# Marsland

**Outline breve (págs. 36 – 47)**

- **Terminología esencial de ML**
    - Definición de **entrada** (vector **x**), **salida** (vector **y**), **objetivo** (vector **t**) y **pesos** (**W**) de un modelo. Introduce notación matricial y vectorial usada en todo el libro.
- **Maldición de la dimensionalidad (2.1.2)**
    - Analiza el volumen decreciente de la hiperesfera unitaria al crecer el número de dimensiones y sus implicaciones: se necesitan muchos más datos para cubrir el espacio y evitar modelos frágiles. Se muestran figuras 2.2 – 2.4 con la caída de volumen y ejemplos intuitivos.
- **Necesidad de pruebas: "Saber lo que sabes" (2.2)**
    - Motivación para separar datos en *train* y *test*; se subraya que medir error solo en entrenamiento no informa sobre generalización.
- **Sobreajuste y conjuntos de validación (2.2.1 – 2.2.2)**
    - Ilustra con la Fig. 2.5 cómo un modelo puede memorizar ruido; introduce el esquema *train / validation / test* y convenciones típicas (50-25-25 % o 60-20-20 %).
- **Validación cruzada (k-fold, leave-one-out)**
    - Describe K-fold CV y su costo computacional: entrenar K modelos y seleccionar el de menor error de validación. Fig. 2.7 ejemplifica el procedimiento.
- **Matriz de confusión y partición aleatoria**
    - Expone cómo construir la matriz para evaluar clasificadores multiclase y advierte sobre sesgos si los datos no se barajan antes de dividirlos (Fig. 2.6).
- **Métricas derivadas**
    - Presenta *accuracy*, sensibilidad, especificidad, precisión, *recall* y la medida F1 como combinaciones clave para clasificación binaria (Ecuaciones 2.2 – 2.7).
- **Curva ROC y AUC (2.2.5)**
    - Define ROC (TPR vs. FPR), explica el significado de la diagonal (azar) y el uso del área bajo la curva para comparar clasificadores (Fig. 2.8).

---

# Geron

**Outline breve (págs. 177 – 202)**

- **Capítulo 3 – Clasificación**
    - Arranque del capítulo con el caso guía MNIST: 70 000 imágenes 28 × 28 de dígitos manuscritos.
    - Uso de `fetch_openml()` para descargar el dataset y separación explícita en 60 000 train / 10 000 test.
- **Primer modelo: clasificador binario "es-5 / no-5"**
    - Construcción de etiquetas booleanas `y_train_5`, `y_test_5`.
    - Entrenamiento de un `SGDClassifier` (descenso de gradiente estocástico) sobre todo el set de entrenamiento.
- **Evaluación inicial y sesgo de clases**
    - `cross_val_score()` con k = 3 muestra ≈ 95 % accuracy, pero se introduce un `DummyClassifier` que predice siempre "no-5" y alcanza > 90 %, evidenciando que la precisión bruta es engañosa en clases desbalanceadas.
- **Validación cruzada manual**
    - Ejemplo con `StratifiedKFold` y `clone()` para ilustrar el procedimiento completo y la importancia del muestreo estratificado.
- **Matriz de confusión y métricas derivadas**
    - Obtención de predicciones "fuera de muestra" con `cross_val_predict()`.
    - Cálculo e interpretación de la confusion matrix (TP, FP, FN, TN).
    - Definición y cálculo de **precisión**, **recall** y **F1-score**; discusión de sus usos y limitaciones.
- **Trade-off precisión ↔ recall mediante el umbral. (recall == 'recuperación')**
    - Acceso a `decision_function()`, obtención de `y_scores` y trazado de **precision-recall curve** con `precision_recall_curve()`.
    - Selección de un umbral para alcanzar ≥ 90 % precisión y su impacto: recall ≈ 48 %.
- **Curva ROC y AUC**
    - Generación con `roc_curve()`, explicación de TPR vs FPR, comparación con clasificador aleatorio.

- Cálculo de **ROC-AUC** (≈ 0.96) y relación con la elección de la métrica según el desbalance de clases.

Este tramo introduce la clasificación binaria paso a paso: carga de datos, primer modelo, validación rigurosa y elección de métricas adecuadas (precisión/recall, F1, PR curve, ROC/AUC) antes de comparar con un segundo algoritmo.

---

# Muller - Guido

**Outline breve (págs. 39-63)**

- **Aprendizaje supervisado:** definición general y distinción entre clasificación y regresión, con ejemplos canónicos (spam/no-spam, predicción de ingresos).
- **Generalización, sobreajuste y subajuste:** explicación del equilibrio deseado, figura 2-1 y discusión de cómo la complejidad del modelo debe ajustarse al tamaño y variedad del conjunto de datos.
- **Efecto del tamaño del dataset:** por qué más datos permiten modelos más complejos y cuándo esa recolección adicional sí aporta valor.
- **Panorama de algoritmos supervisados:** criterios para entender cómo aprenden los modelos, sus fortalezas-debilidades y los parámetros clave que controlan complejidad.

k-Nearest Neighbors (k-NN)

1. **Clasificación con k-NN:**
    - Ejemplo sintético *Forge* y visualización del límite de decisión (Fig. 2-4, 2-5).
    - Influencia de *k*: fronteras abruptas para *k* = 1, suavizado progresivo al aumentar *k* (Fig. 2-6) y medición de precisión en el conjunto *Breast Cancer* para k = 1-10.
2. **Regresión con k-NN:**
    - Ejemplo *Wave*, contraste entre *k* = 1, 3, 9 y efecto sobre la curva de predicción (Fig. 2-8, 2-9).

Modelos lineales (sección inicial)

- **Regresión lineal (Mínimos Cuadrados Ordinarios):** planteo matemático, objetivo de minimizar el error cuadrático medio y ausencia de hiper-parámetros de regularización en la forma básica. (‘cresta’ == ridge, no entra)
- **Datasets de demostración:** mención al *Boston Housing* y su versión extendida para ilustrar la utilidad de la ingeniería de características en modelos lineales.

# Alppaydin

**Outline breve (págs. 41 – 52)**

1.1 ¿Qué es el aprendizaje automático?

- Programar computadoras para optimizar su desempeño en tareas cambiantes mediante modelos estadísticos en lugar de reglas fijas; intimidad con IA y minería de datos.

1.2 Ejemplos de aplicaciones

1. **Clasificación (supervisado):**
    - **Scoring crediticio** con discriminante en el plano ingresos-ahorros (Fig 1.1) para decidir riesgo de préstamo.
    - **Reconocimiento de patrones:** OCR, rostros y diagnóstico médico; destaca la necesidad de modelar variaciones (rotación, pose, síntomas faltantes).
    - **Voz y biometría:** dependencia temporal y combinación de múltiples modalidades para autenticación precisa.
2. **Regresión (supervisado):**
    - **Precio de autos usados** vs. kilometraje (Fig 1.2); función lineal $y = wx + w_0$ y posible extensión a modelos no lineales.
    - **Control de robots / vehículos autónomos:** salida continua = ángulo de giro; se entrena a partir de trayectorias humanas.
    - **Detección de valores atípicos** como derivado de una regla de regresión (fraude).
3. **Aprendizaje no supervisado:**
    - **Estimación de densidad** y **clustering** para segmentación de clientes, identificación de nichos y gestión de relaciones.
    - **Compresión de imágenes** via agrupamiento de píxeles RGB y reducción de redundancia en bitmaps de texto escaneado.
    - **Bioinformática:** clustering para descubrir motivos en secuencias de ADN/proteínas.
    - **Diseño de experimentos / superficie de respuesta:** ejemplo de optimizar la calidad del café ajustando modelos de regresión iterativos sobre las variables de tueste.

# Preliminares

Este capítulo tiene dos propósitos: presentar algunos de los conceptos fundamentales del aprendizaje automático y analizar cómo surgen en él algunas de las ideas básicas del procesamiento de datos y la estadística. Una de las maneras más útiles de desglosar los efectos del aprendizaje, que consiste en expresarlo en términos de los conceptos estadísticos de sesgo y varianza, se presenta en la Sección 2.5, tras una sección donde se introducen dichos conceptos para principiantes.

## 2.1 ALGUNAS TERMINOLOGÍAS

Comenzaremos considerando la terminología que utilizaremos a lo largo del libro; ya hemos visto algunos detalles en la Introducción. Hablaremos de las entradas y los vectores de entrada para nuestros algoritmos de aprendizaje. Asimismo, hablaremos de las salidas del algoritmo. Las entradas son los datos que se introducen en el algoritmo. En general, todos los algoritmos de aprendizaje automático funcionan tomando un conjunto de valores de entrada, generando una salida (respuesta) para ese vector de entrada y luego pasando a la siguiente entrada. El vector de entrada suele estar compuesto por varios números reales, por lo que se describe como un vector: se escribe como una serie de números, por ejemplo, (0.2, 0.45, 0.75, −0.3). El tamaño de este vector, es decir, el número de elementos que lo componen, se denomina dimensionalidad de la entrada. Esto se debe a que, si graficamos el vector como un punto, necesitaríamos una dimensión de espacio para cada uno de los diferentes elementos del vector, de modo que el ejemplo anterior tiene 4 dimensiones. Hablaremos más de esto en la Sección 2.1.1.

A menudo escribiremos ecuaciones en notación vectorial y matricial, usando minúsculas en negrita para vectores y mayúsculas en negrita para matrices. Un vector x tiene elementos (x1, x2, ..., xm). En el libro, usaremos la siguiente notación:

Entradas Un vector de entrada son los datos proporcionados como una entrada al algoritmo. Escrito como x, con elementos $x_i$ , donde i va desde 1 hasta el número de dimensiones de entrada, m.

Los pesos wij son las conexiones ponderadas entre los nodos i y j. En las redes neuronales, estos pesos son análogos a las sinapsis cerebrales. Se organizan en una matriz W.

Salidas: El vector de salida es y, con elementos $y_j$ , donde j va de 1 a n, el número de dimensiones de salida . Podemos escribir y(x,W) para recordar que la salida depende de las entradas del algoritmo y del conjunto actual de pesos de la red.

Objetivos El vector objetivo t, con elementos $t_j$ , donde j va desde 1 hasta el número de dimensiones de salida, n, son los datos adicionales que necesitamos para el aprendizaje supervisado, ya que proporcionan las respuestas "correctas" que el algoritmo está aprendiendo.

Función de activación Para las redes neuronales, g(·) es una función matemática que describe la activación de la neurona como respuesta a las entradas ponderadas, como la función de umbral descrita en la Sección 3.1.2.

Error E, función que calcula las imprecisiones de la red en función de las salidas y y los objetivos t.

### 2.1.1 Espacio de peso

Al trabajar con datos, suele ser útil poder representarlos gráficamente y visualizarlos. Si nuestros datos solo tienen dos o tres dimensiones de entrada, esto es bastante sencillo: usamos el eje x para la característica 1, el eje y para la característica 2 y el eje z para la característica 3. A continuación, representamos gráficamente las posiciones de los vectores de entrada en estos ejes. Esto se puede extender a tantas dimensiones como queramos, siempre que no queramos visualizarlos en nuestro mundo 3D. Incluso si tenemos 200 dimensiones de entrada (es decir, 200 elementos en cada uno de nuestros vectores de entrada), podemos intentar imaginarlos representados gráficamente utilizando 200 ejes ortogonales entre sí (es decir, perpendiculares entre sí). Una de las ventajas de las computadoras es que no están limitadas como nosotros: pídele a una computadora que contenga una matriz de 200 dimensiones y lo hará.
Siempre que el algoritmo sea correcto (¡siempre la parte difícil!), la computadora no sabe que 200 dimensiones son más difíciles que 2 para nosotros, los humanos.

Podemos observar proyecciones de los datos en nuestro mundo 3D al comparar gráficamente solo tres de las características, pero esto suele ser bastante confuso: las cosas pueden parecer muy próximas en los tres ejes elegidos, pero pueden estar muy separadas en el conjunto completo. Ya has experimentado esto en tu vista 2D del mundo 3D; la Figura 1.2 muestra dos vistas diferentes de algunos aerogeneradores. Los dos aerogeneradores parecen estar muy próximos desde un ángulo, pero están claramente separados desde otro.

Además de graficar puntos de datos, también podemos graficar cualquier otra cosa que queramos. En particular, podemos graficar algunos de los parámetros de un algoritmo de aprendizaje automático. Esto es especialmente útil para redes neuronales (que veremos en el siguiente capítulo), ya que los parámetros de una red neuronal son los valores de un conjunto de pesos que conectan las neuronas con las entradas. A la izquierda de la Figura 2.1 se muestra un esquema de una red neuronal, que muestra las entradas a la izquierda y las neuronas a la derecha. Si tratamos los pesos que se introducen en una de las neuronas como un conjunto de coordenadas en lo que se conoce como espacio de pesos, podemos graficarlos. Pensamos en los pesos que se conectan a una neurona en particular y graficamos las fortalezas de los pesos utilizando un eje para cada peso que llega a la neurona y graficando la posición de la neurona como la ubicación, utilizando el valor de w1 como la posición en el primer eje, el valor de w2 en el segundo eje, etc. Esto se muestra a la derecha de la Figura 2.1.

Ahora que tenemos un espacio en el que podemos hablar sobre la proximidad entre neuronas y entradas, podemos imaginar la ubicación de neuronas y entradas en el mismo espacio al representar gráficamente la posición de cada neurona como la ubicación donde sus pesos indican que debería estar. Ambos espacios tendrán la misma dimensión (siempre que no usemos un nodo de sesgo (véase la Sección 3.3.2); de lo contrario, el espacio de pesos tendrá una dimensión adicional), por lo que podemos representar gráficamente la posición de las neuronas en el espacio de entrada. Esto nos ofrece una forma diferente de aprender, ya que al cambiar los pesos, cambiamos la ubicación de las neuronas en este espacio. Podemos medir las distancias entre las entradas y las neuronas calculando la distancia euclidiana, que en dos dimensiones se puede escribir como:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{2.1}$$

Así que podemos usar la idea de que las neuronas y las entradas están "cerca unas de otras" para decidir.

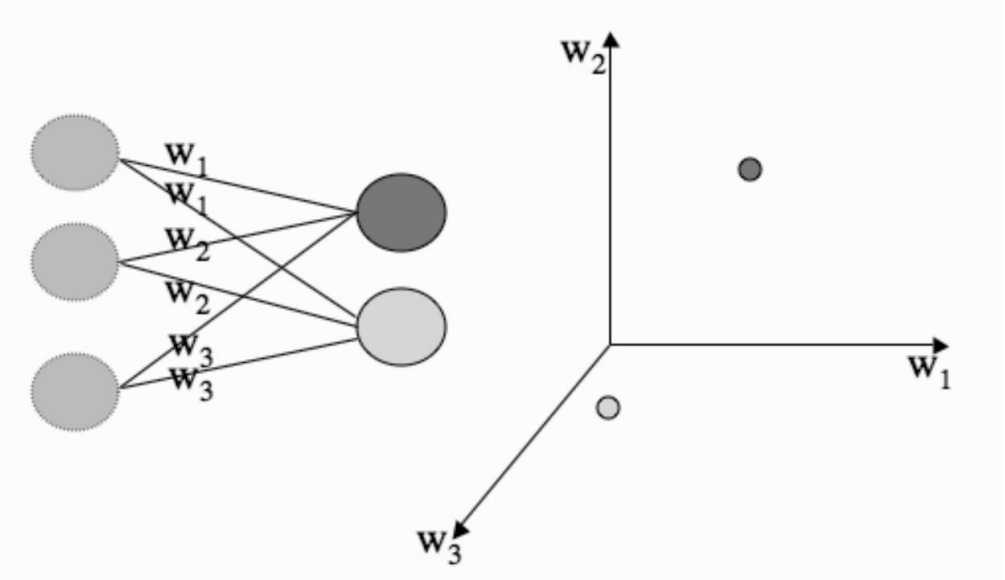


FIGURA 2.1 Posición de dos neuronas en el espacio de ponderaciones. Las etiquetas en la red se refieren a la dimensión en la que se representa dicha ponderación, no a su valor.

Cuándo debería activarse una neurona y cuándo no. Si la neurona está cerca de la entrada en este sentido, debería activarse, y si no está cerca, no debería. Esta imagen del espacio de ponderaciones puede ser útil para comprender otro concepto importante en el aprendizaje automático: el efecto que puede tener el número de dimensiones de entrada. El vector de entrada nos dice todo lo que sabemos sobre ese ejemplo, y normalmente no sabemos lo suficiente sobre los datos para saber qué es útil y qué no (recuerde el ejemplo de clasificación de monedas en la Sección 1.4.2), por lo que podría parecer sensato incluir toda la información posible y dejar que el algoritmo determine por sí mismo lo que necesita. Desafortunadamente, estamos a punto de ver que hacer esto tiene un coste significativo.

### 2.1.2 La maldición de la dimensionalidad

La maldición de la dimensionalidad es un nombre muy fuerte, por lo que probablemente puedas adivinar que es un problema. La esencia de la maldición reside en la comprensión de que, a medida que aumenta el número de dimensiones, el volumen de la hiperesfera unitaria no aumenta con él. La hiperesfera unitaria es la región que obtenemos si comenzamos en el origen (el centro de nuestro sistema de coordenadas) y dibujamos todos los puntos a una distancia de 1 del origen. En dos dimensiones, obtenemos un círculo de radio 1 alrededor de (0, 0) (dibujado en la Figura 2.2), y en tres dimensiones, una esfera alrededor de (0, 0, 0) (Figura 2.3). En dimensiones superiores, la esfera se convierte en una hiperesfera. La siguiente tabla muestra el tamaño de la hiperesfera unitaria para las primeras dimensiones, y el gráfico de la Figura 2.4 muestra lo mismo, pero también muestra claramente que, a medida que el número de dimensiones tiende a infinito, el volumen de la hiperesfera tiende a cero.

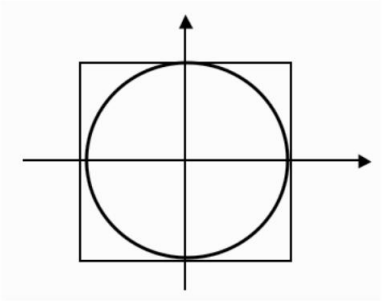

FIGURA 2.2 El círculo unitario en 2D con su cuadro delimitador.

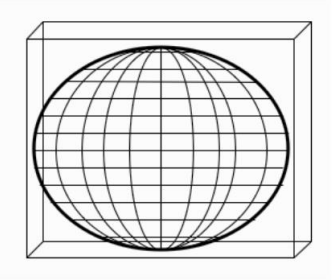

FIGURA 2.3 La esfera unitaria en 3D con su cubo delimitador. La esfera no llegar hasta las esquinas hasta el círculo lo hace, y esto se hace más notorio a medida que El número de dimensiones aumenta.

| Volumen de dimensión | |
|---|---|
| 1 2.0000 | |
| 2 3.1416 | |
| 3 4.1888 | |
| 4 4.9348 | |
| 5 5.2636 | |
| 6 5.1677 | |
| 7 4.7248 | |
| 8 4.0587 | |
| 9 3.2985 | |
| 10 2.5502 | |

A primera vista, esto parece completamente contraintuitivo. Sin embargo, piense en encerrar el hiperesfera en una caja de ancho 2 (entre -1 y 1 a lo largo de cada eje), de modo que la caja toca los lados de la hiperesfera. Para el círculo, casi toda el área dentro de la caja es incluido en el círculo, excepto un poquito en cada esquina (ver Figura 2.2) Lo mismo es cierto en 3D (Figura 2.3), pero si pensamos en la hiperesfera de 100 dimensiones (no necesariamente algo que quieras imaginar) y sigue la línea diagonal desde el origen hasta uno de las esquinas de la caja, entonces intersectamos el límite de la hiperesfera cuando todas las Las coordenadas son 0.1. El 90% restante de la línea dentro del cuadro está fuera de la hiperesfera. Y entonces el volumen de la hiperesfera obviamente se está reduciendo a medida que aumenta el número de dimensiones. crece. El gráfico de la Figura 2.4 muestra que cuando el número de dimensiones es superior a aproximadamente 20, el volumen es prácticamente cero. Se calculó utilizando la fórmula para el volumen de la Hiperesfera de dimensión n cuando $v_n = (2\pi/n)v_{n-2}$. Por lo tanto, tan pronto como $n > 2\pi$, el volumen comienza encogerse.

