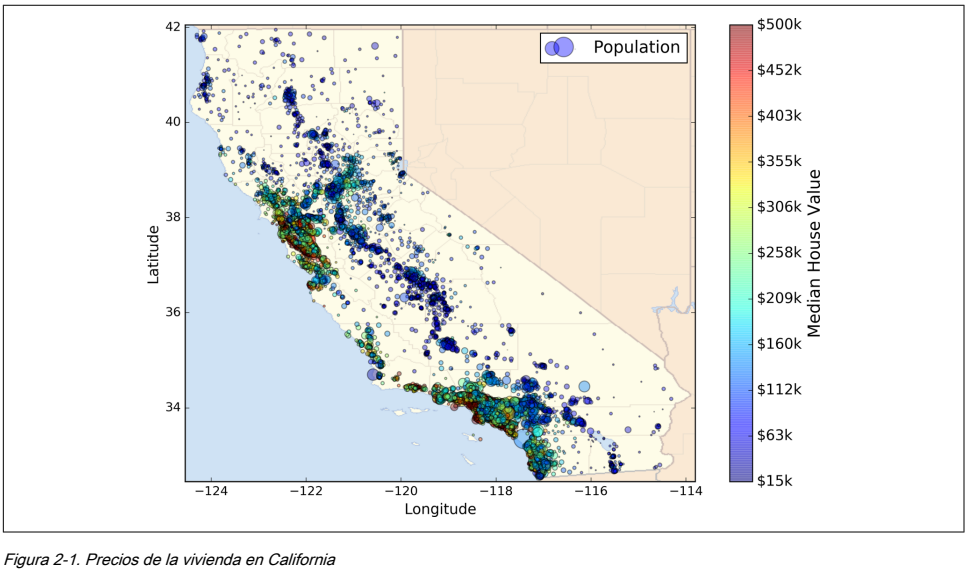


*Figura 2-1. Precios de la vivienda en California*

2 El conjunto de datos original apareció en R. Kelley Pace y Ronald Barry, "Sparse Spatial Autoregressions", *Estadísticas y letras de probabilidad* 33, no. 3 (1997): 291–297.

## Mira la imagen completa

¡Bienvenido a Machine Learning Housing Corporation! La primera tarea que se le pide que realice es crear un modelo de precios de vivienda en California utilizando los datos del censo de California. Estos datos tienen métricas como la población, el ingreso medio, el precio medio de la vivienda, etc. para cada grupo de bloques en California. Los grupos de bloques son la unidad geográfica más pequeña para la cual la Oficina del Censo de EE. UU. Publica datos de muestra (un grupo de bloques generalmente tiene una población de 600 a 3000 personas). Los llamaremos simplemente “distritos” para abreviar.

Su modelo debe aprender de estos datos y poder predecir el precio medio de la vivienda en cualquier distrito, dadas todas las demás métricas.

Como es un científico de datos bien organizado, lo primero que debe hacer es sacar la lista de verificación de su proyecto de aprendizaje automático. Puedes empezar con el de apéndice B ; debería funcionar razonablemente bien para la mayoría de los proyectos de aprendizaje automático, pero asegúrese de adaptarlo a sus necesidades. En este capítulo repasaremos muchos elementos de la lista de verificación, pero también omitiremos algunos, ya sea porque se explican por sí mismos o porque se discutirán en capítulos posteriores.

### Encuadre el problema

La primera pregunta que debe hacerle a su jefe es cuál es exactamente el objetivo comercial; La construcción de un modelo probablemente no sea el objetivo final. ¿Cómo espera la empresa utilizar y beneficiarse de este modelo? Esto es importante porque determinará cómo enmarca el problema, qué algoritmos seleccionará, qué medida de rendimiento utilizará para evaluar su modelo y cuánto esfuerzo debe dedicar a ajustarlo.

Su jefe responde que la salida de su modelo (una predicción del precio medio de la vivienda de un distrito) se enviará a otro sistema de aprendizaje automático (consulte Figura 2-2 ), junto con muchos otros *señales.* [3] Este sistema aguas abajo determinará si vale la pena invertir en un área determinada o no. Hacer esto bien es fundamental, ya que afecta directamente los ingresos.

3 Un dato que se envía a un sistema de aprendizaje automático se denomina *señal* en referencia a la teoría de la información de Shannon: desea una alta relación señal / ruido.

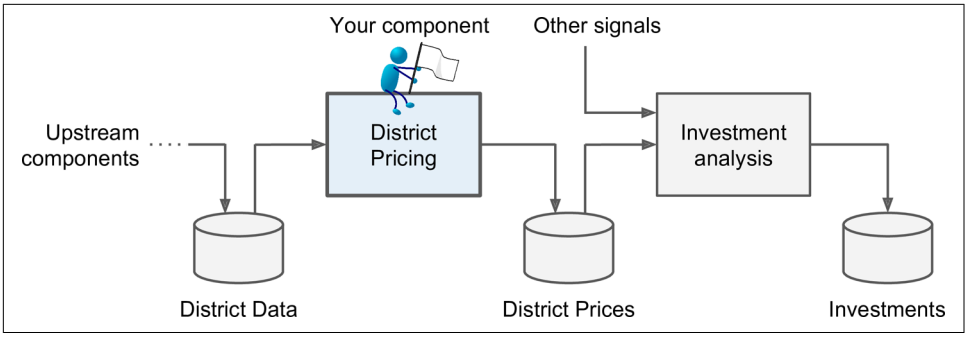


*Figura 2-2. Canalización de AMachine Learning para inversiones inmobiliarias*

### Oleoductos

Una secuencia de procesamiento de datos *componentes* se llama un dato *tubería.* Las canalizaciones son muy comunes en los sistemas de aprendizaje automático, ya que hay muchos datos que manipular y muchas transformaciones de datos que aplicar.

Los componentes normalmente se ejecutan de forma asincrónica. Cada componente extrae una gran cantidad de datos, los procesa y escupe el resultado en otro almacén de datos, y luego, algún tiempo después, el siguiente componente en la canalización extrae estos datos y escupe su propia salida, y así sucesivamente. Cada componente es bastante autónomo: la interfaz entre los componentes es simplemente el almacén de datos. Esto hace que el sistema sea bastante simple de comprender (con la ayuda de un gráfico de flujo de datos), y diferentes equipos pueden enfocarse en diferentes componentes. Además, si un componente se avería, los componentes posteriores a menudo pueden continuar funcionando normalmente (al menos durante un tiempo) simplemente utilizando la última salida del componente averiado. Esto hace que la arquitectura sea bastante robusta.

Por otro lado, un componente roto puede pasar desapercibido durante algún tiempo si no se implementa un monitoreo adecuado. Los datos se vuelven obsoletos y el rendimiento general del sistema disminuye.