La maldición de la dimensionalidad se aplicará a nuestros algoritmos de aprendizaje automático porque, como Cuanto mayor sea el número de dimensiones de entrada, necesitaremos más datos para habilitar el algoritmo. para generalizar suficientemente bien. Nuestros algoritmos intentan separar los datos en clases según la características; por lo tanto, a medida que aumenta el número de características, también aumentará el número de puntos de datos que necesidad. Por esta razón, a menudo tendremos que ser cuidadosos con la información que damos a el algoritmo, lo que significa que necesitamos entender algo sobre los datos de antemano.

Independientemente de cuántas dimensiones de entrada haya, el objetivo del aprendizaje automático es

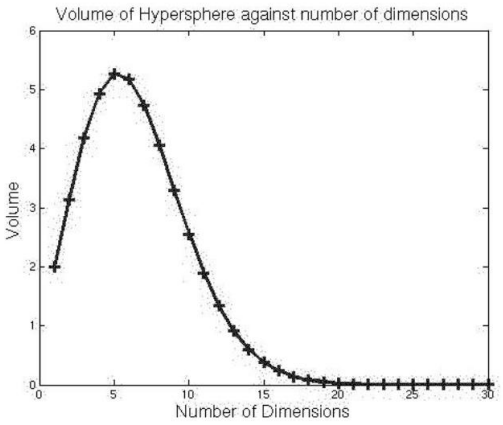


FIGURA 2.4 El volumen de la hiperesfera unitaria para diferentes números de dimensiones.

Realizar predicciones a partir de los datos de entrada. En la siguiente sección, consideraremos cómo evaluar la eficacia con la que un algoritmo logra esto.

## 2.2 SABER LO QUE SABES: PRUEBAS DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO RITMOS

El propósito del aprendizaje es mejorar la predicción de los resultados, ya sean etiquetas de clase o valores de regresión continua. La única manera de saber con certeza el éxito del aprendizaje del algoritmo es comparar las predicciones con etiquetas objetivo conocidas, que es como se realiza el entrenamiento para el aprendizaje supervisado. Esto sugiere que una cosa que se puede hacer es simplemente observar el error que comete el algoritmo en el conjunto de entrenamiento.

Sin embargo, queremos que los algoritmos se generalicen a ejemplos no observados en el conjunto de entrenamiento, y obviamente no podemos probar esto usando dicho conjunto. Por lo tanto, necesitamos datos diferentes, un conjunto de prueba, para probarlo también. Usamos este conjunto de prueba de pares (entrada, objetivo) introduciéndolos en la red y comparando la salida predicha con el objetivo, pero no modificamos sus pesos ni otros parámetros: los usamos para determinar el nivel de aprendizaje del algoritmo. El único problema es que reduce la cantidad de datos disponibles para el entrenamiento, pero es algo con lo que tendremos que lidiar.

### 2.2.1 Sobreajuste

Desafortunadamente, las cosas son un poco más complicadas que eso, ya que también podríamos querer saber qué tan bien se generaliza el algoritmo a medida que aprende: necesitamos asegurarnos de realizar suficiente entrenamiento para que el algoritmo generalice bien. De hecho, existe al menos tanto peligro en el sobreentrenamiento como en el subentrenamiento. El número de grados de variabilidad en la mayoría de los algoritmos de aprendizaje automático es enorme: para una red neuronal hay muchos pesos, y cada uno de ellos puede variar. Sin duda, esto es más variación que la que hay en la función que estamos aprendiendo, así que debemos ser cuidadosos: si entrenamos durante demasiado tiempo, sobreajustaremos los datos, lo que significa que habremos aprendido sobre el ruido y las imprecisiones en los datos, así como en la función real. Por lo tanto, el modelo que aprendamos será demasiado complejo y no podrá generalizar.

La figura 2.5 muestra esto al trazar las predicciones de algún algoritmo (como la curva) en

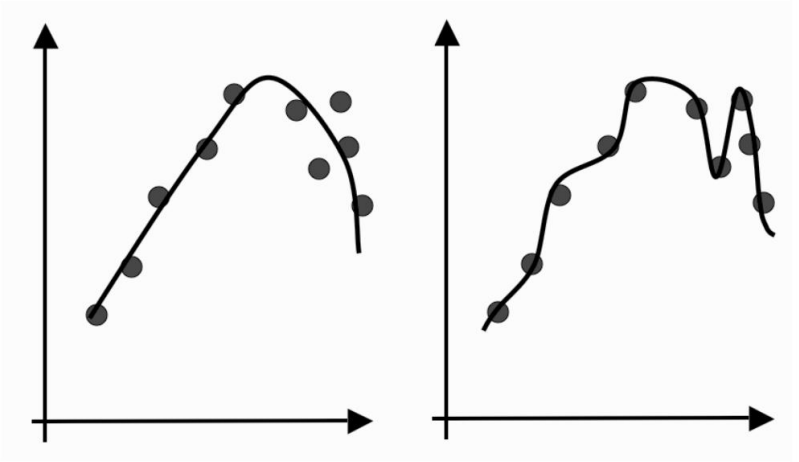

FIGURA 2.5 El efecto del sobreajuste es que, en lugar de encontrar la función generadora (como se muestra a la izquierda), la red neuronal ajusta perfectamente las entradas, incluyendo el ruido presente en ellas (a la derecha). Esto reduce la capacidad de generalización de la red.

Dos puntos diferentes en el proceso de aprendizaje. A la izquierda de la figura, la curva se ajusta correctamente a la tendencia general de los datos (se ha generalizado a la función general subyacente), pero el error de entrenamiento aún no estaría tan cerca de cero, ya que pasa cerca de los datos de entrenamiento, pero no los atraviesa. A medida que la red continúa aprendiendo, eventualmente producirá un modelo mucho más complejo con un error de entrenamiento menor (próximo a cero), lo que significa que ha memorizado los ejemplos de entrenamiento, incluyendo cualquier componente de ruido, por lo que ha sobreajustado los datos de entrenamiento.

Queremos detener el proceso de aprendizaje antes de que el algoritmo se sobreajuste, lo que significa que necesitamos saber qué tan bien se generaliza en cada paso de tiempo. No podemos usar los datos de entrenamiento para esto, ya que no detectaríamos el sobreajuste, pero tampoco podemos usar los datos de prueba, ya que los estamos guardando para las pruebas finales. Por lo tanto, necesitamos un tercer conjunto de datos para este propósito, llamado conjunto de validación, ya que lo usamos para validar el aprendizaje hasta el momento. Esto se conoce como validación cruzada en estadística. Forma parte de la selección del modelo: elegir los parámetros correctos para que el modelo se generalice lo mejor posible.

### 2.2.2 Conjuntos de entrenamiento, prueba y validación

Ahora necesitamos tres conjuntos de datos: el conjunto de entrenamiento para entrenar el algoritmo, el conjunto de validación para monitorizar su rendimiento a medida que aprende y el conjunto de prueba para generar los resultados finales. Esto se está volviendo costoso en el caso de los datos, especialmente porque, para el aprendizaje supervisado, todo debe tener valores objetivo asociados (e incluso para el aprendizaje no supervisado, los conjuntos de validación y prueba necesitan objetivos para tener algo con qué comparar), y no siempre es fácil obtener etiquetas precisas (lo que podría ser la razón por la que se desea obtener información sobre los datos). El aprendizaje semisupervisado intenta abordar esta necesidad de grandes cantidades de datos etiquetados; consulte la sección de Lecturas adicionales para obtener más referencias.

Claramente, cada algoritmo necesitará una cantidad razonable de datos para aprender (las necesidades precisas varían, pero cuantos más datos vea el algoritmo, más probable será que haya visto ejemplos de cada tipo de entrada posible, aunque más datos también aumentan el tiempo computacional para aprender). Sin embargo, el mismo argumento puede usarse para argumentar que la validación y

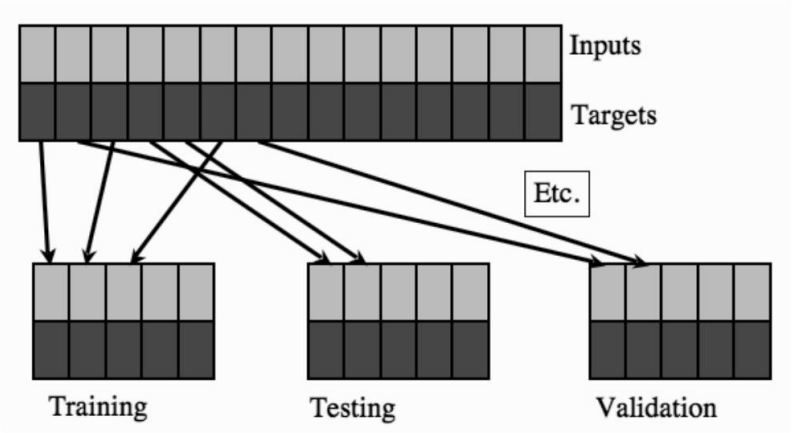


FIGURA 2.6 El conjunto de datos se divide en diferentes conjuntos, algunos para entrenamiento, otros para validación y otros para pruebas.

Los conjuntos de prueba también deben ser razonablemente grandes. Generalmente, la proporción exacta de datos de entrenamiento, prueba y validación depende del usuario, pero lo habitual es una proporción de 50:25:25 si se dispone de suficientes datos, y de 60:20:20 si no. La forma de dividirlos también puede ser importante. Muchos conjuntos de datos se presentan con el primer conjunto de puntos de datos en la clase 1, el siguiente en la clase 2, y así sucesivamente. Si se eligen los primeros puntos como conjunto de entrenamiento, los siguientes como conjunto de prueba, etc., los resultados serán bastante malos, ya que el entrenamiento no vio todas las clases. Esto se puede solucionar reordenando primero los datos aleatoriamente o asignando cada punto de datos aleatoriamente a uno de los conjuntos, como se muestra en la Figura 2.6.

Si se carece de datos de entrenamiento suficientes, por lo que, al disponer de un conjunto de validación independiente, existe la preocupación de que el algoritmo no esté lo suficientemente entrenado, es posible realizar una validación cruzada múltiple con exclusión de algunos datos. La idea se muestra en la Figura 2.7. El conjunto de datos se divide aleatoriamente en K subconjuntos, y uno de ellos se utiliza como conjunto de validación, mientras que el algoritmo se entrena con todos los demás. A continuación, se excluye un subconjunto diferente y se entrena un nuevo modelo con él, repitiendo el mismo proceso para todos los subconjuntos. Finalmente, se prueba y utiliza el modelo que produjo el menor error de validación. Hemos compensado la cantidad de datos con el tiempo de cálculo, ya que hemos tenido que entrenar K modelos diferentes en lugar de uno solo. En el caso más extremo de esto, existe la validación cruzada de dejar uno fuera, donde el algoritmo se valida solo con un dato y se entrena con todo el resto.

### 2.2.3 La Matriz de Confusión

Independientemente de la cantidad de datos que usemos para probar el algoritmo entrenado, aún necesitamos determinar si el resultado es correcto. Aquí analizaremos un método adecuado para problemas de clasificación, conocido como matriz de confusión. Para problemas de regresión, la complejidad es mayor porque los resultados son continuos, por lo que el método más común es el error de suma de cuadrados, que utilizaremos para guiar el entrenamiento en los siguientes capítulos. Veremos el uso de estos métodos al analizar ejemplos.

La matriz de confusión es una idea sencilla y atractiva: crear una matriz cuadrada que contenga todas las clases posibles, tanto en dirección horizontal como vertical, y listar las clases en la parte superior de una tabla como resultados previstos, y luego en el lado izquierdo como objetivos. Por ejemplo, el elemento de la matriz en (i, j) indica cuántos patrones de entrada se incluyeron.

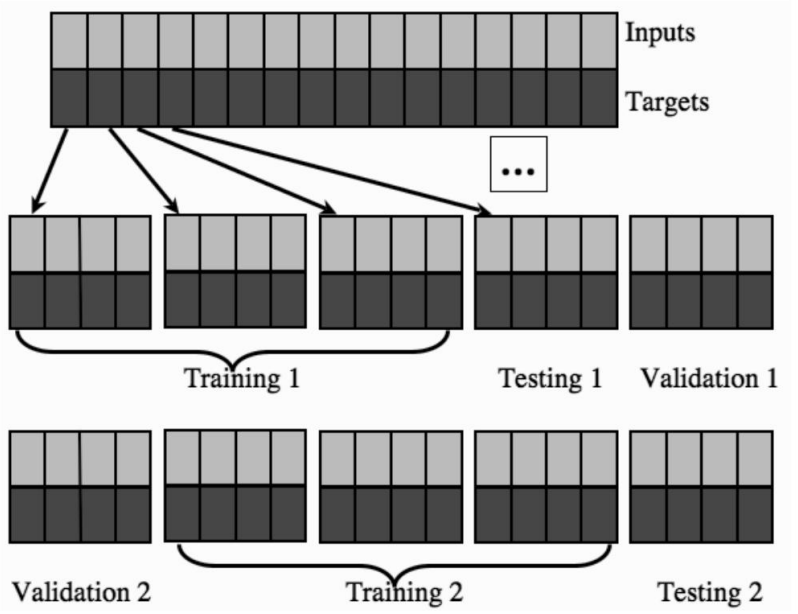


FIGURA 2.7 La validación cruzada múltiple con exclusión de datos soluciona el problema de la escasez de datos entrenando varios modelos. Funciona dividiendo los datos en conjuntos, entrenando un modelo con la mayoría de los conjuntos y reservando uno para validación (y otro para pruebas). Se entrenan diferentes modelos con diferentes conjuntos reservados.

En la clase i en los objetivos, pero en la clase j según el algoritmo. Cualquier valor en la diagonal principal (la diagonal que comienza en la esquina superior izquierda de la matriz y desciende hasta la esquina inferior derecha) es una respuesta correcta. Supongamos que tenemos tres clases: C1, C2 y C3. Ahora contamos el número de veces que la salida fue de clase C1 cuando el objetivo era C1, luego cuando el objetivo era C2, y así sucesivamente hasta completar la tabla:

| | Salidas | | |
|---|---|---|---|
| | C1 | C2 | C3 |
| C1 | 5 | 1 | 0 |
| C2 | 1 | 4 | 1 |
| C3 | 2 | 0 | 4 |

Esta tabla indica que, para las tres clases, la mayoría de los ejemplos se clasificaron correctamente, pero dos ejemplos de la clase C3 se clasificaron erróneamente como C1, y así sucesivamente. Para un número pequeño de clases, esta es una buena manera de ver los resultados. Si solo se busca un número, se puede dividir la suma de los elementos de la diagonal principal entre la suma de todos los elementos de la matriz, lo que da como resultado la fracción de respuestas correctas. Esto se conoce como precisión, y veremos que no es la última palabra al evaluar los resultados de un algoritmo de aprendizaje automático.

### 2.2.4 Métricas de precisión

Podemos hacer más para analizar los resultados que simplemente medir la precisión. Si consideramos los posibles resultados de las clases, podemos organizarlos en un gráfico simple como este.

(donde un verdadero positivo es una observación colocada correctamente en la clase 1, mientras que un falso positivo es una observación colocada incorrectamente en la clase 1, mientras que los ejemplos negativos (tanto verdaderos como falsos) son aquellos colocados en la clase 2):

| | |
|---|---|
| Verdadero Positivos | FALSO Positivos |
| FALSO Negativos | Verdadero Negativos |

Las entradas en la diagonal principal de este gráfico son correctas y las que no lo son son incorrectas, al igual que en la matriz de confusión. Sin embargo, cabe destacar que este gráfico y los conceptos de falsos positivos, etc., se basan en la clasificación binaria.

La precisión se define entonces como la suma del número de verdaderos positivos y verdaderos negativos dividida por el número total de ejemplos (donde # significa 'número de' y TP significa verdadero positivo, etc.):

$$\text{Precisión} = \frac{\#TP + \#FP}{\#TP + \#FP + \#TN + \#FN}. \tag{2.2}$$

El problema con la precisión es que no nos dice todo sobre los resultados, ya que convierte cuatro números en uno solo. Hay dos pares de medidas complementarias que pueden ayudarnos a interpretar el rendimiento de un clasificador: la sensibilidad y la especificidad, y la precisión y la recuperación. Sus definiciones se muestran a continuación, seguidas de una explicación.

$$\text{Sensibilidad} = \frac{\#TP}{\#TP + \#FN} \qquad \text{Sensitivity} \tag{2.3}$$

$$\text{Especificidad} = \frac{\#TN}{\#TN + \#FP} \qquad \text{Specificity} \tag{2.4}$$

$$\text{Precisión} = \frac{\#TP}{\#TP + \#FP} \qquad \text{Precision, != accuracy} \tag{2.5}$$

$$\text{Recordar} = \frac{\#TP}{\#TP + \#FN} \qquad \text{se llama Recall} \tag{2.6}$$

La sensibilidad (también conocida como tasa de verdaderos positivos) es la razón entre el número de ejemplos positivos correctos y el número de ejemplos clasificados como positivos, mientras que la especificidad es la misma razón para los ejemplos negativos. La precisión es la razón entre los ejemplos positivos correctos y el número de ejemplos positivos reales, mientras que la recuperación es la razón entre el número de ejemplos positivos correctos y los que se clasificaron como positivos, lo cual es equivalente a la sensibilidad. Si observa el gráfico de nuevo, puede ver que la sensibilidad y la especificidad suman las columnas del denominador, mientras que la precisión y la recuperación suman la primera columna y la primera fila, por lo que omiten información sobre el rendimiento del alumno en los ejemplos negativos.

En conjunto, cualquiera de estos pares de medidas proporciona más información que sólo la precisión.
Si consideramos la precisión y la recuperación, podemos observar que, hasta cierto punto, están inversamente relacionadas, ya que si el número de falsos positivos aumenta (lo que significa que el algoritmo utiliza una definición más amplia de esa clase), el número de falsos negativos suele disminuir, y viceversa. Pueden combinarse para obtener una única medida, la medida F1 , que puede expresarse en términos de precisión y recuperación como:

$$F1 = 2\,\frac{\text{precisión} \times \text{recuperación}}{\text{precisión} + \text{recuperación}} \qquad \text{recuperacion = recall} \tag{2.7}$$

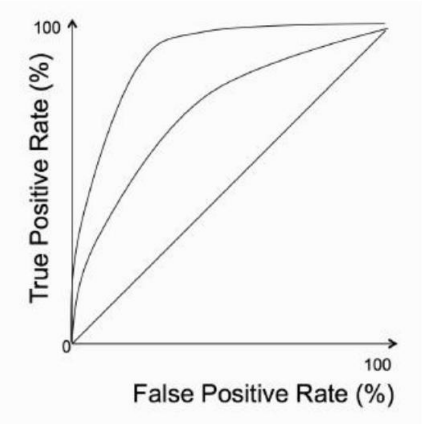


FIGURA 2.8 Ejemplo de una curva ROC. La línea diagonal representa exactamente la casualidad, por lo que cualquier valor por encima de la línea es mejor que la casualidad, y cuanto más lejos de la línea, mejor. De las dos curvas mostradas, la que está más alejada de la línea diagonal representaría un método más preciso.

y en términos de números de falsos positivos, etc. (de donde se puede ver que calcula la media de los ejemplos falsos) como:

$$F1 = \frac{\#TP}{\#TP + (\#F\ N + \#F\ P)/2}. \tag{2.8}$$

### 2.2.5 La curva característica del operador del receptor (ROC) estudiar curva ROC con atencion

Dado que podemos usar estas medidas para evaluar un clasificador en particular, también podemos comparar clasificadores, ya sea el mismo clasificador con diferentes parámetros de aprendizaje o clasificadores completamente diferentes. En este caso, la curva ROC (casi siempre conocida como curva ROC) es útil. Esta es una gráfica del porcentaje de verdaderos positivos en el eje y frente a los falsos positivos en el eje x ; se muestra un ejemplo en la Figura 2.8. Una sola ejecución de un clasificador produce un único punto en la gráfica ROC, y un clasificador perfecto sería un punto en (0, 1) (100 % de verdaderos positivos, 0 % de falsos positivos), mientras que el anticlasificador que se equivocó en todo estaría en (1, 0); por lo tanto, cuanto más cerca de la esquina superior izquierda esté el resultado de un clasificador, mejor ha sido su rendimiento. Cualquier clasificador que se encuentre en la línea diagonal de (0,0) a (1,1) se comporta exactamente en el nivel de azar (asumiendo que las clases positivas y negativas son igualmente comunes) y, por lo tanto, presumiblemente se desperdicia mucho esfuerzo de aprendizaje ya que una moneda justa funcionaría igual de bien.