La siguiente pregunta es cómo se ve la solución actual (si corresponde). A menudo le dará un rendimiento de referencia, así como información sobre cómo resolver el problema. Su jefe responde que los precios de la vivienda del distrito se estiman actualmente manualmente por expertos: un equipo recopila información actualizada sobre un distrito (excluyendo los precios medios de la vivienda) y utilizan reglas complejas para llegar a un estimado. Esto es costoso y requiere mucho tiempo, y sus estimaciones no son muy buenas; su tasa de error típica es de aproximadamente el 15%.

Bien, con toda esta información ya está listo para comenzar a diseñar su sistema. Primero, necesita enmarcar el problema: ¿es aprendizaje supervisado, no supervisado o de refuerzo? ¿Es una tarea de clasificación, una tarea de regresión o algo más? Debería

utiliza técnicas de aprendizaje por lotes o de aprendizaje en línea? Antes de seguir leyendo, haga una pausa e intente responder estas preguntas usted mismo.

¿Has encontrado las respuestas? Veamos: es claramente una tarea típica de aprendizaje supervisado ya que se le da *etiquetado* ejemplos de capacitación (cada instancia viene con el resultado esperado, es decir, el precio medio de la vivienda del distrito). Además, también es una tarea de regresión típica, ya que se le pide que prediga un valor. Más específicamente, esta es una *regresión multivariada* problema ya que el sistema usará múltiples características para hacer una predicción (usará la población del distrito, el ingreso medio, etc.). En el primer capítulo, pronosticó la satisfacción con la vida basándose en una sola característica, el PIB per cápita, por lo que fue un

*regresión univariante* problema. Finalmente, no hay un flujo continuo de datos que ingresan al sistema, no hay una necesidad particular de ajustarse a los datos cambiantes rápidamente, y los datos son lo suficientemente pequeños como para caber en la memoria, por lo que el aprendizaje por lotes simple debería funcionar bien.

Si los datos eran enormes, podría dividir su trabajo de aprendizaje por lotes en varios servidores (utilizando el *Mapa reducido* técnica, como veremos más adelante), o puede utilizar una técnica de aprendizaje en línea.

## Seleccione una medida de rendimiento

El siguiente paso es seleccionar una medida de desempeño. Una medida de rendimiento típica para los problemas de regresión es la raíz del error cuadrático medio (RMSE). Mide el *Desviación Estándar*[4] de los errores que el sistema comete en sus predicciones. Por ejemplo, un RMSE igual a 50.000 significa que aproximadamente el 68% de las predicciones del sistema caen dentro de los $ 50.000 del valor real y aproximadamente el 95% de las predicciones están dentro de los $ 100.000 del valor real.[5] Ecuación 2-1 muestra la fórmula matemática para calcular el RMSE.

*Ecuación 2-1. Error cuadrático medio (RMSE)*

$$\text{RMSE}, h\left(= 1 \sum\right) h\ yo - \sqrt{\sqrt{yo2}} \quad \text{metro} \quad \text{metro}\ yo = \left( \left( \left( \right) \right) \quad \left( \right) \right)$$

4 La desviación estándar, generalmente denotada como σ (la letra griega sigma), es la raíz cuadrada de la *diferencia,* que es el promedio de la desviación al cuadrado de la media.

5 Cuando una característica tiene forma de campana *distribución normal (* también llamado *Distribución gaussiana),* que es muy común, se aplica la regla "68-95-99.7": aproximadamente el 68% de los valores caen dentro de 1σ de la media, el 95% dentro de 2σ, y 99,7% dentro de 3σ.

## Notaciones

Esta ecuación presenta varias notaciones de aprendizaje automático muy comunes que usaremos a lo largo de este libro:

- *metro* es el número de instancias en el conjunto de datos en el que está midiendo el RMSE.

  - Por ejemplo, si está evaluando el RMSE en un conjunto de validación de 2000 distritos, entonces $m = 2.000$.

- $\mathbf{X}^{(i)}$ es un vector de todos los valores de características (excluyendo la etiqueta) de la $i^{th}$ instancia en el conjunto de datos, y $y^{(i)}$ es su etiqueta (el valor de salida deseado para esa instancia).

  - Por ejemplo, si el primer distrito en el conjunto de datos está ubicado en la longitud –118.29 °, latitud 33.91 °, y tiene 1416 habitantes con un ingreso medio de $ 38,372, y el valor medio de la vivienda es $ 156,400 (ignorando las otras características por ahora) , entonces:

$$\mathbf{X}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1,416 \\ 38,372 \end{pmatrix}$$

  y:

$$y^{(1)} = 156,400$$

- **X** es una matriz que contiene todos los valores de las características (excluidas las etiquetas) de todas las instancias del conjunto de datos. Hay una fila por instancia y el $i^{th}$ fila es igual a la transposición de $\mathbf{X}^{(i)}$, señaló ( $\mathbf{X}^{(i)}$ [6]

  - Por ejemplo, si el primer distrito es como se acaba de describir, entonces la matriz **X** Se ve como esto:

$$\mathbf{X} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^{T} \\ \left(\mathbf{x}^{(2)}\right)^{T} \\ \vdots \\ \left(\mathbf{x}^{(1999)}\right)^{T} \\ \left(\mathbf{x}^{(2000)}\right)^{T} \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

6 Recuerde que el operador de transposición invierte un vector de columna en un vector de fila (y viceversa).

- *h* es la función de predicción de su sistema, también llamada *hipótesis*. Cuando su sistema recibe el vector de características de una instancia $\mathbf{x}^{(i)}$, genera un valor predicho $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ para esa instancia ($\hat{y}$ se pronuncia "y-hat").
  - Por ejemplo, si su sistema predice que el precio medio de la vivienda en el primer distrito es $ 158,400, entonces $\hat{y}^{(1)} = h(\mathbf{x}^{(1)}) = 158.400$. El error de predicción para este distrito es $\hat{y}^{(1)} - y^{(1)} = 2.000$.

- RMSE($\mathbf{X}, h$) es la función de costo medida en el conjunto de ejemplos usando su hipótesis *h*.

Usamos fuente en cursiva minúscula para valores escalares (como *metro* o $y^{(i)}$) y nombres de funciones (como *h*), fuente en negrita minúscula para vectores (como $\mathbf{x}^{(i)}$) y fuente en negrita mayúscula para matrices (como $\mathbf{X}$).

Aunque el RMSE es generalmente la medida de rendimiento preferida para las tareas de regresión, en algunos contextos es posible que prefiera utilizar otra función. Por ejemplo, suponga que hay muchos distritos atípicos. En ese caso, puede considerar usar el *Error absoluto medio* (también llamado Desviación Absoluta Promedio; ver Ecuación 2-2):

*Ecuación 2-2. Error absoluto medio*

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h(\mathbf{x}^{(i)}) - y^{(i)}\right|$$

Tanto el RMSE como el MAE son formas de medir la distancia entre dos vectores: el vector de predicciones y el vector de valores objetivo. Varias medidas de distancia, o *normas* es posible:

- Calcular la raíz de una suma de cuadrados (RMSE) corresponde al *Euclidiano norma:* es la noción de distancia con la que está familiarizado. También se llama $\ell_2$ *norma,* célebre $\|\cdot\|_2$ (o solo $\|\cdot\|$).
- Calcular la suma de absolutos (MAE) corresponde al $\ell_1$ *norma,* célebre $\|\cdot\|_1$. A veces se le llama *Norma de Manhattan* porque mide la distancia entre dos puntos de una ciudad si solo puede viajar a lo largo de manzanas ortogonales.
- De manera más general, el $\ell_k$ *norma* de un vector **v** conteniendo *norte* elementos se define como $\|\mathbf{v}\|_k = (|v_0|^k + |v_1|^k + \cdots + |v_n|^k)^{1/k}$. $\ell_0$ solo da la cardinalidad del vector (es decir, el número de elementos), y $\ell_\infty$ da el valor absoluto máximo en el vector.
- Cuanto más alto es el índice de la norma, más se centra en los valores grandes y descuida los pequeños. Por eso, el RMSE es más sensible a los valores atípicos que el MAE. Pero cuando

los valores atípicos son exponencialmente raros (como en una curva en forma de campana), el RMSE funciona muy bien y generalmente se prefiere.

### Verifique las suposiciones

Por último, es una buena práctica enumerar y verificar las suposiciones que se hicieron hasta ahora (por usted u otros); esto puede detectar problemas graves desde el principio. Por ejemplo, los precios de distrito que genera su sistema se introducirán en un sistema de aprendizaje automático posterior, y asumimos que estos precios se utilizarán como tales. Pero, ¿qué pasa si el sistema descendente realmente convierte los precios en categorías (por ejemplo, "barato", "medio" o "caro") y luego usa esas categorías en lugar de los precios mismos? En este caso, conseguir el precio perfectamente correcto no es importante en absoluto; su sistema solo necesita elegir la categoría correcta. Si es así, entonces el problema debería haberse enmarcado como una tarea de clasificación, no como una tarea de regresión. No querrás descubrir esto después de trabajar en un sistema de regresión durante meses.

Afortunadamente, después de hablar con el equipo a cargo del sistema posterior, está seguro de que realmente necesitan los precios reales, no solo las categorías. ¡Excelente! Ya está listo, las luces están en verde y puede comenzar a codificar ahora.

## Obtener los datos

Es hora de ensuciarse las manos. No dude en tomar su computadora portátil y recorrer los siguientes ejemplos de código en una computadora portátil Jupyter. El cuaderno completo de Jupyter está disponible en *https://github.com/ageron/handson-ml* .

### Crear el espacio de trabajo

Primero necesitará tener Python instalado. Probablemente ya esté instalado en su sistema. Si no, puedes conseguirlo en *https://www.python.org/* .[7]

A continuación, debe crear un directorio de espacio de trabajo para su código y conjuntos de datos de Machine Learning. Abra una terminal y escriba los siguientes comandos (después de las indicaciones $):

```
$ export ML_PATH = "$ HOME / ml"          # Puedes cambiar la ruta si lo prefieres
$ mkdir -p $ ML_PATH
```

Necesitará varios módulos de Python: Jupyter, NumPy, Pandas, Matplotlib y Scikit-Learn. Si ya tiene Jupyter ejecutándose con todos estos módulos instalados, puede pasar a "Descargar los datos" en la página 43 . Si aún no los tiene, hay muchas formas de instalarlos (y sus dependencias). Puede usar el sistema de empaquetado de su sistema (por ejemplo, apt-get en Ubuntu, MacPorts o HomeBrew en

7 Se recomienda la última versión de Python 3. Python 2.7+ también debería funcionar bien, pero está obsoleto.

macOS), instale una distribución Scientific Python como Anaconda y use su sistema de empaquetado, o simplemente use el propio sistema de empaquetado de Python, pip, que se incluye por defecto con los instaladores binarios de Python (desde Python 2.7.9). [8] Puede verificar si pip está instalado escribiendo el siguiente comando:

```
$ pip3 --versión
pip 9.0.1 de [...] / lib / python3.5 / site-packages (python 3.5)
```

Debe asegurarse de tener instalada una versión reciente de pip, al menos> 1.4 para admitir la instalación del módulo binario (también conocido como ruedas). Para actualizar el módulo pip, escriba: [9]

```
$ pip3 install - actualizar pip Recopilar pip

[...]
Pip-9.0.1 instalado con éxito
```

## Crear un entorno aislado

Si desea trabajar en un entorno aislado (lo cual se recomienda encarecidamente para que pueda trabajar en diferentes proyectos sin tener versiones de biblioteca conflictivas), instale virtualenv ejecutando el siguiente comando pip:

```
$ pip3 install --user --upgrade virtualenv Collecting virtualenv

[...]
Virtualenv instalado con éxito
```

Ahora puede crear un entorno de Python aislado escribiendo:

```
$ cd $ ML_PATH
$ virtualenv env
Usando el prefijo base '[...]'
Nuevo ejecutable de python en [...] / ml / env / bin / python3.5 También
creando ejecutable en [...] / ml / env / bin / python Instalando setuptools, pip,
wheel ... hecho.
```

Ahora, cada vez que desee activar este entorno, simplemente abra una terminal y escriba:

```
$ cd $ ML_PATH
$ fuente env / bin / activar
```

Mientras el entorno esté activo, cualquier paquete que instale utilizando pip se instalará en este entorno aislado, y Python solo tendrá acceso a estos paquetes (si también desea acceder a los paquetes del sitio del sistema, debe crear el entorno

8 Mostraremos los pasos de instalación usando pip en un shell bash en un sistema Linux o macOS. Es posible que deba adaptar estos comandos a su propio sistema. En Windows, recomendamos instalar Anaconda en su lugar.

9 Puede que necesite tener derechos de administrador para ejecutar este comando; si es así, intente prefijarlo con sudo.

usando virtualenv's - paquetes-sitio-sistema opción). Consulte la documentación de virtualenv para obtener más información.