Para comparar clasificadores o elecciones de configuraciones de parámetros para el mismo clasificador, podría simplemente calcular el punto que está más alejado de la línea de "posibilidad" a lo largo de la diagonal. Sin embargo, lo habitual es calcular el área bajo la curva (AUC). Si solo se tiene un punto para cada clasificador, la curva es el trapezoide que va desde (0,0) hasta el punto y, de ahí, hasta (1,1). Si hay más puntos (basados en más ejecuciones del clasificador, como entrenados o probados en diferentes conjuntos de datos), simplemente se incluyen en orden a lo largo de la línea diagonal.

La clave para obtener una curva en lugar de un punto en la curva ROC es usar la validación cruzada. Si se usa una validación cruzada de 10 pasos, se obtienen 10 clasificadores con 10 valores diferentes.

Conjuntos de prueba, y también se incluyen las etiquetas de "verdad fundamental". Estas etiquetas permiten generar una lista ordenada de los diferentes resultados entrenados mediante validación cruzada, lo que permite especificar una curva a través de los 10 puntos de datos de la curva ROC que corresponden a los resultados de este clasificador. Al generar una curva ROC para cada clasificador, es posible comparar sus resultados.

### 2.2.6 Conjuntos de datos no balanceados

Tenga en cuenta que para lograr precisión hemos asumido implícitamente que hay el mismo número de ejemplos positivos y negativos en el conjunto de datos (lo que se conoce como conjunto de datos equilibrado). Sin embargo, esto a menudo no es cierto (lo que también puede causar problemas a los estudiantes, como veremos más adelante en el libro). En caso contrario, podemos calcular la precisión equilibrada como la suma de la sensibilidad y la especificidad dividida entre 2. Sin embargo, una medida más correcta es el coeficiente de correlación de Matthew, que se calcula como:

$$MCC = \frac{\#TP \times \#TN - \#FP \times \#FN}{(\#TP + \#FP)(\#TP + \#FN)(\#TN + \#FP)(\#TN + \#FN)} \tag{2.9}$$

Si alguno de los paréntesis en el denominador es 0, entonces todo el denominador es Establecido en 1. Esto proporciona un cálculo de precisión equilibrado.

Como nota final sobre estos métodos de evaluación, si hay más de dos clases y resulta útil distinguir los diferentes tipos de error, los cálculos se complican un poco, ya que en lugar de un conjunto de falsos positivos y uno de falsos negativos, se tienen para cada clase. En este caso, la especificidad y la recuperación no son lo mismo. Sin embargo, es posible crear un conjunto de resultados donde se utiliza una clase como positiva y el resto como negativa, y repetir este proceso para cada clase.

### 2.2.7 Precisión de la medición

Existe una forma diferente de evaluar la precisión de un sistema de aprendizaje, que lamentablemente también utiliza el término "precisión", aunque con un significado distinto. El concepto aquí es tratar el algoritmo de aprendizaje automático como un sistema de medición. Introducimos datos de entrada y observamos los resultados que obtenemos. Incluso antes de compararlos con los valores objetivo, podemos medir algo del algoritmo: si introducimos un conjunto de datos de entrada similares, esperaríamos obtener resultados similares. Esta medida de la variabilidad del algoritmo, también conocida como precisión, nos indica la repetibilidad de sus predicciones. Podría ser útil pensar en la precisión como algo similar a la varianza de una distribución de probabilidad: indica la dispersión esperada alrededor de la media.

La cuestión es que la precisión de un algoritmo no implica que sea exacto; puede ser totalmente erróneo si siempre arroja una predicción errónea. Una medida de la precisión con la que las predicciones del algoritmo se ajustan a la realidad se conoce como veracidad, y puede definirse como la distancia promedio entre el resultado correcto y la predicción. La veracidad no suele tener mucho sentido en los problemas de clasificación, a menos que exista el concepto de que ciertas clases sean similares entre sí. La Figura 2.9 ilustra el concepto de veracidad y precisión de forma tradicional: como un juego de dardos, con cuatro ejemplos con veracidad y precisión variables para los tres dardos lanzados por un jugador.

En esta sección se ha considerado el punto final del aprendizaje automático, mirando los resultados y pensando en lo que debemos hacer con los datos de entrada en términos de tener múltiples conjuntos de datos, etc. En la siguiente sección, volvemos al punto de partida y consideramos cómo podemos comenzar a analizar un conjunto de datos al tratar con probabilidades.

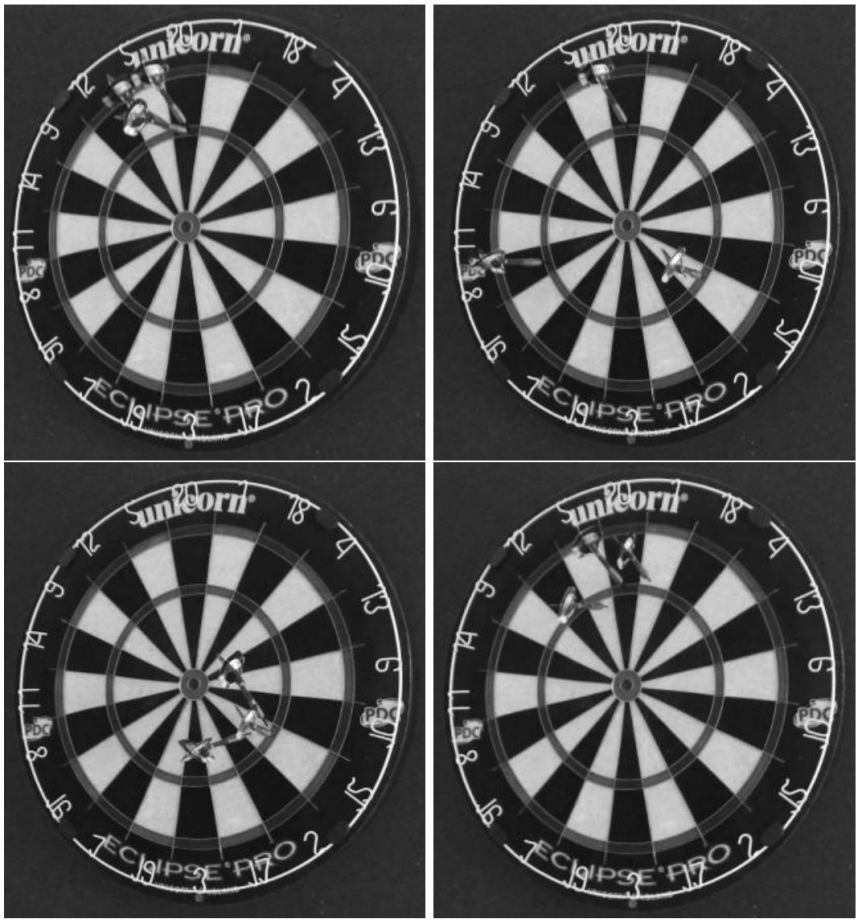


FIGURA 2.9 Suponiendo que el jugador apuntaba al triple 20 de mayor puntuación en dardos (cada segmento puntúa el número que le corresponde; la banda estrecha exterior del círculo puntúa el doble y la banda estrecha a mitad de camino el triple; la diana exterior e interior del centro puntúan 25 y 50, respectivamente), estas cuatro imágenes muestran resultados diferentes. Arriba a la izquierda: muy preciso: alta precisión y veracidad; arriba a la derecha: baja precisión, pero buena veracidad; abajo a la izquierda: alta precisión, pero baja veracidad; y abajo a la derecha: veracidad y precisión razonables, pero los resultados reales no son muy buenos. (Gracias a Stefan Nowicki, cuya diana se utilizó para estas imágenes).

# Capítulo 3. Clasificación

En el Capítulo 1 mencioné que las tareas de aprendizaje supervisado más comunes son la regresión (predicción de valores) y la clasificación (predicción de clases). En el Capítulo 2, exploramos una tarea de regresión para predecir el valor de las viviendas mediante diversos algoritmos, como la regresión lineal, los árboles de decisión y los bosques aleatorios (que se explicarán con más detalle en capítulos posteriores). Ahora nos centraremos en los sistemas de clasificación.

# MNIST

En este capítulo, utilizaremos el conjunto de datos MNIST, compuesto por 70.000 imágenes pequeñas de dígitos escritos a mano por estudiantes de secundaria y empleados de la Oficina del Censo de EE. UU. Cada imagen está etiquetada con el dígito que representa.
Este conjunto se ha estudiado tanto que a menudo se lo llama el “hola mundo” del aprendizaje automático: siempre que las personas inventan un nuevo algoritmo de clasificación, sienten curiosidad por ver cómo funcionará en MNIST, y cualquiera que aprende aprendizaje automático aborda este conjunto de datos tarde o temprano.

Scikit-Learn proporciona muchas funciones útiles para descargar conjuntos de datos populares. MNIST es uno de ellos. El siguiente código obtiene el conjunto de datos MNIST de OpenML.org:

```
1
desde sklearn.datasets importar fetch_openml

mnist = fetch_openml('mnist_784', como_marco=Falso)
```

El paquete sklearn.datasets contiene principalmente tres tipos de funciones: funciones fetch_*, como fetch_openml(), para descargar conjuntos de datos reales; funciones load_*, para cargar pequeños conjuntos de datos de prueba incluidos en Scikit-Learn (para que no sea necesario descargarlos de internet); y funciones make_*, para generar conjuntos de datos ficticios, útiles para pruebas. Los conjuntos de datos generados suelen devolverse como una tupla (X, y) que contiene los datos de entrada y los destinos, ambos como matrices de NumPy. Otros conjuntos de datos se devuelven como objetos sklearn.utils.Bunch, que son diccionarios cuyas entradas también se pueden acceder como atributos. Generalmente contienen las siguientes entradas:

"DESCRIBE"

Una descripción del conjunto de datos

"datos"

Los datos de entrada, generalmente como una matriz NumPy 2D

"objetivo"

Las etiquetas, generalmente como una matriz NumPy 1D

La función fetch_openml() es un poco inusual, ya que, por defecto, devuelve las entradas como un DataFrame de Pandas y las etiquetas como una Serie de Pandas (a menos que el conjunto de datos sea disperso). Sin embargo, el conjunto de datos MNIST contiene imágenes, y los DataFrames no son ideales para ello, por lo que es preferible establecer as_frame=False para obtener los datos como arrays de NumPy. Analicemos estos arrays:

```
>>> X, y = mnist.datos, mnist.objetivo >>> X

matriz([[0., 0., 0., ..., 0., 0., 0.], [0.,
       0., 0., ..., 0., 0., 0.], [0., 0.,
       0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.], [0.,
       0., 0., ..., 0., 0., 0.], [0., 0.,
       0., ..., 0., 0., 0.]])
>>> X.shape
(70000, 784)
>>> y
array(['5', '0', '4', ..., '4', '5', '6'], dtype=object) >>>
y.shape
(70000,)
```

Hay 70.000 imágenes, y cada una tiene 784 características. Esto se debe a que cada imagen tiene 28 × 28 píxeles, y cada característica simplemente representa la intensidad de un píxel, de 0 (blanco) a 255 (negro). Analicemos un dígito del conjunto de datos (Figura 3-1). Solo necesitamos obtener el vector de características de una instancia, transformarlo en una matriz de 28 × 28 y mostrarlo usando la función imshow() de Matplotlib. Usamos cmap="binary" para obtener un mapa de color en escala de grises donde 0 es blanco y 255 es negro:

```
importar matplotlib.pyplot como plt

def plot_digit(datos_de_imagen):
    imagen = datos_de_imagen.reshape(28,
    28) plt.imshow(imagen, cmap="binario")
    plt.axis("desactivado")
```

```
algún_dígito = X[0]
plot_digit(algún_dígito)
plt.show()
```

Figura 3-1. Ejemplo de una imagen MNIST

Esto parece un 5, y de hecho eso es lo que nos dice la etiqueta:

```
>>> y[0] '5'
```

Para darle una idea de la complejidad de la tarea de clasificación, la Figura 3-2 muestra algunas imágenes más del conjunto de datos MNIST.

¡Pero espera! Siempre debes crear un conjunto de prueba y reservarlo antes de inspeccionar los datos con detenimiento. El conjunto de datos MNIST devuelto por fetch_openml() ya está dividido en un conjunto de entrenamiento (las primeras 60 000 imágenes) y un conjunto de prueba (las últimas 10 000 imágenes):[2]

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

El conjunto de entrenamiento ya está reorganizado, lo cual es positivo, ya que garantiza que todos los pliegues de validación cruzada sean similares (no queremos que a un pliegue le falten algunos dígitos). Además, algunos algoritmos de aprendizaje son sensibles al orden de las instancias de entrenamiento y su rendimiento es bajo si obtienen muchas instancias similares seguidas. Reorganizar el conjunto de datos garantiza que esto no ocurra.[3]

Figura 3-2. Dígitos del conjunto de datos MNIST

Entrenando un clasificador binario Simplifiquemos el

problema por ahora e intentemos identificar solo un dígito, por ejemplo, el número 5. Este "detector 5" será un ejemplo de un clasificador binario, capaz de distinguir entre solo dos clases, 5 y no 5.
Primero crearemos los vectores de destino para esta tarea de clasificación:

```
y_train_5 = (y_train == '5') # Verdadero para todos los 5, falso para todos los demás dígitos
y_test_5 = (y_test == '5')
```

Ahora, seleccionemos un clasificador y entrenémoslo. Un buen punto de partida es un clasificador de descenso de gradiente estocástico (SGD, o GD estocástico), utilizando la clase SGDClassifier de Scikit-Learn. Este clasificador es capaz de gestionar conjuntos de datos muy grandes de forma eficiente. Esto se debe, en parte, a que SGD gestiona las instancias de entrenamiento de forma independiente, una a la vez, lo que también lo hace ideal para el aprendizaje en línea, como se verá más adelante. Creemos un SGDClassifier y entrenémoslo con todo el conjunto de entrenamiento:

```
desde sklearn.linear_model importar SGDClassifier

sgd_clf = SGDClassifier(estado_aleatorio=42)
sgd_clf.fit(entrenamiento_X, entrenamiento_y_5)
```

Ahora podemos usarlo para detectar imágenes del número 5:

```
>>> sgd_clf.predict([algún_dígito])
array([Verdadero])
```

El clasificador supone que esta imagen representa un 5 (Verdadero). ¡Parece que acertó en este caso! Ahora, evaluemos el rendimiento de este modelo.

## Medidas de desempeño

Evaluar un clasificador suele ser mucho más complejo que evaluar un regresor, por lo que dedicaremos gran parte de este capítulo a este tema. Hay muchas medidas de rendimiento disponibles, así que ¡prepárate para aprender un montón de conceptos y acrónimos nuevos!

Medición de la precisión mediante validación cruzada Una buena

forma de evaluar un modelo es utilizar la validación cruzada, tal como lo hizo en el Capítulo 2. Utilicemos la función cross_val_score() para evaluar nuestro modelo SGDClassifier, utilizando la validación cruzada de k pliegues con tres pliegues. Recuerde que la validación cruzada de k pliegues significa dividir el conjunto de entrenamiento en k pliegues (en este caso, tres), luego entrenar el modelo k veces, dejando un pliegue diferente cada vez para la evaluación (ver Capítulo 2):

```
>>> de sklearn.model_selection importar cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="precisión")
array([0.95035, 0.96035, 0.9604 ])
```

¡Guau! ¿Más del 95 % de precisión (proporción de predicciones correctas) en todos los pliegues de validación cruzada? ¡Parece increíble, verdad? Bueno, antes de que te emociones demasiado, veamos un clasificador ficticio que simplemente clasifica cada imagen en la clase más frecuente, que en este caso es la clase negativa (es decir, distinta de 5):

```
de sklearn.dummy importar DummyClassifier

dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train_5)
print(any(dummy_clf.predict(X_train))) # imprime Falso: no se detectaron 5
```

¿Puedes adivinar la precisión de este modelo? Averigüémoslo:

```
>>> cross_val_score(dummy_clf, X_train, y_train_5, cv=3, puntuación="precisión")
array([0.90965, 0.90965, 0.90965])
```

Así es, ¡tiene más del 90% de precisión! Esto se debe simplemente a que solo alrededor del 10% de las imágenes son 5, así que si siempre adivinas que una imagen no es un 5, acertarás aproximadamente el 90% de las veces. Supera a Nostradamus.

Esto demuestra por qué la precisión generalmente no es la medida de rendimiento preferida para los clasificadores, especialmente cuando se trabaja con conjuntos de datos sesgados (es decir, cuando algunas clases son mucho más frecuentes que otras). Una mucho mejor

La forma de evaluar el rendimiento de un clasificador es observar la matriz de confusión (CM).

### IMPLEMENTACIÓN DE LA VALIDACIÓN CRUZADA

En ocasiones, necesitará más control sobre el proceso de validación cruzada que el que ofrece Scikit-Learn de fábrica. En estos casos, puede implementar la validación cruzada usted mismo. El siguiente código realiza prácticamente lo mismo que la función cross_val_score() de Scikit-Learn e imprime el mismo resultado:

```
desde sklearn.model_selection importar StratifiedKFold desde
sklearn.base importar clonar

skfolds = StratifiedKFold(n_splits=3) # agrega shuffle=True si el conjunto de datos es
                                      # no ya mezclados para
train_index, test_index en skfolds.split(X_train, y_train_5): clone_clf =
    clone(sgd_clf)
    X_tren_pliegues = X_tren[índice_tren]
    y_tren_pliegues = y_tren_5[índice_tren]
    X_prueba_fold = X_train[índice_prueba]
    y_prueba_fold = y_train_5[índice_prueba]

    clone_clf.fit(X_train_folds, y_train_folds) y_pred =
    clone_clf.predict(X_test_fold) n_correct =
    suma(y_pred == y_test_fold) print(n_correct /
    len(y_pred)) # imprime 0,95035, 0,96035 y 0,9604
```

La clase StratifiedKFold realiza un muestreo estratificado (como se explica en el Capítulo 2) para generar pliegues que contienen una proporción representativa de cada clase. En cada iteración, el código crea un clon del clasificador, lo entrena con los pliegues de entrenamiento y realiza predicciones con el pliegue de prueba. Luego cuenta el número de predicciones correctas y genera la proporción de predicciones correctas.

## Matrices de confusión

La idea general de una matriz de confusión es contar el número de veces que las instancias de la clase A se clasifican como clase B, para todos los pares A/B. Por ejemplo, para saber cuántas veces el clasificador confundió imágenes de 8 con 0, se debe consultar la fila 8, columna 0 de la matriz de confusión.