Ahora puede instalar todos los módulos necesarios y sus dependencias utilizando este sencillo comando pip:

```
$ pip3 install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn Recopilación de jupyter

  Descargando jupyter-1.0.0-py2.py3-none-any.whl
Recolectando matplotlib
  [...]
```

Para verificar su instalación, intente importar cada módulo como este:

```
$ python3 -c "importar jupyter, matplotlib, numpy, pandas, scipy, sklearn"
```

No debería haber salida ni error. Ahora puede iniciar Jupyter escribiendo:

```
$ jupyter cuaderno
[I 15:24 NotebookApp] Sirviendo cuadernos desde el directorio local: [...] / ml [I 15:24 NotebookApp] 0
kernels activos
[I 15:24 NotebookApp] El Jupyter Notebook se está ejecutando en: http: // localhost: 8888 / [I 15:24 NotebookApp]
Utilice Control-C para detener este servidor y apagar todos los núcleos (dos veces para omitir la confirmación).
```

Un servidor Jupyter ahora se está ejecutando en su terminal, escuchando el puerto 8888. Puede visitar este servidor abriendo su navegador web para *http: // localhost: 8888 / (* esto suele suceder automáticamente cuando se inicia el servidor). Debería ver su directorio de espacio de trabajo vacío (que contiene solo el *env* directorio si siguió las instrucciones virtualenv anteriores).

Ahora cree un nuevo cuaderno de Python haciendo clic en el botón Nuevo y seleccionando la versión apropiada de Python [10] ( ver Figura 2-3 ).

Esto hace tres cosas: primero, crea un nuevo archivo de cuaderno llamado *Untitled.ipynb* en su espacio de trabajo; segundo, inicia un kernel de Jupyter Python para ejecutar este cuaderno; y tercero, abre este cuaderno en una nueva pestaña. Debe comenzar cambiando el nombre de este cuaderno a "Vivienda" (esto automáticamente cambiará el nombre del archivo a *Vivienda.ipynb)* haciendo clic en Sin título y escribiendo el nuevo nombre.

10 Tenga en cuenta que Jupyter puede manejar múltiples versiones de Python, e incluso muchos otros lenguajes como R o Octava.

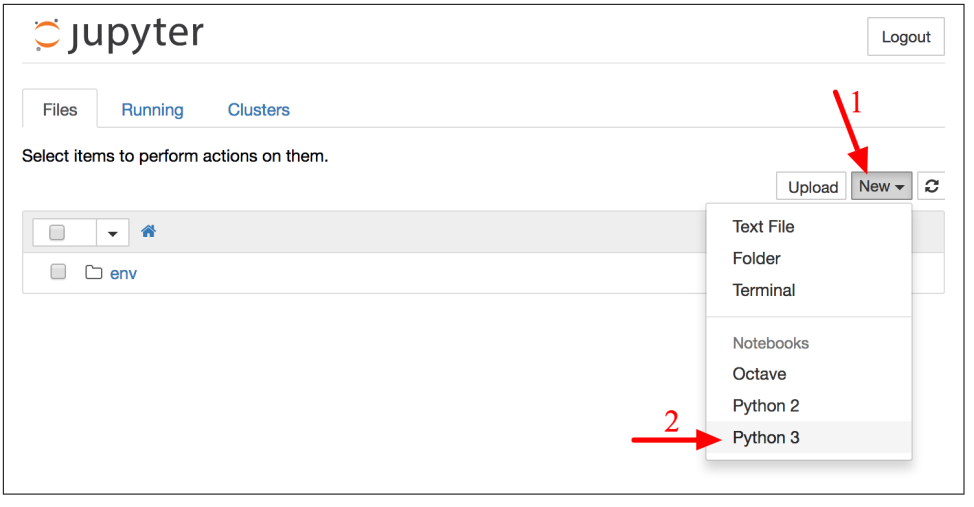


*Figura 2-3. Tu espacio de trabajo en Jupyter*

Un cuaderno contiene una lista de celdas. Cada celda puede contener código ejecutable o texto formateado. En este momento, el cuaderno contiene solo una celda de código vacía, etiquetada como "En [1]:". Intenta escribir **print ("¡Hola mundo!")** en la celda y haga clic en el botón de reproducción (ver Figura 2-4 ) o presione Shift-Enter. Esto envía la celda actual al kernel de Python de este cuaderno, que lo ejecuta y devuelve el resultado. El resultado se muestra debajo de la celda y, dado que llegamos al final del cuaderno, se crea automáticamente una nueva celda. Realice el recorrido por la interfaz de usuario del menú Ayuda de Jupyter para aprender los conceptos básicos.

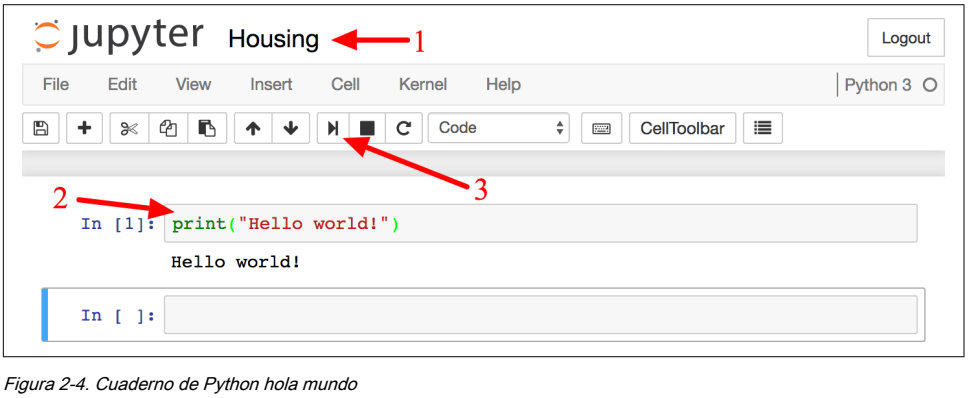


*Figura 2-4. Cuaderno de Python hola mundo*

## Descarga los datos

En entornos típicos, sus datos estarían disponibles en una base de datos relacional (o en algún otro almacén de datos común) y se distribuirían en varias tablas / documentos / archivos. A

acceder a él, primero debe obtener sus credenciales y autorizaciones de acceso,[11] y familiarícese con el esquema de datos. En este proyecto, sin embargo, las cosas son mucho más simples: solo descargará un solo archivo comprimido, *vivienda.tgz,* que contiene un archivo de valores separados por comas (CSV) llamado *housing.csv* con todos los datos.

Puede usar su navegador web para descargarlo y ejecutar tar xzf housing.tgz para descomprimir el archivo y extraer el archivo CSV, pero es preferible crear una pequeña función para hacerlo. Es útil en particular si los datos cambian con regularidad, ya que le permite escribir una pequeña secuencia de comandos que puede ejecutar siempre que necesite obtener los datos más recientes (o puede configurar un trabajo programado para que lo haga automáticamente a intervalos regulares). Automatizar el proceso de obtención de datos también es útil si necesita instalar el conjunto de datos en varias máquinas.

Aquí está la función para recuperar los datos:[12]

```
importar os
importar tarfile
de seis.movimientos importar urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = "conjuntos de datos / vivienda"
HOUSING_URL = DOWNLOAD_ROOT + HOUSING_PATH + "/housing.tgz"

def fetch_housing_data ( vivienda_url = HOUSING_URL , camino_de_vivienda = HOUSING_PATH ):
    Si no os . camino . isdir ( camino_de_vivienda ):
        os . makedirs ( camino_de_vivienda )
    tgz_path = os . camino . unirse ( camino_de_vivienda , "vivienda.tgz" )
    urllib . solicitud . urlretrieve ( vivienda_url , tgz_path )
    vivienda_tgz = tarfile . abierto ( tgz_path )
    vivienda_tgz . extraer todo ( camino = camino_de_vivienda )
    vivienda_tgz . cerrar ()
```

Ahora cuando llamas fetch_housing_data (), crea un *conjuntos de datos / vivienda* directorio en su espacio de trabajo, descarga el *vivienda.tgz* archivo, y extrae el *housing.csv* de él en este directorio.

Ahora carguemos los datos usando Pandas. Una vez más deberías escribir una pequeña función para cargar los datos:

```
importar pandas como pd

def load_housing_data ( camino_de_vivienda = HOUSING_PATH ):
    csv_path = os . camino . unirse ( camino_de_vivienda , "vivienda.csv" )
    regreso pd . read_csv ( csv_path )
```

11 Es posible que también deba verificar las restricciones legales, como los campos privados que nunca deben copiarse en almacenes de datos.

12 En un proyecto real, guardaría este código en un archivo Python, pero por ahora solo puede escribirlo en su Jupyter cuaderno.

Esta función devuelve un objeto Pandas DataFrame que contiene todos los datos.

## Eche un vistazo rápido a la estructura de datos

Echemos un vistazo a las cinco filas superiores usando el DataFrame cabeza() método (ver Figura 2-5 ).

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatio |
|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 |

*Figura 2-5. Las cinco primeras filas del conjunto de datos*

Cada fila representa un distrito. Hay 10 atributos (puede ver los primeros 6 en el captura de pantalla): longitud, latitud, vivienda_median_age, total_rooms, total_bed cuartos, población, hogares, median_inome, median_house_value, y ocean_proximity.

los info () El método es útil para obtener una descripción rápida de los datos, en particular el número total de filas y el tipo de atributo y el número de valores no nulos (ver Figura 2-6 ).

```
In [6]: housing.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

*Figura 2-6. Información de vivienda*

Hay 20,640 instancias en el conjunto de datos, lo que significa que es bastante pequeño para los estándares de Machine Learning, pero es perfecto para comenzar. Note que el total_bed habitaciones El atributo tiene solo 20,433 valores no nulos, lo que significa que 207 distritos no tienen esta característica. Tendremos que ocuparnos de esto más tarde.

Todos los atributos son numéricos, excepto el oceano_proximidad campo. Su tipo es objeto, por lo que podría contener cualquier tipo de objeto Python, pero como cargó estos datos desde un archivo CSV, sabe que debe ser un atributo de texto. Cuando miró las cinco filas superiores, probablemente notó que los valores en esa columna eran repetitivos, lo que significa que probablemente sea un atributo categórico. Puede averiguar qué categorías existen y cuántos distritos pertenecen a cada categoría utilizando el value_counts () método:

```
>>> alojamiento [ "ocean_proximity" ] . value_counts ()
<1H OCÉANO 9136
INTERIOR 6551
CERCA DEL OCÉANO 2658
CERCA DE LA BAHÍA 2290
ISLA 5
Nombre: ocean_proximity, dtype: int64
```

Veamos los otros campos. los describir() El método muestra un resumen de los atributos numéricos ( Figura 2-7 ).

```
In [8]: housing.describe()
```

Out[8]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrc |
|---|---|---|---|---|---|
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.0000 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 |
| **max** | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.00000 |

*Figura 2-7. Resumen de cada atributo numérico*

los contar, media, min, y max las filas se explican por sí mismas. Tenga en cuenta que los valores nulos se ignoran (por ejemplo, contar de total_drooms es 20.433, no 20.640). los std
La fila muestra la desviación estándar (que mide qué tan dispersos están los valores). Las filas de 25%, 50% y 75% muestran el correspondiente *percentiles:* un percentil indica el valor por debajo del cual cae un porcentaje dado de observaciones en un grupo de observaciones. Por ejemplo, el 25% de los distritos tienen un vivienda_median_age Más bajo que

18, mientras que el 50% son inferiores a 29 y el 75% son inferiores a 37. A menudo se denominan 25 th percentil (o 1 S t *cuartilla),* la mediana y el 75 th percentil (o 3 rd cuartilla).

Otra forma rápida de tener una idea del tipo de datos con los que está tratando es trazar un histograma para cada atributo numérico. Un histograma muestra el número de instancias (en el eje vertical) que tienen un rango de valores dado (en el eje horizontal). Puede graficar este atributo a la vez, o puede llamar al hist () método en todo el conjunto de datos, y trazará un histograma para cada atributo numérico (ver

Figura 2-8 ). Por ejemplo, puede ver que poco más de 800 distritos tienen un

median_house_value equivalente a unos $ 500.000.

```
% matplotlib en línea              # solo en un cuaderno de Jupyter
importar matplotlib.pyplot como plt
alojamiento . hist ( contenedores = 50 , figsize = ( 20 , 15 ))
plt . show ()
```

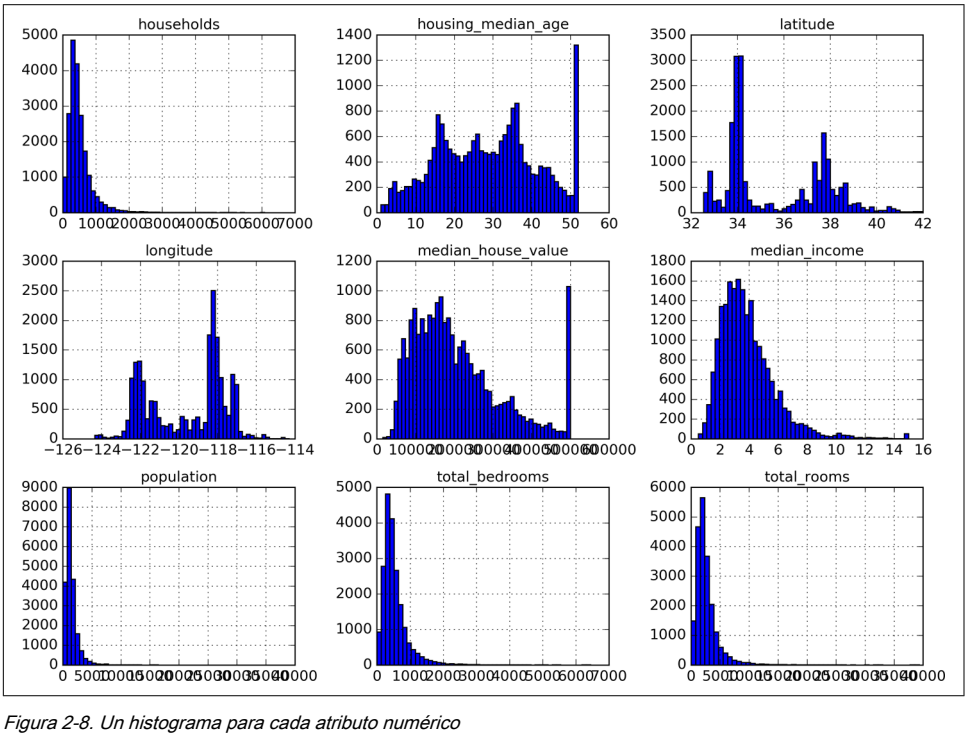


*Figura 2-8. Un histograma para cada atributo numérico*

los hist () El método se basa en Matplotlib, que a su vez se basa en un backend gráfico especificado por el usuario para dibujar en su pantalla. Entonces, antes de que pueda trazar algo, debe especificar qué backend debe usar Matplot- lib. La opción más simple es usar el comando mágico de Jupyter% matplotlib en línea. Esto le dice a Jupyter que configure Matplotlib para que use el propio backend de Jupyter. Luego, los gráficos se renderizan dentro del propio cuaderno. Tenga en cuenta que llamando show() es opcional en un cuaderno de Jupyter, ya que Jupyter mostrará automáticamente los gráficos cuando se ejecute una celda.

Observe algunas cosas en estos histogramas:

1. Primero, el atributo de ingreso mediano no parece estar expresado en dólares estadounidenses (USD). Después de consultar con el equipo que recopiló los datos, se le informa que los datos se han escalado y limitado a 15 (en realidad, 15.0001) para los ingresos medios más altos, y a 0,5 (en realidad, 0,4999) para los ingresos medios más bajos. Trabajar con atributos preprocesados es común en Machine Learning y no es necesariamente un problema, pero debe intentar comprender cómo se calcularon los datos.

2. También se limitaron la edad media de la vivienda y el valor medio de la vivienda. El último puede ser un problema serio ya que es su atributo de destino (sus etiquetas). Sus algoritmos de aprendizaje automático pueden aprender que los precios nunca superan ese límite. Debe consultar con su equipo cliente (el equipo que utilizará la salida de su sistema) para ver si esto es un problema o no. Si le dicen que necesitan predicciones precisas incluso más allá de $ 500,000, entonces tiene principalmente dos opciones:

   a. Reúna las etiquetas adecuadas para los distritos cuyas etiquetas fueron tapadas.

   segundo. Elimine esos distritos del conjunto de capacitación (y también del conjunto de prueba, ya que su sistema no debe evaluarse mal si predice valores superiores a $ 500 000).

3. Estos atributos tienen escalas muy diferentes. Discutiremos esto más adelante en este capítulo cuando exploremos el escalado de características.

4. Finalmente, muchos histogramas son *cola pesada:* se extienden mucho más a la derecha de la mediana que a la izquierda. Esto puede hacer que sea un poco más difícil para algunos algoritmos de aprendizaje automático detectar patrones. Intentaremos transformar estos atributos más adelante para tener más distribuciones en forma de campana.

Es de esperar que ahora comprenda mejor el tipo de datos que está tratando.

¡Espere! Antes de seguir analizando los datos, debe crear un conjunto de prueba, dejarlo a un lado y no mirarlo nunca.

## Crear un conjunto de prueba

Puede parecer extraño dejar de lado voluntariamente parte de los datos en esta etapa. Después de todo, solo ha echado un vistazo rápido a los datos, y seguramente debería aprender mucho más sobre ellos antes de decidir qué algoritmos usar, ¿verdad? Esto es cierto, pero su cerebro es un increíble sistema de detección de patrones, lo que significa que es muy propenso a sobreajustarse: si observa el conjunto de prueba, puede tropezar con algún patrón aparentemente interesante en los datos de la prueba que lo lleve a seleccionar un tipo particular de modelo de aprendizaje automático. Cuando calcule el error de generalización utilizando el conjunto de prueba, su estimación será demasiado optimista y lanzará un sistema que no funcionará tan bien como se esperaba. Se llama *espionaje de datos* parcialidad.

La creación de un conjunto de prueba es teóricamente bastante simple: simplemente elija algunas instancias al azar, generalmente el 20% del conjunto de datos, y déjelas a un lado:

```
importar numpy como notario público

def split_train_test ( datos , test_ratio ):
    índices_ barajados = notario público . aleatorio . permutación ( len ( datos ))
    test_set_size = En t ( len ( datos ) * test_ratio )
    test_indices = índices_ barajados [: test_set_size ]
    train_indices = índices_ barajados [ test_set_size :]
    regreso datos . iloc [ train_indices ], datos . iloc [ test_indices ]
```

Luego puede usar esta función de esta manera:

```
>>> juego de trenes , equipo de prueba = split_train_test ( alojamiento , 0,2 )
>>> impresión ( len ( juego de trenes ), "tren +" , len ( equipo de prueba ), "prueba" )
16512 tren + 4128 prueba
```

Bueno, esto funciona, pero no es perfecto: si ejecuta el programa nuevamente, ¡generará un conjunto de prueba diferente! Con el tiempo, usted (o sus algoritmos de aprendizaje automático) podrá ver el conjunto de datos completo, que es lo que desea evitar.

Una solución es guardar el conjunto de prueba en la primera ejecución y luego cargarlo en ejecuciones posteriores. Otra opción es establecer la semilla del generador de números aleatorios (por ejemplo, np.ran dom.seed (42)) [13] antes de llamar np.random.permutation (), para que siempre genere los mismos índices mezclados.

13 A menudo verá que las personas establecen la semilla aleatoria en 42. Este número no tiene ninguna propiedad especial, aparte de ser La respuesta a la pregunta fundamental de la vida, el universo y todo.

Pero ambas soluciones se romperán la próxima vez que obtenga un conjunto de datos actualizado. Una solución común es usar el identificador de cada instancia para decidir si debe ir o no en el conjunto de prueba (asumiendo que las instancias tienen un identificador único e inmutable). Por ejemplo, puede calcular un hash del identificador de cada instancia, conservar solo el último byte del hash y poner la instancia en el conjunto de prueba si este valor es menor o igual a 51 (~ 20% de 256). Esto asegura que el conjunto de prueba seguirá siendo consistente en múltiples ejecuciones, incluso si actualiza el conjunto de datos. El nuevo conjunto de prueba contendrá el 20% de las nuevas instancias, pero no contendrá ninguna instancia que estuviera anteriormente en el conjunto de entrenamiento. Aquí hay una posible implementación:

```
importar hashlib

def test_set_check ( identificador , test_ratio , picadillo ):
    regreso picadillo ( notario público . int64 ( identificador )) . digerir () [ - 1 ] < 256 * test_ratio

def split_train_test_by_id ( datos , test_ratio , id_column , picadillo = hashlib . md5 ):
    identificadores = datos [ id_column ]
    in_test_set = identificadores . aplicar ( lambda carné de identidad_ : test_set_check ( carné de identidad_ , test_ratio , picadillo ))
    regreso datos . loc [ ~ in_test_set ], datos . loc [ in_test_set ]
```

Desafortunadamente, el conjunto de datos de vivienda no tiene una columna de identificación. La solución más simple es usar el índice de fila como ID:

```
vivienda_con_id = alojamiento . reset_index ()                    # agrega una columna `índice`
juego de trenes , equipo de prueba = split_train_test_by_id ( vivienda_con_id , 0,2 , "índice" )
```

Si usa el índice de fila como un identificador único, debe asegurarse de que los datos nuevos se agreguen al final del conjunto de datos y que nunca se elimine ninguna fila. Si esto no es posible, puede intentar utilizar las funciones más estables para crear un identificador único. Por ejemplo, se garantiza que la latitud y la longitud de un distrito serán estables durante unos pocos millones de años, por lo que podría combinarlas en una identificación como esta: [14]

```
vivienda_con_id [ "carné de identidad" ] = alojamiento [ "longitud" ] * 1000 + alojamiento [ "latitud" ]
juego de trenes , equipo de prueba = split_train_test_by_id ( vivienda_con_id , 0,2 , "carné de identidad" )
```

Scikit-Learn proporciona algunas funciones para dividir conjuntos de datos en varios subconjuntos de varias formas. La función más simple es train_test_split, que hace prácticamente lo mismo que la función split_train_test definido anteriormente, con un par de características adicionales. Primero hay un estado_aleatorio parámetro que le permite establecer la semilla del generador aleatorio como se explicó anteriormente, y en segundo lugar, puede pasarle múltiples conjuntos de datos con un número idéntico de filas, y los dividirá en los mismos índices (esto es muy útil, por ejemplo, si tiene un DataFrame separado para etiquetas):

14 La información de ubicación es bastante burda y, como resultado, muchos distritos tendrán exactamente el mismo ID, por lo que terminarán en el mismo conjunto (prueba o entrenamiento). Esto introduce un desafortunado sesgo de muestreo.

```
de sklearn.model_selection importar train_test_split

juego de trenes , equipo de prueba = train_test_split ( alojamiento , test_size = 0,2 , estado_aleatorio = 42 )
```

Hasta ahora hemos considerado métodos de muestreo puramente aleatorios. Esto generalmente está bien si su conjunto de datos es lo suficientemente grande (especialmente en relación con el número de atributos), pero si no lo es, corre el riesgo de introducir un sesgo de muestreo significativo. Cuando una empresa de encuestas decide llamar a 1000 personas para hacerles algunas preguntas, no solo eligen
1.000 personas al azar en una cabina telefónica. Intentan asegurarse de que estas 1.000 personas sean representativas de toda la población. Por ejemplo, la población de EE. UU. Está compuesta por un 51,3% de mujeres y un 48,7% de hombres, por lo que una encuesta bien realizada en EE. UU. Trataría de mantener esta proporción en la muestra: 513 mujeres y 487 hombres. Se llama *muestreo estratificado:* la población se divide en subgrupos homogéneos denominados *Estratos,*

y se muestrea el número correcto de instancias de cada estrato para garantizar que el conjunto de prueba sea representativo de la población general. Si utilizaran un muestreo puramente aleatorio, habría aproximadamente un 12% de posibilidades de muestrear un conjunto de pruebas sesgado con menos del 49% de mujeres o más del 54% de mujeres. De cualquier manera, los resultados de la encuesta estarían significativamente sesgados.

Suponga que conversó con expertos que le dijeron que el ingreso medio es un atributo muy importante para predecir los precios medios de la vivienda. Es posible que desee asegurarse de que el conjunto de prueba sea representativo de las diversas categorías de ingresos en todo el conjunto de datos. Dado que el ingreso medio es un atributo numérico continuo, primero debe crear un atributo de categoría de ingresos. Veamos el histograma de ingresos medios más de cerca (ver Figura 2-9 ):

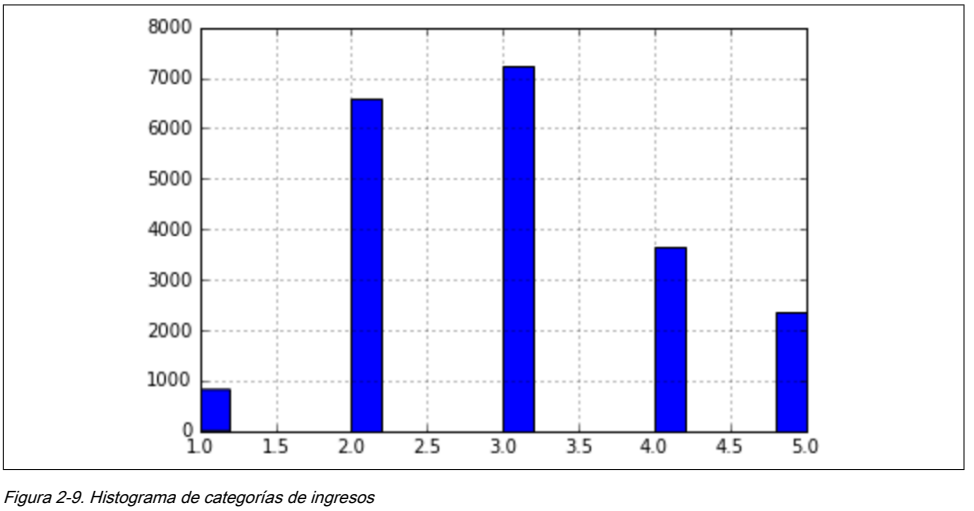


*Figura 2-9. Histograma de categorías de ingresos*

La mayoría de los valores de ingresos medios se agrupan alrededor de 2 a 5 (decenas de miles de dólares), pero algunos ingresos medios van mucho más allá de 6. Es importante tener un número suficiente

número de instancias en su conjunto de datos para cada estrato, o de lo contrario la estimación de la importancia del estrato puede estar sesgada. Esto significa que no debe tener demasiados estratos y cada estrato debe ser lo suficientemente grande. El siguiente código crea un atributo de categoría de ingresos dividiendo el ingreso medio por 1.5 (para limitar el número de categorías de ingresos) y redondeando hacia arriba usando el techo tener categorías discretas), y luego fusionar todas las categorías mayores de 5 en la categoría 5:

```
alojamiento [ "gato_ingreso" ] = notario público . hacer techo ( alojamiento [ "ingreso medio" ] / 1,5 )
alojamiento [ "gato_ingreso" ] . dónde ( alojamiento [ "gato_ingreso" ] < 5 , 5,0 , en su lugar = Cierto )
```

Ahora está listo para realizar un muestreo estratificado según la categoría de ingresos. Para ello, puede utilizar Scikit-Learn's EstratificadoShuffleSplit clase:

```
de sklearn.model_selection importar EstratificadoShuffleSplit

división = EstratificadoShuffleSplit ( n_splits = 1 , test_size = 0,2 , estado_aleatorio = 42 )
para train_index , test_index en división . división ( alojamiento , alojamiento [ "gato_ingreso" ]):
    strat_train_set = alojamiento . loc [ train_index ]
    strat_test_set = alojamiento . loc [ test_index ]
```

Veamos si esto funcionó como se esperaba. Puede comenzar observando las proporciones de las categorías de ingresos en el conjunto de datos de vivienda completo:

```
>>> alojamiento [ "gato_ingreso" ] . value_counts () / len ( alojamiento )
3,0 0.350581
2.0 0.318847
4.0 0.176308
5,0 0.114438
1.0 0.039826
Nombre: income_cat, dtype: float64
```

Con un código similar, puede medir las proporciones de la categoría de ingresos en el conjunto de prueba. Figura 2-10 compara las proporciones de las categorías de ingresos en el conjunto de datos general, en el conjunto de prueba generado con muestreo estratificado y en un conjunto de prueba generado mediante muestreo puramente aleatorio. Como puede ver, el conjunto de pruebas generado mediante muestreo estratificado tiene proporciones de categoría de ingresos casi idénticas a las del conjunto de datos completo, mientras que el conjunto de pruebas generado mediante muestreo puramente aleatorio está bastante sesgado.

| | Overall | Random | Stratified | Rand. %error | Strat. %error |
|---|---|---|---|---|---|
| **1.0** | 0.039826 | 0.040213 | 0.039738 | 0.973236 | -0.219137 |
| **2.0** | 0.318847 | 0.324370 | 0.318876 | 1.732260 | 0.009032 |
| **3.0** | 0.350581 | 0.358527 | 0.350618 | 2.266446 | 0.010408 |
| **4.0** | 0.176308 | 0.167393 | 0.176399 | -5.056334 | 0.051717 |
| **5.0** | 0.114438 | 0.109496 | 0.114369 | -4.318374 | -0.060464 |

*Figura 2-10. Comparación del sesgo de muestreo del muestreo estratificado versus el muestreo puramente aleatorio*

Ahora deberías quitar el gato_ingreso atributo para que los datos vuelvan a su estado original:

```
para conjunto en strat_train_set , strat_test_set ):
    conjunto . soltar ([ "gato_ingreso" ], eje = 1 , en su lugar = Cierto )
```

Dedicamos bastante tiempo a la generación de conjuntos de prueba por una buena razón: esta es una parte crítica a menudo descuidada de un proyecto de aprendizaje automático. Además, muchas de estas ideas serán útiles más adelante cuando analicemos la validación cruzada. Ahora es el momento de pasar a la siguiente etapa: explorar los datos.

# Descubra y visualice los datos para obtener conocimientos

Hasta ahora, solo ha echado un vistazo rápido a los datos para obtener una comprensión general del tipo de datos que está manipulando. Ahora el objetivo es profundizar un poco más.

Primero, asegúrese de haber dejado a un lado la prueba reservada y de que solo está explorando el conjunto de entrenamiento. Además, si el conjunto de entrenamiento es muy grande, es posible que desee probar un conjunto de exploración para que las manipulaciones sean fáciles y rápidas. En nuestro caso, el conjunto es bastante pequeño, por lo que puede trabajar directamente en el conjunto completo. Creemos una copia para que puedas jugar con ella sin dañar el conjunto de entrenamiento:

```
alojamiento = strat_train_set . Copiar ()
```

## Visualización de datos geográficos

Dado que existe información geográfica (latitud y longitud), es una buena idea crear un diagrama de dispersión de todos los distritos para visualizar los datos ( Figura 2-11 ):

```
alojamiento . trama ( tipo = "dispersión" , X = "longitud" , y = "latitud" )
```

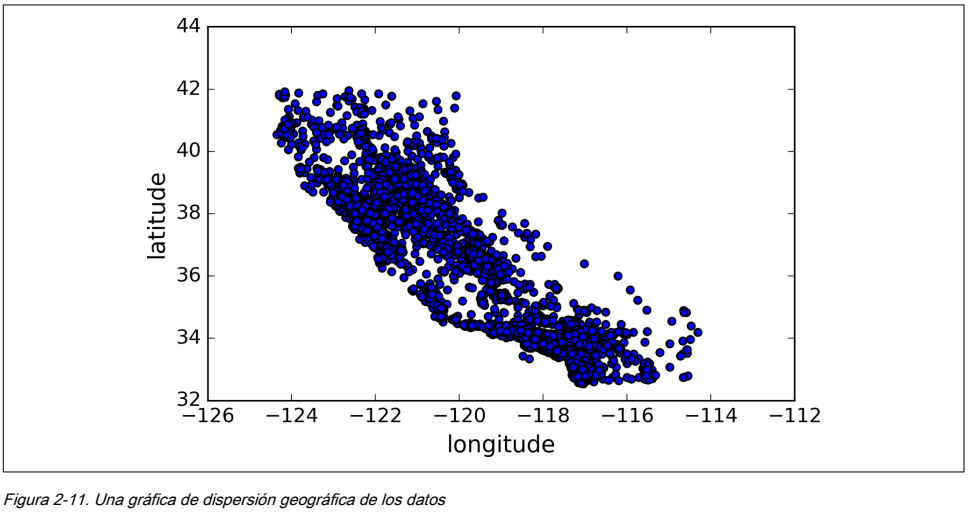


*Figura 2-11. Una gráfica de dispersión geográfica de los datos*