Para calcular la matriz de confusión, primero necesita un conjunto de predicciones para compararlas con los objetivos reales. Podría hacer predicciones con el conjunto de pruebas, pero es mejor no modificarlo por ahora (recuerde que solo debe usar el conjunto de pruebas al final del proyecto, una vez que tenga un clasificador listo para ejecutar). En su lugar, puede usar la función cross_val_predict():

```
desde sklearn.model_selection importar cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

Al igual que la función cross_val_score(), cross_val_predict() realiza una validación cruzada de k-folds, pero en lugar de devolver las puntuaciones de evaluación, devuelve las predicciones realizadas en cada fold de prueba. Esto significa que se obtiene una predicción limpia para cada instancia del conjunto de entrenamiento (por "limpia" me refiero a "fuera de muestra": el modelo realiza predicciones con datos que nunca vio durante el entrenamiento).

Ahora está listo para obtener la matriz de confusión usando la función confusion_matrix(). Simplemente pásele las clases objetivo (y_train_5) y las clases predichas (y_train_pred):

```
>>> de sklearn.metrics importar matriz_confusión >>> cm =
matriz_confusión(y_train_5, y_train_pred) >>> cm matriz([[53892, 687],
[ 1891,
3530]])
```

Cada fila de una matriz de confusión representa una clase real, mientras que cada columna representa una clase predicha. La primera fila de esta matriz considera imágenes distintas de 5 (la clase negativa): 53.892 de ellas se clasificaron correctamente como no-

5 (se denominan verdaderos negativos), mientras que los 687 restantes se clasificaron erróneamente como 5 (falsos positivos, también llamados errores de tipo I). La segunda fila considera las imágenes de 5 (la clase positiva): 1891 se clasificaron erróneamente como no 5 (falsos negativos, también llamados errores de tipo II), mientras que los 3530 restantes se clasificaron correctamente como 5 (verdaderos positivos). Un clasificador perfecto solo tendría verdaderos positivos y verdaderos negativos, por lo que su matriz de confusión tendría valores distintos de cero solo en su diagonal principal (de arriba a la izquierda a abajo a la derecha).

```
>>> y_train_perfect_predictions = y_train_5 # imaginamos que alcanzamos la perfección >>>
confusion_matrix(y_train_5, y_train_perfect_predictions) array([[54579, 0], [ 0,
5421]])
```

La matriz de confusión proporciona mucha información, pero a veces se prefiere una métrica más concisa. Una métrica interesante es la precisión de las predicciones positivas; esto se denomina precisión del clasificador (Ecuación 3-1).

Ecuación 3-1. Precisión

precisión = TP TP+FP

TP es el número de verdaderos positivos y FP es el número de falsos positivos.

Una forma sencilla de lograr una precisión perfecta es crear un clasificador que siempre realice predicciones negativas, excepto una predicción positiva en la instancia en la que tiene mayor confianza. Si esta predicción es correcta, el clasificador tiene una precisión del 100 % (precisión = 1/1 = 100 %). Obviamente, este clasificador no sería muy útil, ya que ignoraría todas las instancias positivas menos una. Por lo tanto, la precisión se suele usar junto con otra métrica llamada recuperación, también llamada sensibilidad o tasa de verdaderos positivos (TPR): esta es la proporción de instancias positivas que el clasificador detecta correctamente (Ecuación 3-2 ).

Ecuación 3-2. Recordatorio

recuperación = TP TP+FN

FN es, por supuesto, el número de falsos negativos.

Si está confundido acerca de la matriz de confusión, la Figura 3-3 puede ayudarlo.

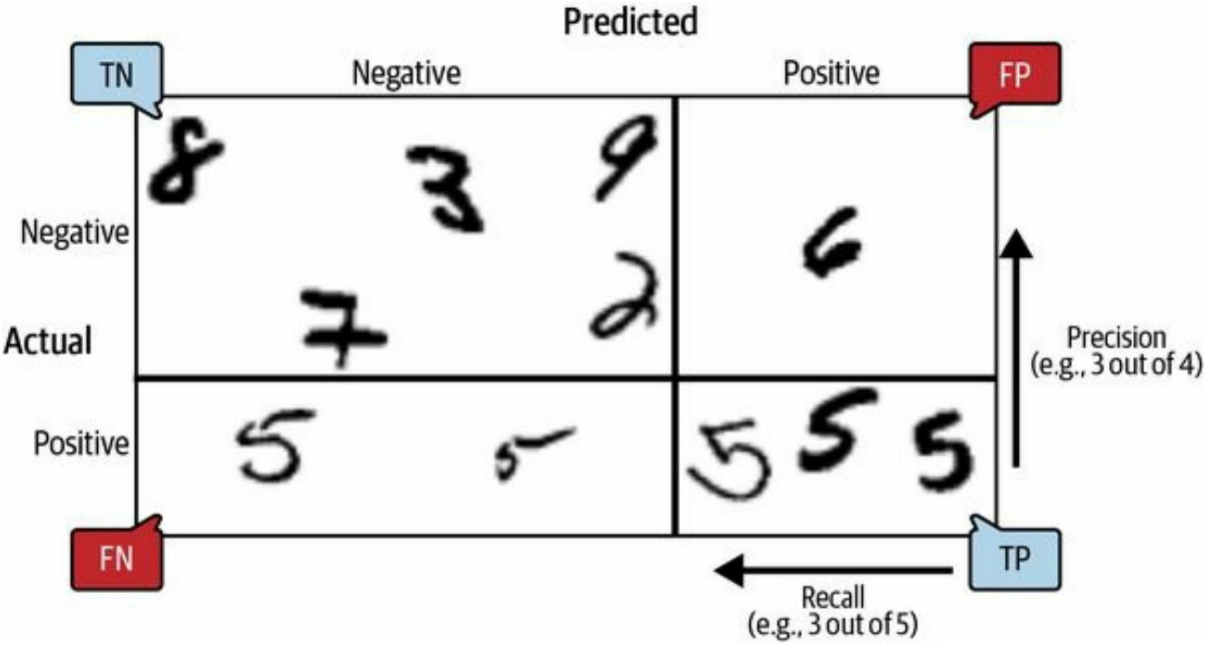


Figura 3-3. Matriz de confusión ilustrada que muestra ejemplos de verdaderos negativos (arriba a la izquierda), falsos positivos (arriba a la derecha), falsos negativos (abajo a la izquierda) y verdaderos positivos (abajo a la derecha).

## Precisión y recuperación

Scikit-Learn proporciona varias funciones para calcular métricas del clasificador, incluida la precisión y la recuperación:

```
>>> de sklearn.metrics importar puntuación_precisión, puntuación_recuperación
>>> puntuación_precisión(y_train_5, y_train_pred) # == 3530 / (687 + 3530)
0.8370879772350012 >>>
puntuación_recuperación(y_train_5, y_train_pred) # == 3530 / (1891 + 3530)
0.6511713705958311
```

Ahora nuestro detector de 5 no luce tan brillante como cuando analizamos su precisión. Cuando afirma que una imagen representa un 5, solo acierta el 83,7 % de las veces. Además, solo detecta el 65,1 % de los 5.

Suele ser conveniente combinar la precisión y la recuperación en una única métrica llamada puntuación F, especialmente cuando se necesita una sola métrica para comparar dos clasificadores. La puntuación F es la media armónica de la precisión y la recuperación (Ecuación 3-3). Mientras que la media regular trata todos los valores por igual, la media armónica otorga mucha más importancia a los valores bajos. En consecuencia, el clasificador solo obtendrá una puntuación F alta si tanto la recuperación como la precisión son altas.

Ecuación 3-3. Puntuación F

$$F_1 = 2 \frac{1}{\frac{1}{\text{precisión}} + \frac{1}{\text{recuperación}}} = 2 \times \frac{\text{precisión} \times \text{recuperación}}{\text{precisión} + \text{recuperación}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Para calcular la puntuación F, simplemente llame a la función f1_score():

```
>>> de sklearn.metrics importar f1_score >>>
f1_score(y_train_5, y_train_pred)
0.7325171197343846
```

La puntuación F favorece a los clasificadores con precisión y recuperación similares. Esto no siempre es lo deseado: en algunos contextos, la precisión es fundamental, mientras que en otros, la recuperación es fundamental. Por ejemplo, si se entrena un clasificador para detectar vídeos seguros para niños, probablemente se preferiría un clasificador que rechace muchos vídeos buenos (baja recuperación) y conserve solo los seguros.

(alta precisión), en lugar de un clasificador que tiene una recuperación mucho mayor pero permite que aparezcan algunos videos realmente malos en su producto (en tales casos, es posible que incluso desee agregar una tubería humana para verificar la selección de videos del clasificador). Por otro lado, supongamos que entrenas un clasificador para detectar ladrones de tiendas en imágenes de vigilancia: probablemente esté bien si tu clasificador solo tiene un 30% de precisión siempre que tenga un 99% de recuperación (claro, los guardias de seguridad recibirán algunas alertas falsas, pero casi todos los ladrones de tiendas serán atrapados).

Desafortunadamente, no se puede tener todo a la vez: aumentar la precisión reduce la recuperación, y viceversa. Esto se conoce como el equilibrio entre precisión y recuperación.

## El equilibrio entre precisión y recuperación

Para comprender esta compensación, veamos cómo el SGDClassifier toma sus decisiones de clasificación. Para cada instancia, calcula una puntuación basada en una función de decisión. Si dicha puntuación supera un umbral, asigna la instancia a la clase positiva; de lo contrario, la asigna a la clase negativa.
La Figura 3-4 muestra algunos dígitos ubicados desde la puntuación más baja a la izquierda hasta la más alta a la derecha. Supongamos que el umbral de decisión se encuentra en la flecha central (entre los dos 5): encontrará 4 verdaderos positivos (5 reales) a la derecha de dicho umbral y 1 falso positivo (un 6 real).
Por lo tanto, con ese umbral, la precisión es del 80 % (4 de 5). Sin embargo, de 6 5 reales, el clasificador solo detecta 4, por lo que la recuperación es del 67 % (4 de 6). Si se aumenta el umbral (muévalo hacia la flecha de la derecha), el falso positivo (el 6) se convierte en un verdadero negativo, lo que aumenta la precisión (hasta el 100 % en este caso), pero un verdadero positivo se convierte en un falso negativo, lo que reduce la recuperación al 50 %. Por el contrario, al reducir el umbral, se aumenta la recuperación y se reduce la precisión.

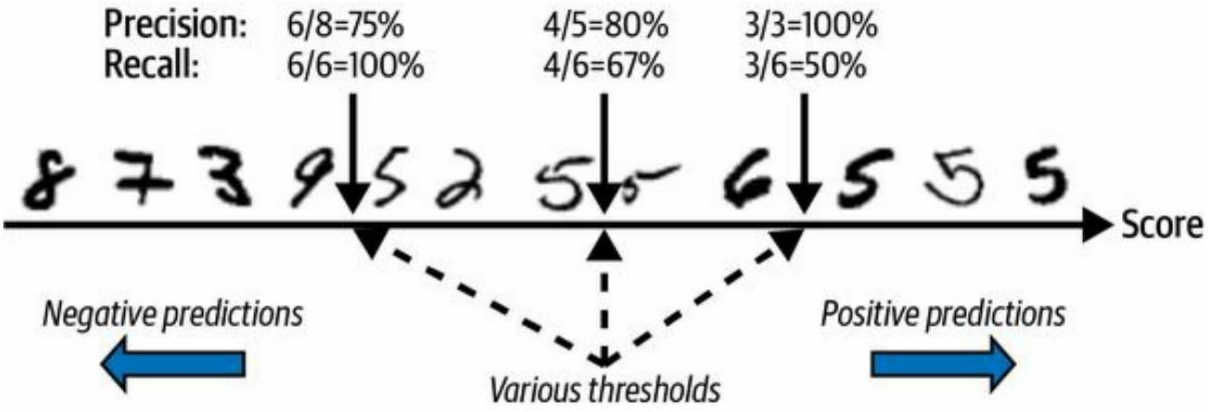


Figura 3-4. El equilibrio entre precisión y recuperación: las imágenes se clasifican según su puntuación de clasificador, y las que superan el umbral de decisión elegido se consideran positivas; cuanto mayor sea el umbral, menor será la recuperación, pero (en general) mayor será la precisión.

Scikit-Learn no permite establecer el umbral directamente, pero sí da acceso a las puntuaciones de decisión que utiliza para realizar predicciones. En lugar de llamar al método predict() del clasificador, se puede llamar a su método decision_function(), que devuelve una puntuación para cada instancia, y luego usar cualquier umbral que se desee para realizar predicciones basadas en esas puntuaciones:

```
>>> y_scores = sgd_clf.decision_function([algún_dígito]) >>> y_scores

array([2164.22030239]) >>>
umbral = 0 >>>
y_algún_dígito_pred = (y_scores > umbral) array([Verdadero])
```

El SGDClassifier usa un umbral igual a 0, por lo que el código anterior devuelve el mismo resultado que el método predict() (es decir, True). Aumentemos el umbral:

```
>>> umbral = 3000 >>>
y_algunos_dígitos_pred = (y_puntuaciones > umbral) >>>
y_algunos_dígitos_pred
array([Falso])
```

Esto confirma que aumentar el umbral disminuye la recuperación. La imagen representa un 5, y el clasificador lo detecta cuando el umbral es 0, pero no lo detecta cuando el umbral se eleva a 3000.

¿Cómo se decide qué umbral usar? Primero, use la función cross_val_predict() para obtener las puntuaciones de todas las instancias del conjunto de entrenamiento, pero esta vez especifique que desea obtener puntuaciones de decisión en lugar de predicciones:

```
puntuaciones_y = predicción_val_cruzada(sgd_clf, entrenamiento_X, entrenamiento_y_5,
                             cv=3, método="función_de_decisión")
```

Con estos puntajes, use la función precision_recall_curve() para calcular la precisión y la recuperación para todos los umbrales posibles (la función agrega una última precisión de 0 y una última recuperación de 1, correspondiente a un umbral infinito):

```
de sklearn.metrics importar precision_recall_curve

precisiones, recuperaciones, umbrales = precisión_recuperación_curva(y_train_5, y_scores)
```

Finalmente, utilice Matplotlib para representar gráficamente la precisión y la recuperación en función del valor umbral (Figura 3-5). A continuación, mostremos el umbral de 3000 que seleccionamos:

```
plt.plot(umbrales, precisiones[:-1], "b--", etiqueta="Precisión", ancho de línea=2) plt.plot(umbrales,
recuperaciones[:-1], "g-", etiqueta="Recuperación", ancho de línea=2) plt.vlines(umbral, 0,
1.0, "k", "punteado", etiqueta="umbral")
```

```
[...] # embellecer la figura: agregar cuadrícula, leyenda, eje, etiquetas y círculos
plt.show()
```

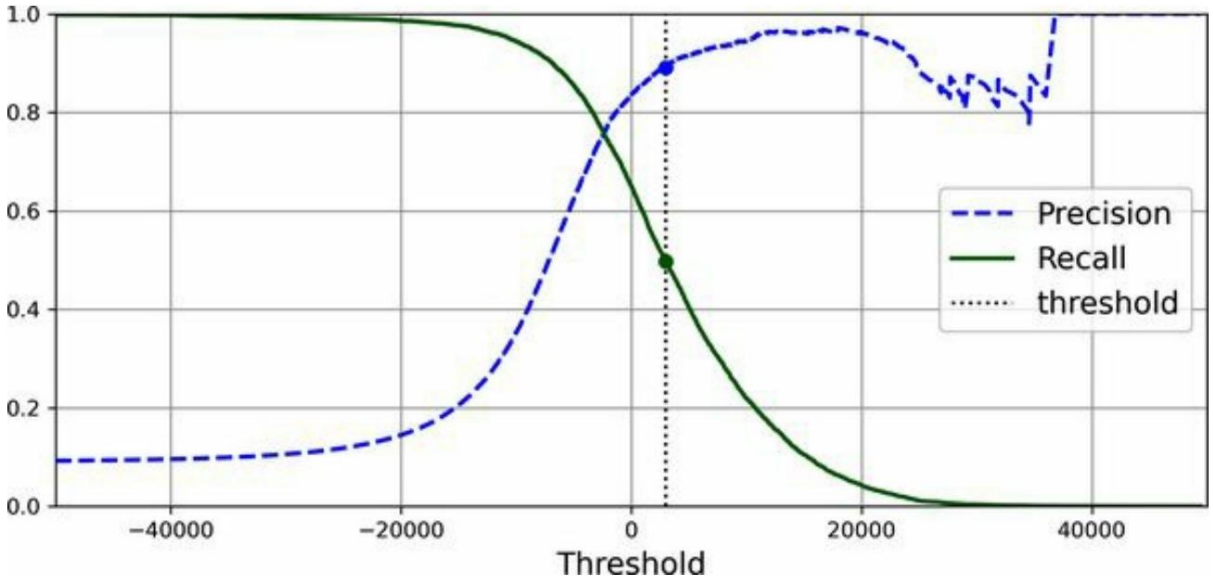


Figura 3-5. Precisión y recuperación frente al umbral de decisión

> NOTA
>
> Quizás se pregunte por qué la curva de precisión es más irregular que la curva de recuperación en la Figura 3-5. La razón es que la precisión a veces puede disminuir al aumentar el umbral (aunque, en general, aumentará). Para comprender por qué, revise la Figura 3-4 y observe lo que sucede al comenzar desde el umbral central y moverlo solo un dígito a la derecha: la precisión baja de 4/5 (80%) a 3/4 (75%). Por otro lado, la recuperación solo puede disminuir al aumentar el umbral, lo que explica por qué su curva se ve suave.

En este valor umbral, la precisión es cercana al 90% y la recuperación es de alrededor del 50%.

Otra forma de seleccionar un buen equilibrio entre precisión y recuperación es representar gráficamente la precisión directamente en función de la recuperación, como se muestra en la Figura 3-6 (se muestra el mismo umbral):

```
plt.plot(recalls, precisions, linewidth=2, label="Curva de precisión/recall") [...] # embellecer
la figura: agregar etiquetas, cuadrícula, leyenda, flecha y texto plt.show()
```

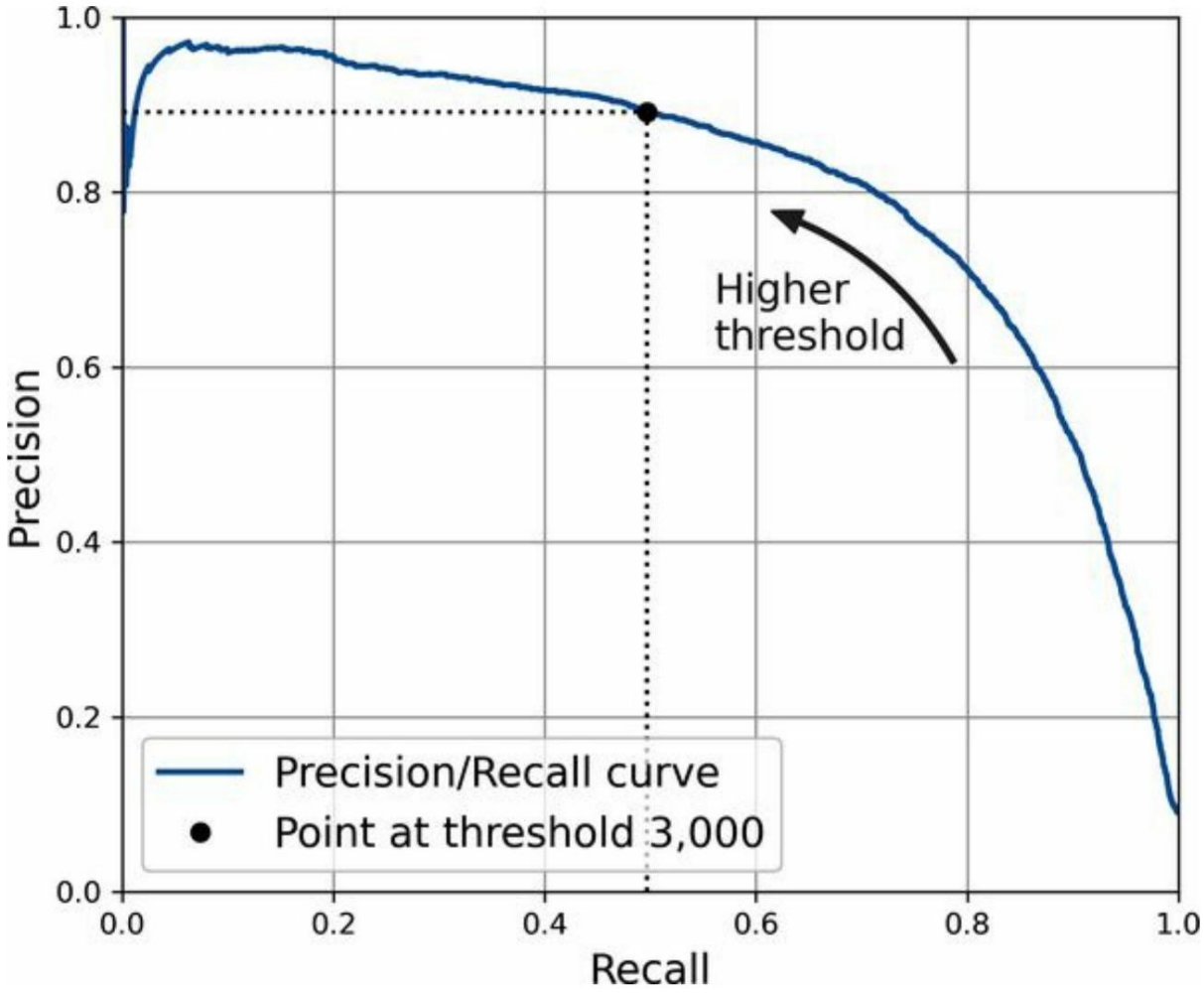


Figura 3-6. Precisión versus recuperación

Se puede observar que la precisión realmente comienza a caer drásticamente alrededor del 80% de recuperación. Probablemente querrá seleccionar un equilibrio entre precisión y recuperación justo antes de esa caída, por ejemplo, alrededor del 60 % de recuperación. Pero, por supuesto, la elección depende de su proyecto.

Supongamos que decide alcanzar una precisión del 90 %. Podría usar el primer gráfico para determinar el umbral necesario, pero no es muy preciso. Como alternativa, puede buscar el umbral más bajo que le proporcione al menos un 90 % de precisión.

Para ello, puede usar el método argmax() de la matriz NumPy. Este método devuelve el primer índice del valor máximo, que en este caso corresponde al primer valor verdadero:

```
>>> idx_para_90_precisión = (precisiones >= 0.90).argmax() >>>
umbral_para_90_precisión = umbrales[idx_para_90_precisión]
```

```
>>> umbral_para_precisión_90
3370.0194991439557
```

Para hacer predicciones (en el conjunto de entrenamiento por ahora), en lugar de llamar al método predict() del clasificador, puede ejecutar este código:

```
y_train_pred_90 = (y_scores >= umbral_para_precisión_90)
```

Verifiquemos la precisión y la recordación de estas predicciones:

```
>>> puntuación_de_precisión(y_train_5, y_train_pred_90)
0.9000345901072293
>>> recuperación_con_precisión_de_90 = puntuación_de_recuperación(y_train_5,
y_train_pred_90) >>>
recuperación_con_precisión_de_90 0.4799852425751706
```

¡Genial, tienes un clasificador con una precisión del 90%! Como puedes ver, es bastante fácil crear un clasificador con prácticamente cualquier precisión: simplemente establece un umbral lo suficientemente alto y listo. Pero espera, no te apresures: un clasificador de alta precisión no es muy útil si su recuperación es demasiado baja. Para muchas aplicaciones, una recuperación del 48% no sería ideal.

CONSEJO

Si alguien dice: "Alcancemos una precisión del 99%", deberías preguntar: "¿Con qué nivel de recuperación?"

# La curva ROC

La curva ROC (curva característica operativa del receptor ) es otra herramienta común en los clasificadores binarios. Es muy similar a la curva de precisión/recuperación, pero en lugar de representar gráficamente la precisión frente a la recuperación, la curva ROC representa gráficamente la tasa de verdaderos positivos (también conocida como recuperación) frente a la tasa de falsos positivos (FPR). La FPR (también llamada caída) es la proporción de casos negativos que se clasifican incorrectamente como positivos. Es igual a 1 (la tasa de verdaderos negativos , TNR), que es la proporción de casos negativos que se clasifican correctamente como negativos. La TNR también se denomina especificidad. Por lo tanto, la curva ROC representa la sensibilidad (recuperación) frente a 1 ( especificidad).

Para trazar la curva ROC, primero utilice la función roc_curve() para calcular la TPR y FPR para varios valores umbral:

```
desde sklearn.metrics importar roc_curve

fpr, tpr, umbrales = roc_curve(y_train_5, y_scores)
```

Luego, puede graficar el FPR frente al TPR usando Matplotlib. El siguiente código genera el gráfico de la Figura 3-7. Para encontrar el punto que corresponde al 90% de precisión, necesitamos buscar el índice del umbral deseado. Dado que los umbrales se enumeran en orden decreciente en este caso, usamos <= en lugar de >= en la primera línea:

```
idx_para_umbral_a_90 = (umbrales <= umbral_para_precisión_de_90).argmax()
tpr_90, fpr_90 = tpr[idx_para_umbral_a_90], fpr[idx_para_umbral_a_90]

plt.plot(fpr, tpr, linewidth=2, label=" Curva ROC")
plt.plot([0, 1], [0, 1], 'k:', label="Curva ROC del clasificador aleatorio")
plt.plot([fpr_90], [tpr_90], "ko", label="Umbral para una precisión del 90%)
[...] # embellecer la figura: agregar etiquetas, cuadrícula, leyenda,
flecha y texto plt.show()
```

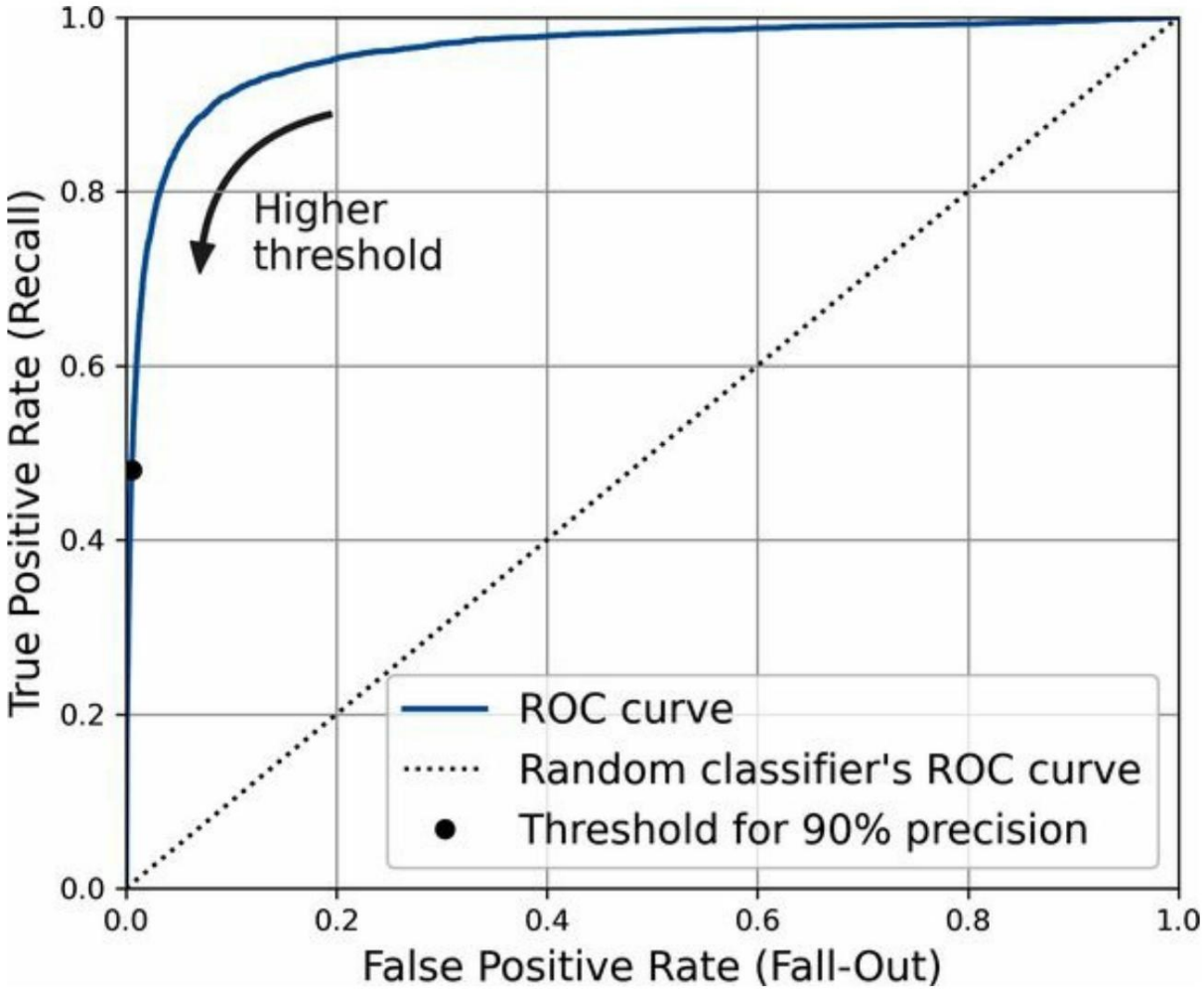


Figura 3-7. Curva ROC que representa la tasa de falsos positivos frente a la tasa de verdaderos positivos para todos los umbrales posibles; el círculo negro resalta la proporción seleccionada (con una precisión del 90 % y una recuperación del 48 %).

Nuevamente, existe una disyuntiva: cuanto mayor sea la tasa de recuperación (TPR), más falsos positivos (FPR) producirá el clasificador. La línea punteada representa la curva ROC de un clasificador puramente aleatorio; un buen clasificador se mantiene lo más alejado posible de esa línea (hacia la esquina superior izquierda).

Una forma de comparar clasificadores es medir el área bajo la curva (AUC). Un clasificador perfecto tendrá un AUC ROC igual a 1, mientras que un clasificador puramente aleatorio tendrá un AUC ROC igual a 0,5. Scikit-Learn proporciona una función para estimar el AUC ROC:

```
>>> de sklearn.metrics importar roc_auc_score >>>
roc_auc_score(y_train_5, y_scores)
0.9604938554008616
```

> CONSEJO
>
> Dado que la curva ROC es tan similar a la curva de precisión/recuperación (PR), quizás se pregunte cómo decidir cuál usar. Como regla general, debería preferir la curva PR cuando la clase positiva sea poco frecuente o cuando le importen más los falsos positivos que los falsos negativos. De lo contrario, utilice la curva ROC. Por ejemplo, al observar la curva ROC anterior (y la puntuación del AUC ROC), podría pensar que el clasificador es realmente bueno. Pero esto se debe principalmente a que hay pocos positivos (5) en comparación con los negativos (no 5). En cambio, la curva PR deja claro que el clasificador tiene margen de mejora: la curva podría estar realmente más cerca de la esquina superior derecha (véase la Figura 3-6 de nuevo).

Creemos ahora un RandomForestClassifier, cuya curva PR y puntuación F podemos comparar con las del SGDClassifier:

```
de sklearn.ensemble importar RandomForestClassifier

bosque_clf = ClasificadorForestAleatorio(estado_aleatorio=42)
```

La función precision_recall_curve() espera etiquetas y puntuaciones para cada instancia, por lo que necesitamos entrenar el clasificador de bosque aleatorio y hacer que asigne una puntuación a cada instancia. Sin embargo, la clase RandomForestClassifier no tiene un método decision_function() debido a su funcionamiento (lo abordaremos en el Capítulo 7). Afortunadamente, cuenta con un método predict_proba() que devuelve las probabilidades de clase para cada instancia, y podemos usar la probabilidad de la clase positiva como puntuación, por lo que funcionará correctamente. Podemos[4] llamar a la función cross_val_predict() para entrenar el RandomForestClassifier mediante validación cruzada y hacer que prediga las probabilidades de clase para cada imagen, como se indica a continuación:

```
y_probas_bosque = cross_val_predict(bosque_clf, X_tren, y_tren_5, cv=3,
                                    método="predict_proba")
```

Veamos las probabilidades de clase para las dos primeras imágenes del conjunto de entrenamiento:

```
>>> y_probas_forest[:2]
matriz([[0.11, 0.89],
       [0.99, 0.01]])
```

El modelo predice que la primera imagen es positiva con un 89% de probabilidad y

Predice que la segunda imagen es negativa con un 99 % de probabilidad. Como cada imagen es positiva o negativa, las probabilidades en cada fila suman el 100 %.

> ADVERTENCIA
>
> Estas son probabilidades estimadas , no reales. Por ejemplo, si se observan todas las imágenes que el modelo clasificó como positivas con una probabilidad estimada de entre el 50 % y el 60 %, aproximadamente el 94 % son realmente positivas. Por lo tanto, las probabilidades estimadas del modelo fueron demasiado bajas en este caso, pero los modelos también pueden ser demasiado confiados. El paquete sklearn.calibration contiene herramientas para calibrar las probabilidades estimadas y aproximarlas considerablemente a las reales. Consulte la sección de material adicional en el cuaderno de este capítulo para obtener más detalles.

La segunda columna contiene las probabilidades estimadas para la clase positiva, así que pasémoslas a la función precision_recall_curve():

```
y_scores_forest = y_probas_forest[:, 1]
precisiones_forest, recuperaciones_forest, umbrales_forest = precisión_recuperación_curva(
    y_train_5, y_scores_forest)
```

Ahora estamos listos para trazar la curva PR. Es útil trazar también la primera curva PR para compararlas (Figura 3-8):

```
plt.plot(recalls_forest, precisions_forest, "b-", linewidth=2,
         label="Random Forest")
plt.plot(recalls, precisions, "--", linewidth=2, label="SGD") [...] #
embellecer la figura: agregar etiquetas, cuadrícula y leyenda
plt.show()
```

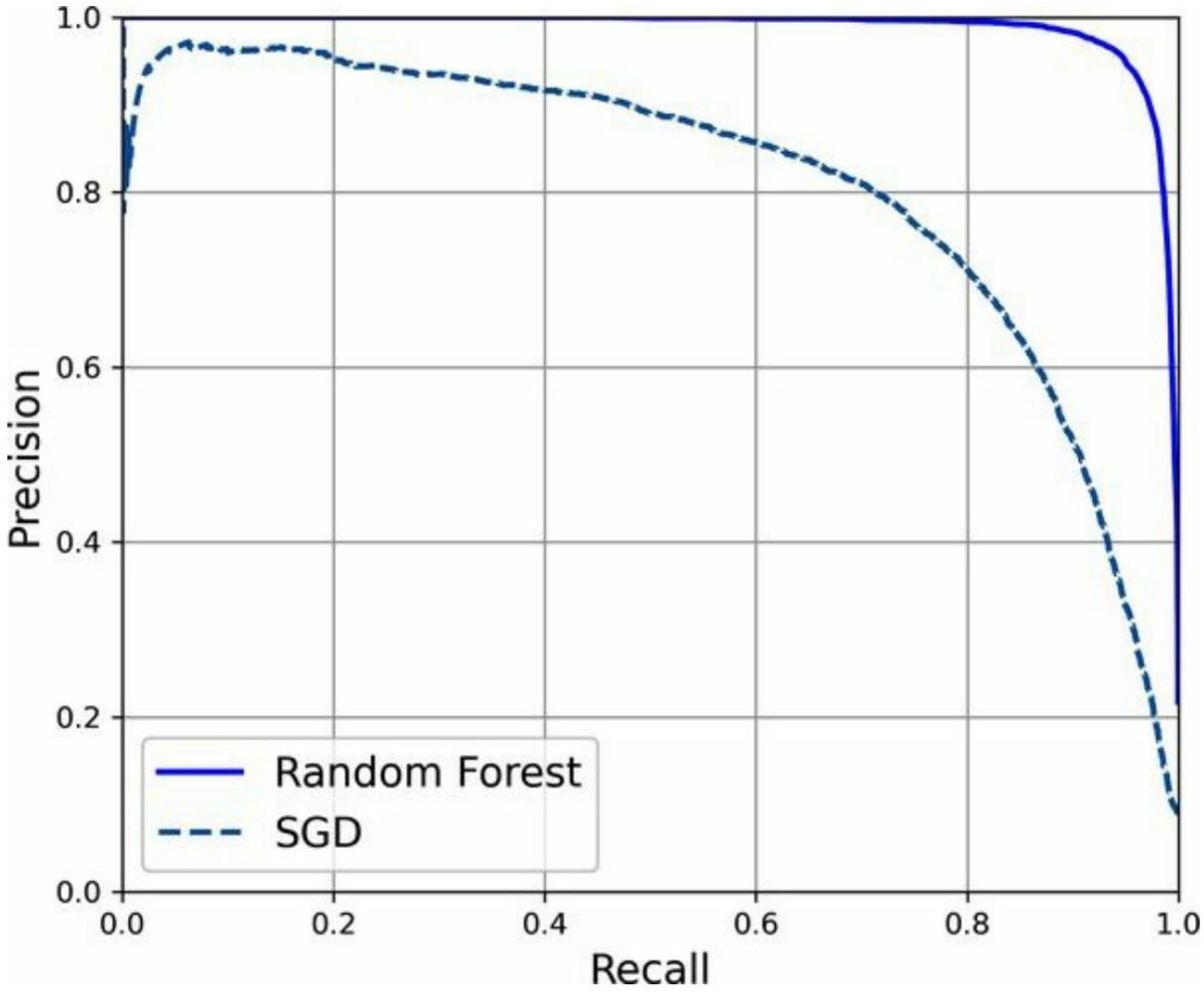


Figura 3-8. Comparación de curvas PR: el clasificador de bosque aleatorio es superior al clasificador SGD porque su curva PR está mucho más cerca de la esquina superior derecha y tiene un AUC mayor.

Como se puede observar en la Figura 3-8, la curva PR del RandomForestClassifier es mucho mejor que la del SGDClassifier: se acerca mucho más a la esquina superior derecha. Sus puntuaciones F y ROC AUC también son significativamente mejores:

```
>>> y_train_pred_forest = y_probas_forest[:, 1] >= 0.5 # probabilidad positiva ≥ 50% >>>
f1_score(y_train_5, y_pred_forest)
0.9242275142688446
>>> roc_auc_score(y_train_5, y_scores_forest)
0.9983436731328145
```

Intenta medir la precisión y la recuperación: deberías obtener un 99,1 % de precisión y un 86,6 % de recuperación. ¡Nada mal!

Ahora ya sabes cómo entrenar clasificadores binarios, elige la métrica adecuada

Para su tarea, evalúe sus clasificadores mediante validación cruzada, seleccione el equilibrio entre precisión y recuperación que mejor se adapte a sus necesidades y utilice diversas métricas y curvas para comparar distintos modelos. Está listo para intentar detectar más allá de los 5.

CAPÍTULO 2

# Aprendizaje supervisado

Como mencionamos anteriormente, el aprendizaje automático supervisado es uno de los tipos de aprendizaje automático más utilizados y exitosos. En este capítulo, describiremos el aprendizaje supervisado con más detalle y explicaremos varios algoritmos populares. Ya vimos una aplicación del aprendizaje automático supervisado en el capítulo 1: la clasificación de flores de iris en varias especies mediante mediciones físicas de las flores.

Recuerde que el aprendizaje supervisado se utiliza siempre que queremos predecir un resultado determinado a partir de una entrada dada, y tenemos ejemplos de pares de entrada/salida. Construimos un modelo de aprendizaje automático a partir de estos pares de entrada/salida, que conforman nuestro conjunto de entrenamiento. Nuestro objetivo es realizar predicciones precisas para datos nuevos e inéditos. El aprendizaje supervisado suele requerir esfuerzo humano para construir el conjunto de entrenamiento, pero posteriormente automatiza y, a menudo, acelera una tarea que de otro modo sería laboriosa o inviable.

## Clasificación y regresión

Hay dos tipos principales de problemas de aprendizaje automático supervisado, llamados clasificación y regresión.

En la clasificación, el objetivo es predecir una etiqueta de clase, que es una opción entre una lista predefinida de posibilidades. En el capítulo 1, usamos el ejemplo de clasificar iris en una de tres especies posibles. La clasificación a veces se divide en clasificación binaria, que es el caso especial de distinguir entre exactamente dos clases, y clasificación multiclase, que es la clasificación entre más de dos clases. Se puede pensar en la clasificación binaria como intentar responder a una pregunta de sí o no. Clasificar correos electrónicos como spam o no spam es un ejemplo de un problema de clasificación binaria. En esta tarea de clasificación binaria, la pregunta de sí o no sería "¿Es este correo electrónico spam?".

En la clasificación binaria, solemos hablar de una clase como positiva y la otra como negativa. En este caso, positivo no representa beneficio o valor, sino el objeto de estudio. Por lo tanto, al buscar spam, «positivo» podría referirse a la clase de spam. Cuál de las dos clases se considera positiva suele ser una cuestión subjetiva y específica del dominio.

El ejemplo del iris, por otro lado, es un ejemplo de un problema de clasificación multiclase. Otro ejemplo es predecir el idioma de un sitio web a partir del texto que contiene. En este caso, las clases serían una lista predefinida de posibles idiomas.

En las tareas de regresión, el objetivo es predecir un número continuo o, en términos de programación, un número de punto flotante (o un número real en términos matemáticos). Predecir los ingresos anuales de una persona a partir de su educación, edad y lugar de residencia es un ejemplo de tarea de regresión. Al predecir los ingresos, el valor predicho es una cantidad y puede ser cualquier número dentro de un rango dado. Otro ejemplo de tarea de regresión es predecir el rendimiento de una granja de maíz dados atributos como los rendimientos anteriores, el clima y el número de empleados que trabajan en la granja. El rendimiento, a su vez, puede ser un número arbitrario.

Una forma sencilla de distinguir entre tareas de clasificación y regresión es preguntarse si existe algún tipo de continuidad en el resultado. Si existe continuidad entre los posibles resultados, entonces el problema es de regresión. Consideremos la predicción de ingresos anuales. Existe una clara continuidad en el resultado. Que una persona gane $40,000 o $40,001 al año no supone una diferencia tangible, aunque se trate de cantidades de dinero diferentes; si nuestro algoritmo predice $39,999 o $40,001 cuando debería haber predicho $40,000, no nos importa demasiado.

En cambio, para reconocer el idioma de un sitio web (que es un problema de clasificación), no importa el grado. Un sitio web está en un idioma o en otro. No hay continuidad entre idiomas, y no existe ningún idioma intermedio entre el inglés y el francés.

## Generalización, sobreajuste y subajuste

En el aprendizaje supervisado, buscamos construir un modelo con los datos de entrenamiento y, posteriormente, poder realizar predicciones precisas con datos nuevos e inéditos que tengan las mismas características que el conjunto de entrenamiento utilizado. Si un modelo puede realizar predicciones precisas con datos inéditos, decimos que puede generalizar del conjunto de entrenamiento al conjunto de prueba. Buscamos construir un modelo que pueda generalizar con la mayor precisión posible.

1 Pedimos a los lingüistas que disculpen la presentación simplificada de las lenguas como entidades distintas y fijas.

Generalmente construimos un modelo de tal manera que pueda hacer predicciones precisas sobre el
conjunto de entrenamiento. Si los conjuntos de entrenamiento y prueba tienen suficiente en común, esperamos que
modelo para que también sea preciso en el conjunto de prueba. Sin embargo, hay algunos casos en los que esto
puede salir mal. Por ejemplo, si nos permitimos construir modelos muy complejos,
Siempre podemos ser tan precisos como queramos en el conjunto de entrenamiento.

Veamos un ejemplo inventado para ilustrar este punto. Digamos que un científico de datos novato...
Un experto quiere predecir si un cliente comprará un barco, dados los registros de compras anteriores.
compradores de barcos y clientes que sabemos que no están interesados en comprar un barco.2 El
El objetivo es enviar correos electrónicos promocionales a personas que probablemente realmente realicen una compra.
comprar, pero no molestar a aquellos clientes que no estarán interesados.

Supongamos que tenemos los registros de clientes que se muestran en la Tabla 2-1.

### Tabla 2-1. Ejemplo de datos sobre clientes

| Edad | Número de coches que posee | Tiene casa | Número de hijos | Estado civil | Tiene un perro | Compró un barco |
|---|---|---|---|---|---|---|
| 66 | 1 | Sí | 2 | viudo | No | Sí |
| 52 | 2 | Sí | 3 | casado | No | Sí |
| 22 | 0 | No | 0 | casado | Sí | No |
| 25 | 1 | No | 1 | soltero | No | No |
| 44 | 0 | No | 2 | divorciado | Sí | No |
| 39 | 1 | Sí | 2 | casado | Sí | No |
| 26 | 1 | No | 2 | soltero | No | No |
| 40 | 3 | Sí | 1 | casado | Sí | No |
| 53 | 2 | Sí | 2 | divorciado | No | Sí |
| 64 | 2 | Sí | 3 | divorciado | No | No |
| 58 | 2 | Sí | 2 | casado | Sí | Sí |
| 33 | 1 | No | 1 | soltero | No | No |

Después de observar los datos durante un tiempo, nuestro científico de datos novato llega a la siguiente conclusión:
Regla siguiente: “Si el cliente es mayor de 45 años y tiene menos de 3 hijos o no está
divorciados, entonces quieren comprar un barco”. Cuando se le preguntó qué tan bien funciona esta regla suya,
Nuestro científico de datos responde: "¡Es 100 por ciento preciso!" Y, de hecho, en los datos que son
En la tabla, la regla es perfectamente precisa. Hay muchas reglas posibles que podríamos...
Se me ocurre algo que explicaría perfectamente si alguien en este conjunto de datos quiere comprar un
barco. Ninguna edad aparece dos veces en los datos, por lo que podríamos decir que las personas tienen 66, 52, 53 o

2 En el mundo real, este es un problema complicado. Si bien sabemos que los demás clientes no han comprado un...
Todavía no han comprado un barco de nosotros, es posible que hayan comprado uno a otra persona o que todavía estén ahorrando y planeen comprar uno.
Uno en el futuro.

58 años quiere comprar un barco, mientras que todos los demás no. Si bien podemos crear muchas reglas que funcionen bien con estos datos, recuerde que no nos interesa hacer predicciones para este conjunto de datos; ya conocemos las respuestas para estos clientes. Queremos saber si es probable que los nuevos clientes compren un barco. Por lo tanto, queremos encontrar una regla que funcione bien para los nuevos clientes, y lograr una precisión del 100 por ciento en el conjunto de entrenamiento no nos ayuda en ese aspecto. Es posible que no esperemos que la regla que ideó nuestro científico de datos funcione muy bien con los nuevos clientes. Parece demasiado compleja y está respaldada por muy pocos datos. Por ejemplo, la parte "o no está divorciado" de la regla depende de un solo cliente.

La única medida del buen rendimiento de un algoritmo con datos nuevos es su evaluación en el conjunto de prueba. Sin embargo, intuitivamente[3] esperamos que los modelos simples se generalicen mejor a nuevos datos. Si la regla fuera «Las personas mayores de 50 años quieren comprar un barco», y esto explicara el comportamiento de todos los clientes, confiaríamos más en ella que en la regla que incluye hijos y estado civil, además de la edad. Por lo tanto, siempre buscamos el modelo más simple. Construir un modelo demasiado complejo para la cantidad de información disponible, como hizo nuestro científico de datos novato, se denomina sobreajuste. El sobreajuste se produce cuando se ajusta un modelo demasiado estrechamente a las particularidades del conjunto de entrenamiento y se obtiene un modelo que funciona bien en dicho conjunto, pero que no se puede generalizar a nuevos datos. Por otro lado, si el modelo es demasiado simple (por ejemplo, «Todos los que poseen una casa compran un barco»), es posible que no se puedan capturar todos los aspectos y la variabilidad de los datos, y el modelo tendrá un rendimiento deficiente incluso en el conjunto de entrenamiento. Elegir un modelo demasiado simple se llama underfitting.

Cuanto más complejo sea nuestro modelo, mejor podremos predecir con los datos de entrenamiento. Sin embargo, si se vuelve demasiado complejo, nos centraremos demasiado en cada punto de datos individual del conjunto de entrenamiento, y el modelo no se generalizará correctamente a los nuevos datos.

Existe un punto óptimo entre ambos que producirá el mejor rendimiento de generalización.
Este es el modelo que queremos encontrar.

El equilibrio entre sobreajuste y subajuste se ilustra en la Figura 2-1.

3 Y también es demostrable, con los cálculos correctos.

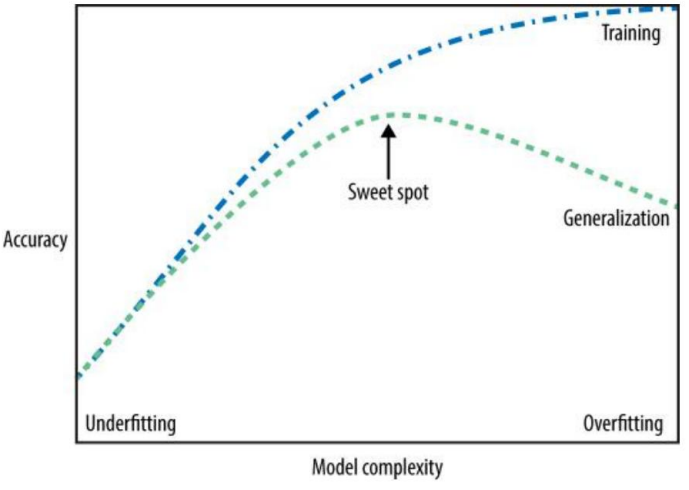


Figura 2-1. Relación entre la complejidad del modelo y la precisión del entrenamiento y las pruebas

Relación de la complejidad del modelo con el tamaño del conjunto de

datos Es importante tener en cuenta que la complejidad del modelo está íntimamente ligada a la variación de las entradas contenidas en su conjunto de datos de entrenamiento: cuanto mayor sea la variedad de puntos de datos que contenga su conjunto de datos, más complejo será el modelo que pueda utilizar sin sobreajustar. Por lo general, recopilar más puntos de datos producirá más variedad, por lo que los conjuntos de datos más grandes permiten construir modelos más complejos. Sin embargo, simplemente duplicar los mismos puntos de datos o recopilar datos muy similares no ayudará.

Volviendo al ejemplo de la venta de barcos, si viéramos 10.000 filas más de datos de clientes, y todas ellas cumplieran con la regla “Si el cliente es mayor de 45 años, tiene menos de 3 hijos o no está divorciado, entonces quiere comprar un barco”, sería mucho más probable que creyéramos que se trataba de una buena regla que cuando se desarrolló utilizando solo las 12 filas de la Tabla 2-1.

Disponer de más datos y construir modelos adecuadamente más complejos puede ser muy beneficioso para las tareas de aprendizaje supervisado. En este libro, nos centraremos en trabajar con conjuntos de datos de tamaño fijo. En la práctica, a menudo se puede decidir cuántos datos recopilar, lo que puede ser más beneficioso que ajustar y perfeccionar el modelo. Nunca subestimes el poder de contar con más datos.

## Algoritmos de aprendizaje automático supervisado

Ahora revisaremos los algoritmos de aprendizaje automático más populares y explicaremos cómo aprenden de los datos y cómo realizan predicciones. También analizaremos cómo se aplica el concepto de complejidad del modelo a cada uno de estos modelos y ofreceremos una visión general.

Vista de cómo cada algoritmo construye un modelo. Examinaremos las fortalezas y debilidades de cada algoritmo y a qué tipo de datos se aplican mejor. También explicaremos el significado de los parámetros y opciones más importantes. Muchos algoritmos tienen una clasificación y una variante de regresión, y describiremos ambas.

No es necesario leer las descripciones de cada algoritmo en detalle, pero comprender los modelos le permitirá comprender mejor las diferentes maneras en que funcionan los algoritmos de aprendizaje automático. Este capítulo también puede usarse como guía de referencia y puede consultarlo si tiene dudas sobre el funcionamiento de alguno de los algoritmos.

Algunos conjuntos de datos de

muestra. Utilizaremos varios conjuntos de datos para ilustrar los diferentes algoritmos. Algunos conjuntos de datos serán pequeños y sintéticos (es decir, inventados), diseñados para destacar aspectos específicos de los algoritmos. Otros conjuntos de datos serán ejemplos extensos del mundo real.

Un ejemplo de un conjunto de datos sintético de clasificación de dos clases es el conjunto de datos Forge , que tiene dos características. El siguiente código crea un diagrama de dispersión (Figura 2-2) que visualiza todos los puntos de datos de este conjunto. El gráfico muestra la primera característica en el eje x y la segunda en el eje y. Como siempre ocurre en los diagramas de dispersión, cada punto se representa como un punto. El color y la forma del punto indican su clase:

En[2]:

```
# generar el conjunto
de datos X, y = mglearn.datasets.make_forge()
# trazar el
conjunto de datos mglearn.discrete_scatter(X[:, 0], X[:,
1], y) plt.legend(["Clase 0", "Clase 1"], loc=4)
plt.xlabel("Primera característica")
plt.ylabel("Segunda característica")
print("X.shape: {}".format(X.shape))
```

Salida[2]:

```
Forma de X: (26, 2)
```

Discutirlos todos está más allá del alcance del libro, por lo que le remitimos a la documentación de scikit-learn . Para más detalles.

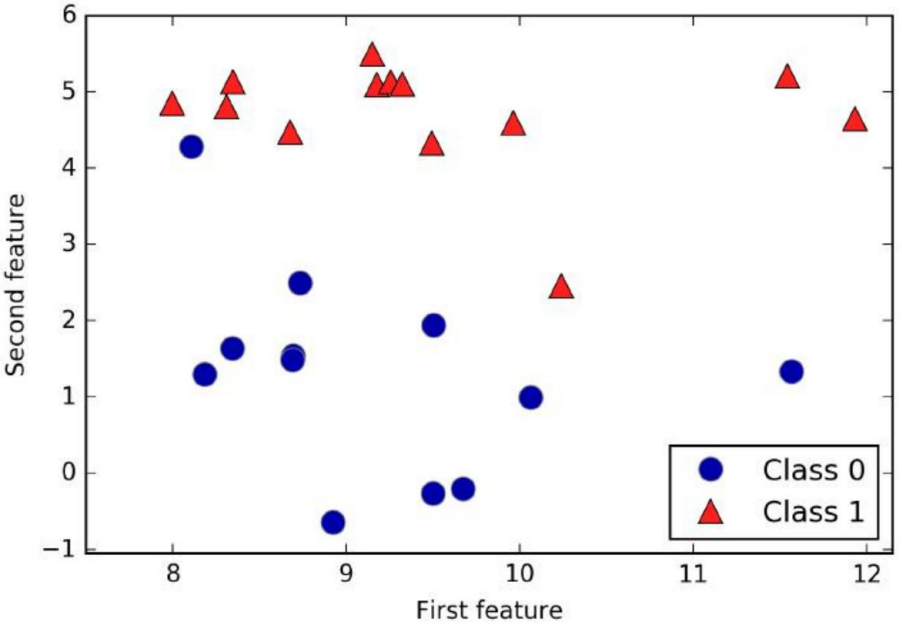


Figura 2-2. Diagrama de dispersión del conjunto de datos de forja

Como puede ver en X.shape, este conjunto de datos consta de 26 puntos de datos, con 2 características.

Para ilustrar los algoritmos de regresión, utilizaremos el conjunto de datos sintéticos de olas . Este conjunto de datos tiene una única característica de entrada y una variable objetivo continua (o respuesta) que queremos modelar. El gráfico creado aquí (Figura 2-3) muestra la característica única en el eje x y el objetivo de regresión (la salida) en el eje y:

En[3]:

```
X, y = mglearn.datasets.make_wave(n_muestras=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Característica")
plt.ylabel("Objetivo")
```

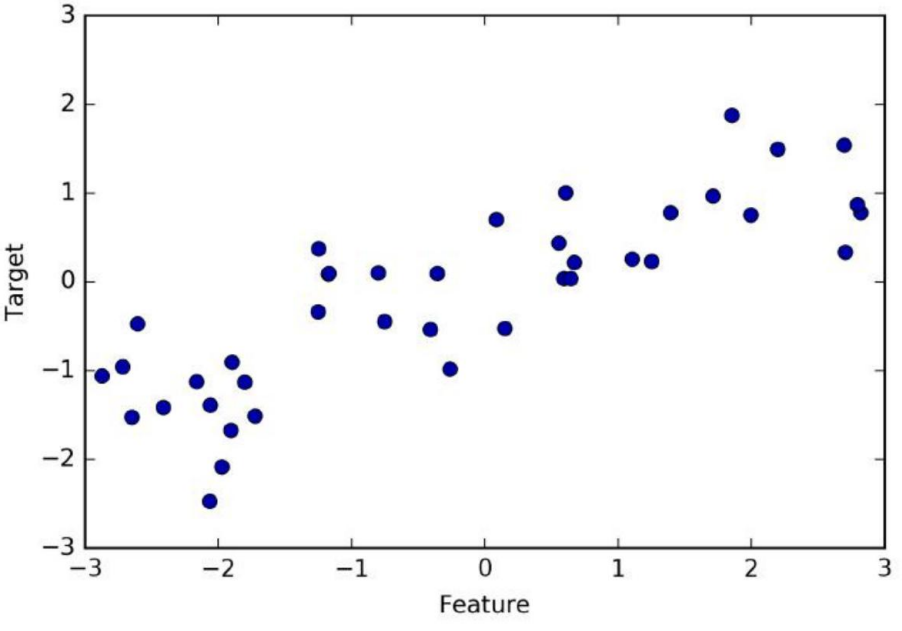


Figura 2-3. Gráfico del conjunto de datos de ondas, con el eje x mostrando la característica y el eje y mostrando el objetivo de regresión.

Utilizamos estos conjuntos de datos muy simples y de baja dimensión porque podemos visualizarlos fácilmente: una página impresa tiene dos dimensiones, por lo que es difícil mostrar datos con más de dos características. Cualquier intuición derivada de conjuntos de datos con pocas características (también llamados conjuntos de datos de baja dimensión) podría no ser válida en conjuntos de datos con muchas características (conjuntos de datos de alta dimensión). Teniendo esto en cuenta, la inspección de algoritmos en conjuntos de datos de baja dimensión puede ser muy instructiva.

Complementaremos estos pequeños conjuntos de datos sintéticos con dos conjuntos de datos reales incluidos en scikit-learn. Uno es el conjunto de datos de cáncer de mama de Wisconsin (cáncer, para abreviar), que registra mediciones clínicas de tumores de cáncer de mama. Cada tumor se etiqueta como "benigno" (para tumores inofensivos) o "maligno" (para tumores cancerosos), y la tarea consiste en aprender a predecir si un tumor es maligno con base en las mediciones del tejido.

Los datos se pueden cargar utilizando la función load_breast_cancer de scikit-learn:

En[4]:

```
de sklearn.datasets importar cargar_cáncer_de_mama
cáncer = cargar_cáncer_de_mama()
imprimir("cáncer.claves(): \n{}".formato(cáncer.claves()))
```

Salida[4]:

```
cancer.keys():
dict_keys(['nombres_de_características', 'datos', 'DESCR', 'objetivo', 'nombres_de_objetivo'])
```

Los conjuntos de datos incluidos en scikit-learn suelen almacenarse como objetos Bunch , que contienen información sobre el conjunto de datos, además de los datos en sí. Lo único que necesita saber sobre los objetos Bunch es que se comportan como diccionarios, con la ventaja adicional de que se puede acceder a los valores mediante un punto (como en ` bundle.key` en lugar de `bundle['key']` ).

El conjunto de datos consta de 569 puntos de datos, con 30 características cada uno:

En[5]:

```
print("Forma de los datos del cáncer: {}".format(cancer.data.shape))
```

Salida[5]:

```
Forma de los datos del cáncer: (569, 30)
```

De estos 569 puntos de datos, 212 están etiquetados como malignos y 357 como benignos:

En[6]:

```
print("Recuento de muestras por clase:\n{}".format(
      {n: v para n, v en zip(cáncer.objetivo_nombres, np.bincount(cáncer.objetivo))}))
```

Salida[6]:

```
Recuento de muestras por
clase: {'benigno': 357, 'maligno': 212}
```

Para obtener una descripción del significado semántico de cada característica, podemos echar un vistazo al atributo feature_names :

En[7]:

```
print(" Nombres de las características:\n{}".format(cancer.feature_names))
```

Salida[7]:

```
Nombres de las funciones:
['radio medio' 'textura media' 'perímetro medio' 'área media' 'suavidad media'
  'compacidad media' 'concavidad media' 'puntos cóncavos medios'
  'simetría media' 'dimensión fractal media' 'error de radio' 'error de textura' 'error
  de perímetro' 'error de área' 'error de suavidad' 'error de compacidad' 'error de
  concavidad' 'error de puntos cóncavos' 'error de simetría' 'error de
  dimensión fractal' 'peor radio' 'peor textura' 'peor perímetro' 'peor área' 'peor suavidad'
  'peor compacidad' 'peor concavidad' 'peores puntos cóncavos' 'peor simetría'
  'peor dimensión fractal']
```

Si está interesado, puede obtener más información sobre los datos leyendo cancer.DESCR .

También utilizaremos un conjunto de datos de regresión del mundo real: el conjunto de datos de vivienda de Boston. La tarea asociada a este conjunto de datos es predecir el valor medio de las viviendas en varios barrios de Boston en la década de 1970, utilizando información como la tasa de delincuencia, la proximidad al río Charles, la accesibilidad a las carreteras, etc. El conjunto de datos contiene 506 puntos de datos, descritos por 13 características:

En[8]:

```
de sklearn.datasets importar load_boston boston =
load_boston() imprimir(" Forma
de los datos: {}".format(boston.data.shape))
```

Salida[8]:

```
Forma de los datos: (506, 13)
```

Nuevamente, puede obtener más información sobre el conjunto de datos leyendo el atributo DESCR de Boston. Para nuestros propósitos, ampliaremos este conjunto de datos considerando no solo estas 13 mediciones como características de entrada, sino también todos los productos (también llamados interacciones) entre las características. En otras palabras, no solo consideraremos la tasa de delincuencia y la accesibilidad a las carreteras como características, sino también el producto de la tasa de delincuencia y la accesibilidad a las carreteras. Incluir características derivadas como estas se denomina ingeniería de características, que analizaremos con más detalle en el Capítulo 4. Este conjunto de datos derivado se puede cargar mediante la función load_extended_boston :

En[9]:

```
X, y = mglearn.datasets.load_extended_boston() print("X.shape:
{}".format(X.shape))
```

Salida[9]:

```
Forma de X: (506, 104)
```

Las 104 características resultantes son las 13 características originales junto con las 91 combinaciones posibles de dos características dentro de esas 13,5

Utilizaremos estos conjuntos de datos para explicar e ilustrar las propiedades de los diferentes algoritmos de aprendizaje automático. Pero por ahora, analicemos los algoritmos en sí.
Primero, revisaremos el algoritmo de k vecinos más cercanos (k-NN) que vimos en el capítulo anterior.

5 Esto se llama coeficiente binomial, que es el número de combinaciones de k elementos que se pueden seleccionar de un conjunto de n elementos. A menudo esto se escribe como $\binom{n}{k}$ y se pronuncia como “n elige k”—en este caso, “13 elige 2”.

El algoritmo k -NN es

posiblemente el algoritmo de aprendizaje automático más simple. Construir el modelo consiste únicamente en almacenar el conjunto de datos de entrenamiento. Para predecir un nuevo punto de datos, el algoritmo encuentra los puntos de datos más cercanos en el conjunto de datos de entrenamiento: sus "vecinos más cercanos".

Clasificación de k-vecinos. En

su versión más simple, el algoritmo k-NN solo considera un vecino más cercano, que es el punto de datos de entrenamiento más cercano al punto para el que queremos realizar una predicción. La predicción es, entonces, simplemente el resultado conocido para este punto de entrenamiento. La Figura 2-4 ilustra esto para el caso de la clasificación en el conjunto de datos de Forge :

En[10]:

```
mglearn.plots.plot_knn_classification(n_vecinos=1)
```

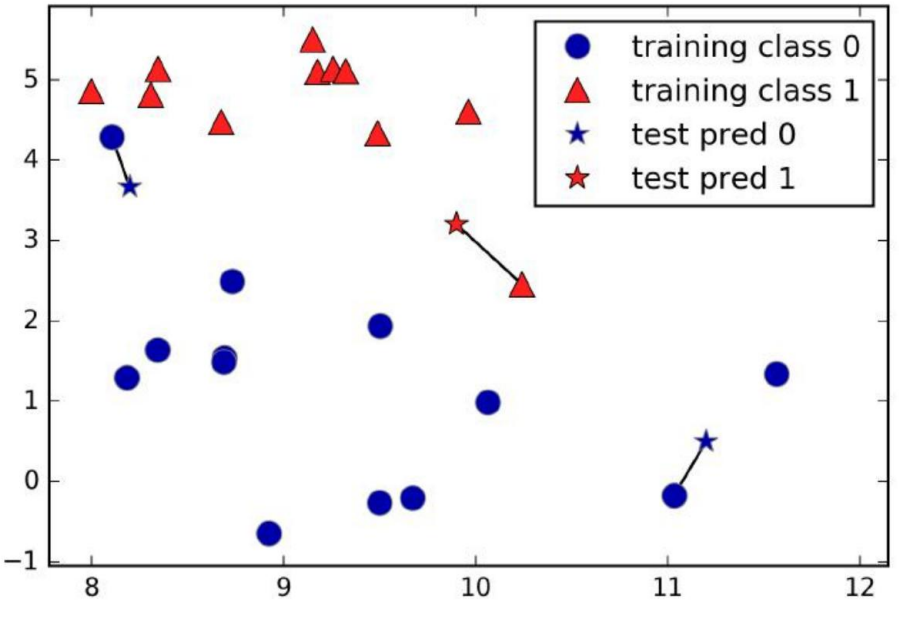


Figura 2-4. Predicciones realizadas por el modelo de un vecino más cercano en el conjunto de datos de forja

Aquí, añadimos tres nuevos puntos de datos, representados por estrellas. Para cada uno, marcamos el punto más cercano en el conjunto de entrenamiento. La predicción del algoritmo de un vecino más cercano es la etiqueta de ese punto (mostrada por el color de la cruz).

En lugar de considerar solo al vecino más cercano, también podemos considerar un número arbitrario, k, de vecinos. De ahí proviene el nombre del algoritmo de los k vecinos más cercanos. Al considerar más de un vecino, utilizamos la votación para asignar una etiqueta. Esto significa que, para cada punto de prueba, contamos cuántos vecinos pertenecen a la clase 0 y cuántos a la clase 1. A continuación, asignamos la clase más frecuente: es decir, la clase mayoritaria entre los k vecinos más cercanos. El siguiente ejemplo (Figura 2-5) utiliza los tres vecinos más cercanos:

En[11]:

```
mglearn.plots.plot_knn_classification(n_vecinos=3)
```

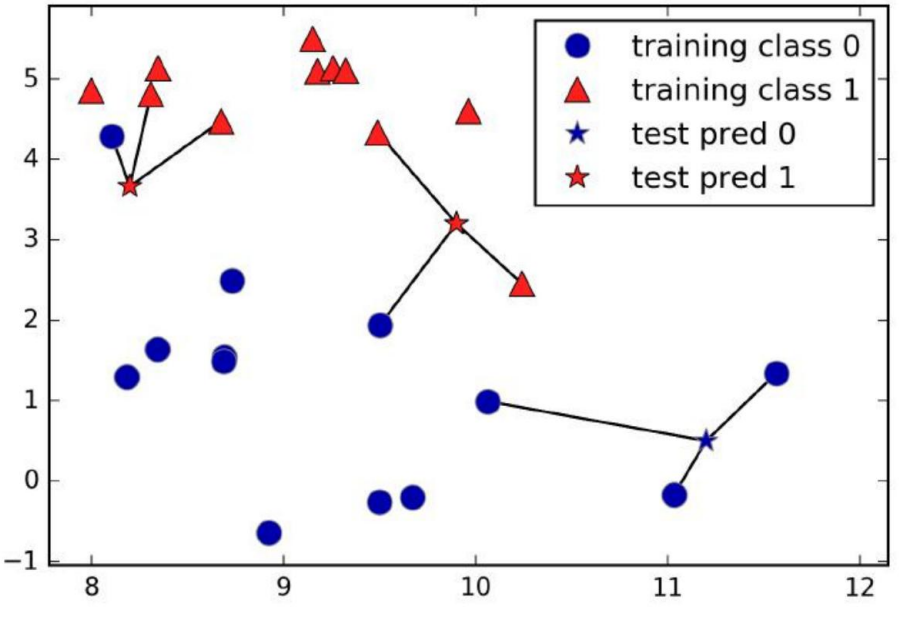


Figura 2-5. Predicciones realizadas por el modelo de los tres vecinos más próximos en el conjunto de datos de forja.

Nuevamente, la predicción se muestra con el color de la cruz. Se puede observar que la predicción para el nuevo punto de datos en la esquina superior izquierda no coincide con la predicción obtenida cuando usamos solo un vecino.

Aunque esta ilustración corresponde a un problema de clasificación binaria, este método puede aplicarse a conjuntos de datos con cualquier número de clases. Para más clases, contamos cuántos vecinos pertenecen a cada clase y, de nuevo, predecimos la clase más común.

Ahora veamos cómo podemos aplicar el algoritmo de k vecinos más cercanos con scikit-learn. Primero, dividimos nuestros datos en un conjunto de entrenamiento y uno de prueba para evaluar el rendimiento de la generalización, como se explicó en el Capítulo 1.

En[12]:

```
desde sklearn.model_selection importar train_test_split X, y =
mglearn.datasets.make_forge()

X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = tren_prueba_división(X, y, estado_aleatorio=0)
```

A continuación, importamos e instanciamos la clase. Aquí es cuando podemos establecer parámetros, como el número de vecinos a usar. Aquí, lo establecemos en 3:

En[13]:

```
de sklearn.neighbors importar KNeighborsClassifier clf =
KNeighborsClassifier(n_neighbors=3)
```

Ahora, ajustamos el clasificador usando el conjunto de entrenamiento. Para KNeighborsClassifier , esto implica almacenar el conjunto de datos para poder calcular los vecinos durante la predicción:

En[14]:

```
clf.fit(X_tren, y_tren)
```

Para realizar predicciones sobre los datos de prueba, utilizamos el método de predicción . Para cada punto de datos del conjunto de prueba, este método calcula sus vecinos más cercanos en el conjunto de entrenamiento y encuentra la clase más común entre ellos:

En[15]:

```
print(" Predicciones del conjunto de pruebas: {}".format(clf.predict(X_test)))
```

Salida[15]:

```
Predicciones del conjunto de pruebas: [1 0 1 0 1 0 0]
```

Para evaluar qué tan bien se generaliza nuestro modelo, podemos llamar al método de puntuación con los datos de prueba junto con las etiquetas de prueba:

En[16]:

```
print(" Precisión del conjunto de pruebas: {:.2f}".format(clf.score(X_test, y_test)))
```

Salida[16]:

```
Precisión del conjunto de prueba: 0,86
```

Vemos que nuestro modelo tiene una precisión de aproximadamente el 86%, lo que significa que el modelo predijo la clase correctamente para el 86% de las muestras en el conjunto de datos de prueba.

Análisis de KNeighborsClassifier. Para

conjuntos de datos bidimensionales, también podemos ilustrar la predicción para todos los posibles puntos de prueba en el plano xy. Coloreamos el plano según la clase que se asignaría a un punto en esta región. Esto nos permite visualizar el límite de decisión, que es la división entre el punto donde el algoritmo asigna la clase 0 y el punto donde asigna la clase 1.

El siguiente código produce las visualizaciones de los límites de decisión para uno, tres y nueve vecinos que se muestran en la Figura 2-6:

En[17]:

```
fig, ejes = plt.subplots(1, 3, figsize=(10, 3))

para n_vecinos, ax en zip([1, 3, 9], ejes):
    # el método de ajuste devuelve el objeto mismo, por lo que podemos
    instanciarlo # y ajustarlo
    en una línea clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax) ax.set_title("{}
    vecino(s)".format(n_neighbors)) ax.set_xlabel("característica
    0") ax.set_ylabel("característica
    1") axes[0].legend(loc=3)
```

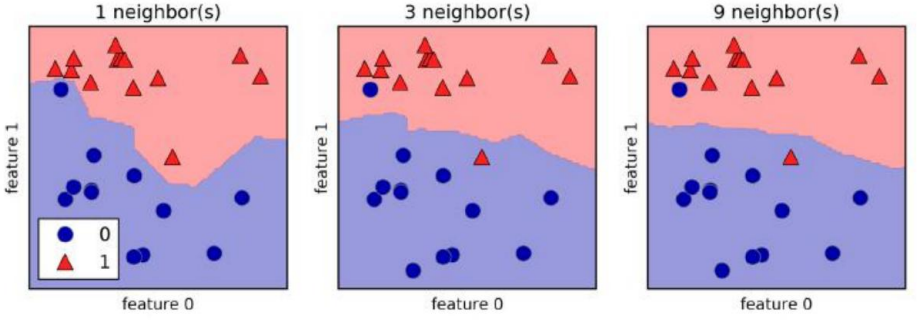


Figura 2-6. Límites de decisión creados por el modelo de vecinos más cercanos para diferentes valores de n_vecinos

Como puede ver a la izquierda de la figura, usar un solo vecino da como resultado un límite de decisión que sigue de cerca los datos de entrenamiento. Considerar más y más vecinos conduce a un límite de decisión más suave. Un límite más suave corresponde a un modelo más simple. En otras palabras, usar pocos vecinos corresponde a una alta complejidad del modelo (como se muestra en el lado derecho de la Figura 2-1), y usar muchos vecinos corresponde a una baja complejidad del modelo (como se muestra en el lado izquierdo de la Figura 2-1). Si considera el caso extremo donde el número de vecinos es el número de todos los puntos de datos en el conjunto de entrenamiento, cada punto de prueba tendría exactamente los mismos vecinos (todos los puntos de entrenamiento) y todas las predicciones serían las mismas: la clase que es más frecuente en el conjunto de entrenamiento.

Investiguemos si podemos confirmar la conexión entre la complejidad del modelo y la generalización que comentamos anteriormente. Lo haremos con el conjunto de datos de cáncer de mama del mundo real. Comenzamos dividiendo el conjunto de datos en un conjunto de entrenamiento y uno de prueba. Luego...

Evaluamos el rendimiento del conjunto de entrenamiento y prueba con diferentes cantidades de vecinos. Los resultados se muestran en la Figura 2-7:

En[18]:

```
de sklearn.datasets importar load_breast_cancer

cáncer = cargar_cáncer_de_mama()
X_train, X_test, y_train, y_test = train_test_split( cáncer.datos, cáncer.objetivo,
        estratificar=cáncer.objetivo, estado_aleatorio=66)

precisión_de_entrenamiento = []
precisión_de_prueba = []
# prueba n_vecinos de 1 a 10
configuración_de_vecinos = rango(1, 11)

para n_vecinos en configuración de vecinos:
        # construir el modelo clf
        = KNeighborsClassifier(n_neighbors=n_neighbors) clf.fit(X_train, y_train) #
        registrar la precisión del conjunto de
        entrenamiento
        training_accuracy.append(clf.score(X_train, y_train)) # registrar la precisión de
        generalización test_accuracy.append(clf.score(X_test,
        y_test))

plt.plot(configuración_de_vecinos, precisión_de_entrenamiento, etiqueta="precisión_de_entrenamiento")
plt.plot(configuración_de_vecinos, precisión_de_prueba, etiqueta=" precisión_de_prueba")
plt.ylabel("Precisión")
plt.xlabel("n_vecinos") plt.legend()
```

El gráfico muestra la precisión del conjunto de entrenamiento y prueba en el eje y contra la configuración de n_vecinos en el eje x. Si bien los gráficos del mundo real rara vez son muy suaves, aún podemos reconocer algunas de las características del sobreajuste y el subajuste (tenga en cuenta que debido a que considerar menos vecinos corresponde a un modelo más complejo, el gráfico está invertido horizontalmente en relación con la ilustración de la Figura 2-1). Considerando un solo vecino más cercano, la predicción en el conjunto de entrenamiento es perfecta. Pero cuando se consideran más vecinos, el modelo se vuelve más simple y la precisión del entrenamiento disminuye. La precisión del conjunto de prueba para usar un solo vecino es menor que cuando se usan más vecinos, lo que indica que usar el solo vecino más cercano conduce a un modelo demasiado complejo. Por otro lado, al considerar 10 vecinos, el modelo es demasiado simple y el rendimiento es aún peor. El mejor rendimiento está en algún lugar en el medio, usando alrededor de seis vecinos. Aún así, es bueno mantener la escala del gráfico en mente. El peor rendimiento ronda el 88% de precisión, que todavía podría ser aceptable.

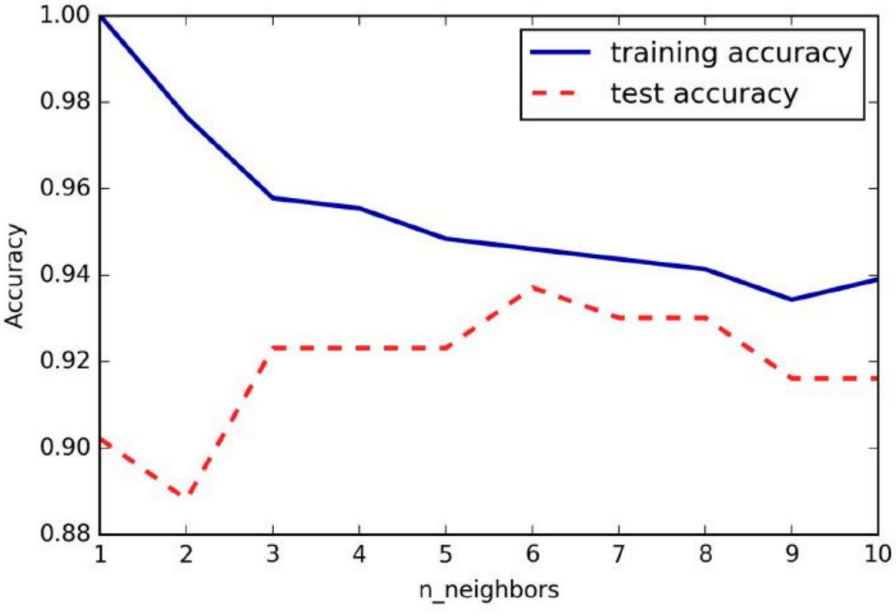


Figura 2-7. Comparación de la precisión del entrenamiento y la prueba en función de n_vecinos

Regresión de k vecinos.

También existe una variante de regresión del algoritmo de k vecinos más cercanos. De nuevo, comencemos usando el vecino más cercano, esta vez con el conjunto de datos de ondas . Hemos añadido tres puntos de prueba como estrellas verdes en el eje x. La predicción con un solo vecino es simplemente el valor objetivo del vecino más cercano. Estos se muestran como estrellas azules en la Figura 2-8.

En[19]:

```
mglearn.plots.plot_knn_regression(n_vecinos=1)
```

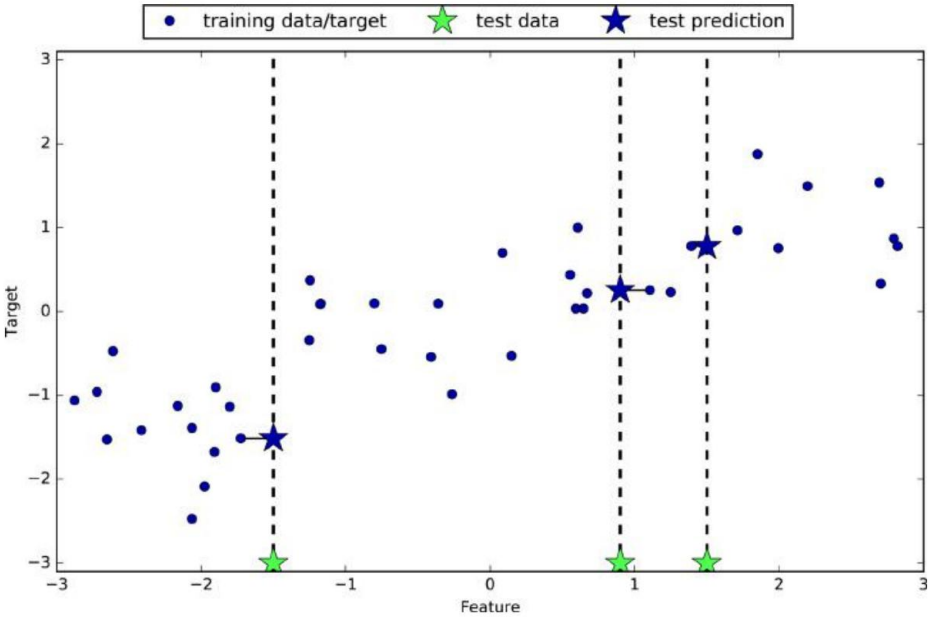


Figura 2-8. Predicciones realizadas mediante la regresión de un vecino más cercano en el conjunto de datos de ondas

Nuevamente, podemos usar más de un vecino más cercano para la regresión. Al usar varios vecinos más cercanos, la predicción es el promedio o la media de los vecinos relevantes (Figura 2-9):

En[20]:

```
mglearn.plots.plot_knn_regression(n_vecinos=3)
```

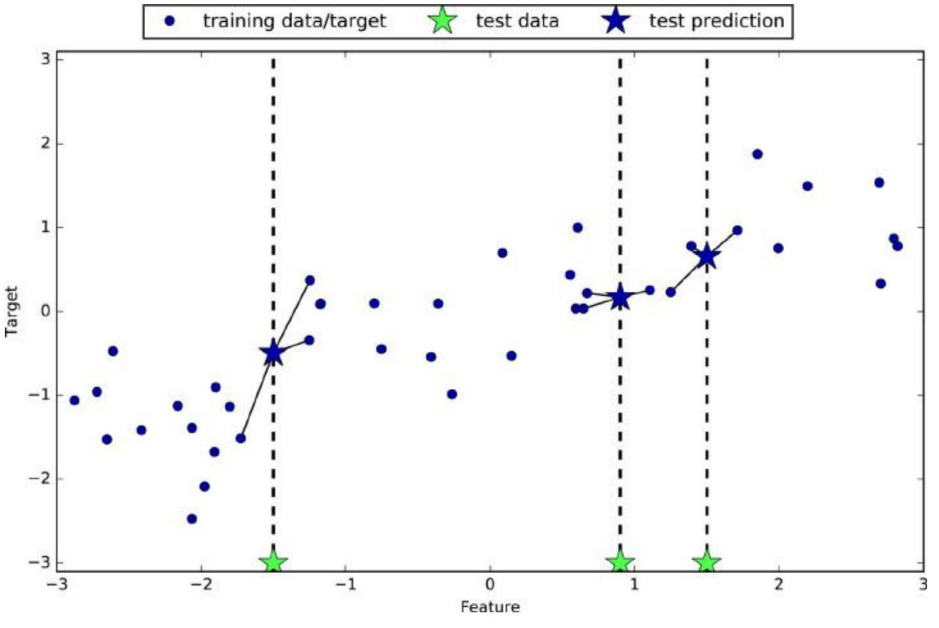


Figura 2-9. Predicciones realizadas mediante la regresión de los tres vecinos más próximos en el conjunto de datos de ondas.

El algoritmo de k vecinos más cercanos para regresión se implementa en la clase KNeighbors Regressor de scikit-learn. Su uso es similar al de KNeighborsClassifier:

En[21]:

```
de sklearn.neighbors importa KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_muestras=40)

# dividir el conjunto de datos de ondas en un conjunto de entrenamiento y uno de prueba
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = tren_prueba_división(X, y, estado_aleatorio=0)

# instancia el modelo y establece el número de vecinos a considerar en 3 reg =
KNeighborsRegressor(n_neighbors=3) # ajusta el modelo
usando los datos de entrenamiento y los objetivos de entrenamiento reg.fit(X_train,
y_train)
```

Ahora podemos hacer predicciones en el conjunto de prueba:

En[22]:

```
print(" Predicciones del conjunto de pruebas :\n{}".format(reg.predict(X_test)))
```

Salida[22]:

```
Predicciones del conjunto
de pruebas: [-0,054 0,357 1,137 -1,894 -1,139 -1,631 0,357 0,912 -0,447 -1,139]
```

También podemos evaluar el modelo usando el método de puntuación , que para los regresores devuelve la puntuación R, también conocida como coeficiente de determinación, es una medida de bondad de una predicción para un modelo de regresión y produce una puntuación entre 0 y 1. Un valor de 1 corresponde a una predicción perfecta y un valor de 0 corresponde a un modelo constante que solo predice la media de las respuestas del conjunto de entrenamiento, y_train:

### Puntuación $R^2$

En[23]:

```
print(" Conjunto de pruebas R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

Salida[23]:

```
Conjunto de prueba R^2: 0,83
```

Aquí, la puntuación es 0,83, lo que indica un ajuste del modelo relativamente bueno.

Análisis de KNeighborsRegressor. Para

nuestro conjunto de datos unidimensional, podemos ver cómo se ven las predicciones para todos los valores de características posibles (Figura 2-10). Para ello, creamos un conjunto de datos de prueba compuesto por varios puntos en la línea:

En[24]:

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4)) # crea 1000
puntos de datos, espaciados uniformemente entre -3 y 3 line =
np.linspace(-3, 3, 1000).reshape(-1, 1) for n_neighbors, ax
in zip([1, 3, 9], axes):
    # hacer predicciones usando 1, 3 o 9 vecinos reg =
    KNeighborsRegressor(n_neighbors=n_neighbors) reg.fit(X_train,
    y_train) ax.plot(line,
    reg.predict(line)) ax.plot(X_train, y_train,
    '^', c=mglearn.cm2(0), markersize=8) ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1),
    markersize=8)

    ax.set_title(
        "{} vecino(s)\n puntuación del tren: {:.2f} puntuación de la prueba: {:.2f}".format(
            n_vecinos, reg.score(X_train, y_train),
            reg.score(X_test, y_test)))
    ax.set_xlabel("Característica")
    ax.set_ylabel("Objetivo")
axes[0].legend([" Predicciones del modelo", "Datos de entrenamiento/
                objetivo", "Datos de prueba/objetivo"], loc="mejor")
```

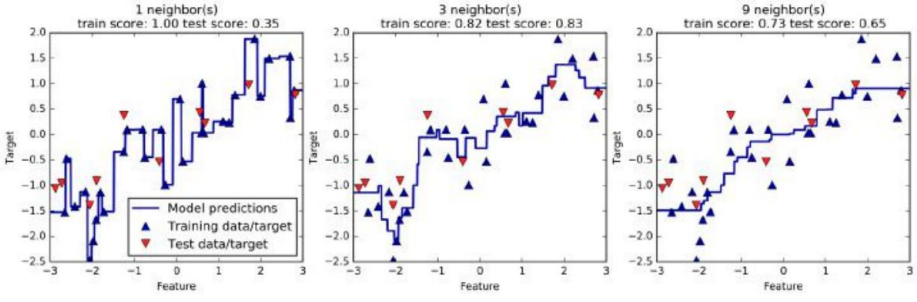


Figura 2-10. Comparación de las predicciones realizadas mediante la regresión de vecinos más cercanos para diferentes valores de n_vecinos

Como se puede observar en el gráfico, al usar un solo vecino, cada punto del conjunto de entrenamiento tiene una influencia evidente en las predicciones, y los valores predichos abarcan todos los puntos de datos. Esto genera una predicción muy inestable. Considerar más vecinos resulta en predicciones más uniformes, pero estas no se ajustan tan bien a los datos de entrenamiento.

Fortalezas, debilidades y parámetros. En

principio, el clasificador KNeighbors tiene dos parámetros importantes : el número de vecinos y la forma de medir la distancia entre los puntos de datos. En la práctica, usar un número pequeño de vecinos, como tres o cinco, suele ser adecuado, pero conviene ajustar este parámetro. Elegir la medida de distancia correcta queda fuera del alcance de este libro. Por defecto, se utiliza la distancia euclidiana, que funciona bien en muchos entornos.

Una de las ventajas de k-NN es que el modelo es muy fácil de entender y suele ofrecer un rendimiento razonable sin muchos ajustes. Este algoritmo es un buen método de referencia para probar antes de considerar técnicas más avanzadas. Construir el modelo de vecinos más cercanos suele ser muy rápido, pero cuando el conjunto de entrenamiento es muy grande (ya sea en número de características o de muestras), la predicción puede ser lenta.

Al utilizar el algoritmo k-NN, es importante preprocesar los datos (véase el capítulo 3). Este enfoque no suele funcionar bien en conjuntos de datos con muchas características (cientos o más), y es especialmente deficiente con conjuntos de datos donde la mayoría de las características son 0 la mayor parte del tiempo (los denominados conjuntos de datos dispersos).

Por lo tanto, si bien el algoritmo de k vecinos más cercanos es fácil de entender, no se suele utilizar en la práctica debido a que la predicción es lenta y a su incapacidad para manejar muchas características.
El método que analizaremos a continuación no tiene ninguno de estos inconvenientes.

## Modelos lineales

Los modelos lineales son una clase de modelos ampliamente utilizados en la práctica y que se han estudiado extensamente en las últimas décadas, con una historia que se remonta a más de cien años. Los modelos lineales realizan una predicción utilizando una función lineal de las características de entrada, que explicaremos en breve.

### Modelos lineales para regresión

Para la regresión, la fórmula de predicción general para un modelo lineal es la siguiente:

ŷ = w[0] * x[0] + w[1] * x[1] + ⋯ + w[p] * x[p] + b

Aquí, x[0] a x[p] denotan las características (en este ejemplo, el número de características es p) de un único punto de datos, w y b son los parámetros del modelo aprendidos, y ŷ es la predicción que realiza el modelo. Para un conjunto de datos con una única característica, esto es:

ŷ = w[0] * x[0] + b

que quizás recuerdes de las matemáticas de la secundaria como la ecuación de una línea.
Aquí, w[0] es la pendiente y b es el desplazamiento del eje y. Para más características, w contiene las pendientes a lo largo de cada eje de características. Alternativamente, se puede considerar la respuesta predicha como una suma ponderada de las características de entrada, con ponderaciones (que pueden ser negativas) dadas por las entradas de w.

Intentar aprender los parámetros w[0] y b en nuestro conjunto de datos de ondas unidimensionales podría conducir a la siguiente línea (ver Figura 2-11):

En[25]:

```
mglearn.plots.plot_regresión_lineal_onda()
```

Salida[25]:

```
w[0]: 0,393906 b: -0,031804
```

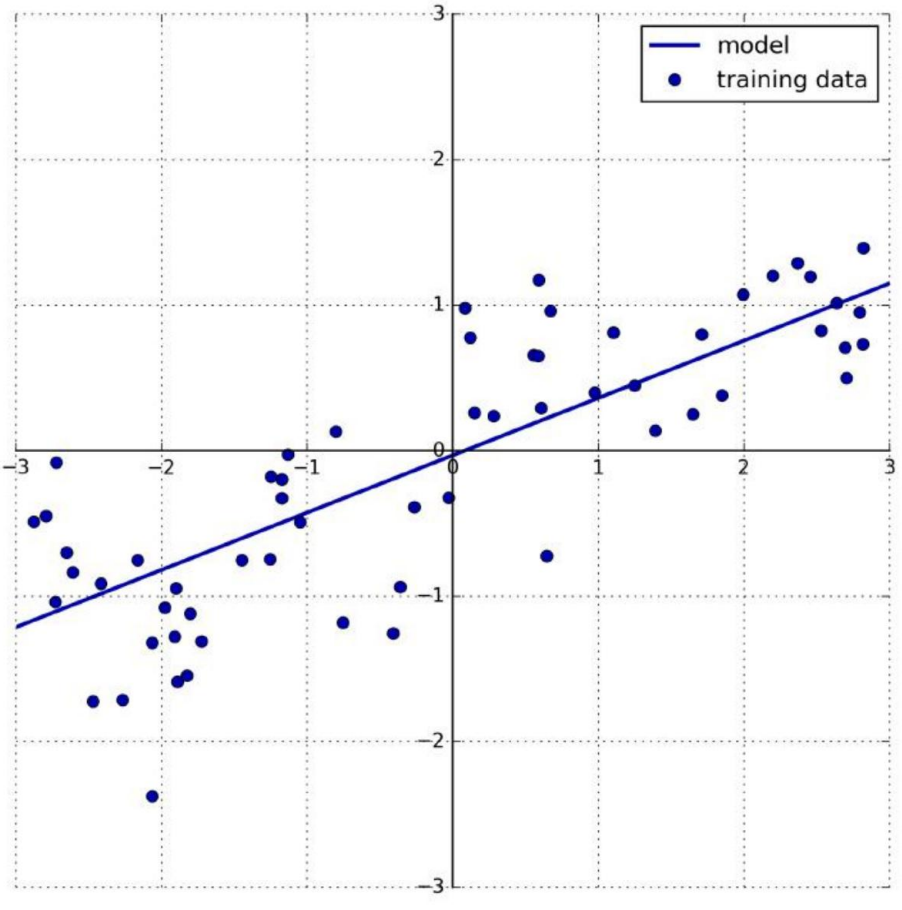


Figura 2-11. Predicciones de un modelo lineal en el conjunto de datos de olas

Agregamos una cruz de coordenadas al gráfico para que sea más fácil entender la línea.
Al observar w[0], vemos que la pendiente debería rondar 0,4, lo cual se puede confirmar visualmente en el gráfico. La intersección es donde la línea de predicción debería cruzar el eje y: esto está ligeramente por debajo de cero, lo cual también se puede confirmar en la imagen.

Los modelos lineales de regresión se pueden caracterizar como modelos de regresión para los cuales la predicción es una línea para una sola característica, un plano cuando se utilizan dos características o un hiperplano en dimensiones superiores (es decir, cuando se utilizan más características).

Si se comparan las predicciones realizadas con la línea recta con las del KNeighborsRegressor en la Figura 2-10, usar una línea recta para realizar predicciones parece muy restrictivo. Parece que se pierden todos los detalles finos de los datos. En cierto sentido, esto es cierto. Es una suposición sólida (y algo irreal) que nuestro objetivo y sea lineal.

Combinación de las características. Sin embargo, analizar datos unidimensionales ofrece una perspectiva algo sesgada. Para conjuntos de datos con muchas características, los modelos lineales pueden ser muy eficaces. En particular, si se tienen más características que puntos de datos de entrenamiento, cualquier valor objetivo y puede modelarse perfectamente (en el conjunto de entrenamiento) como una función lineal.

Existen muchos modelos lineales de regresión. La diferencia entre estos modelos radica en cómo se aprenden los parámetros w y b a partir de los datos de entrenamiento y cómo se puede controlar su complejidad. A continuación, analizaremos los modelos lineales de regresión más populares.

#### Regresión lineal (también conocida como mínimos cuadrados ordinarios)

La regresión lineal, o mínimos cuadrados ordinarios (MCO), es el método lineal más simple y clásico para la regresión. La regresión lineal encuentra los parámetros w y b que minimizan el error cuadrático medio entre las predicciones y los objetivos de regresión reales, y, en el conjunto de entrenamiento. El error cuadrático medio es la suma de las diferencias al cuadrado entre las predicciones y los valores reales. La regresión lineal no tiene parámetros, lo cual es una ventaja, pero tampoco permite controlar la complejidad del modelo.

Aquí está el código que produce el modelo que puedes ver en la Figura 2-11:

En[26]:

```
de sklearn.linear_model importar LinearRegression X, y =
mglearn.datasets.make_wave(n_muestras=60)
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = tren_prueba_división(X, y, estado_aleatorio=42)

lr = Regresión lineal().fit(X_train, y_train)
```

Los parámetros de “pendiente” (w), también llamados pesos o coeficientes, se almacenan en el atributo coef_ , mientras que el desplazamiento o intersección (b) se almacena en el atributo intercept_ :

En[27]:

```
imprimir("lr.coef_: {}".format(lr.coef_))
imprimir("lr.intercept_: {}".format(lr.intercept_))
```

Salida[27]:

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759746
```

Esto es fácil de ver si conoces algo de álgebra lineal.

Quizás notes el extraño guión bajo final al final de coef_ e intercept_. scikit-learn siempre almacena cualquier información derivada de los datos de entrenamiento en atributos que terminan con un guión bajo final. Esto sirve para separarlos de los parámetros definidos por el usuario.

El atributo intercept_ siempre es un número de coma flotante, mientras que el atributo coef_ es una matriz NumPy con una entrada por cada característica de entrada. Dado que solo tenemos una característica de entrada en el conjunto de datos de ondas , lr.coef_ solo tiene una entrada.

Veamos el rendimiento del conjunto de entrenamiento y del conjunto de prueba:

En[28]:

```
print("Puntuación del conjunto de entrenamiento: {:.2f}".format(lr.score(X_train, y_train)))
print(" Puntuación del conjunto de prueba: {:.2f}".format(lr.score(X_test, y_test)))
```

Salida[28]:

```
Puntuación del conjunto de entrenamiento: 0,67
Puntuación del conjunto de prueba: 0,66
```

Un valor de 2 de alrededor de 0,66 no es muy bueno, pero podemos ver que las puntuaciones en los conjuntos de entrenamiento An R y de prueba son muy similares. Esto significa que probablemente estemos subajusteando, no sobreajusteando. Para este conjunto de datos unidimensional, hay poco peligro de sobreajuste, ya que el modelo es muy simple (o restringido). Sin embargo, con conjuntos de datos de mayor dimensión (es decir, conjuntos de datos con un gran número de características), los modelos lineales se vuelven más potentes y hay una mayor probabilidad de sobreajuste. Veamos cómo se comporta la regresión LinearRe en un conjunto de datos más complejo, como el conjunto de datos de Boston Housing.
Recuerde que este conjunto de datos tiene 506 muestras y 105 características derivadas. Primero, cargamos el conjunto de datos y lo dividimos en un conjunto de entrenamiento y uno de prueba. Luego, construimos el modelo de regresión lineal como antes:

En[29]:

```
X, y = mglearn.datasets.load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0) lr =
LinearRegression().fit(X_train, y_train)
```

Al comparar las puntuaciones del conjunto de entrenamiento y del conjunto de prueba, descubrimos que predecimos con mucha precisión en el conjunto de entrenamiento, pero el $R^2$ En el conjunto de pruebas es mucho peor:

En[30]:

```
print("Puntuación del conjunto de entrenamiento: {:.2f}".format(lr.score(X_train, y_train)))
print(" Puntuación del conjunto de prueba: {:.2f}".format(lr.score(X_test, y_test)))
```

Salida[30]:

```
Puntuación del conjunto de entrenamiento: 0,95
Puntuación del conjunto de pruebas: 0,61
```

Esta discrepancia entre el rendimiento en el conjunto de entrenamiento y el de prueba es una clara señal de sobreajuste, por lo que debemos buscar un modelo que nos permita controlar la complejidad. Una de las alternativas más utilizadas a la regresión lineal estándar es la regresión de cresta, que analizaremos a continuación.

Regresión de

cresta La regresión de cresta también es un modelo lineal para la regresión, por lo que la fórmula que utiliza para hacer predicciones es la misma que se utiliza para los mínimos cuadrados ordinarios. Sin embargo, en la regresión de cresta, los coeficientes (w) se eligen no solo para que predigan bien en los datos de entrenamiento, sino también para ajustarse a una restricción adicional. También queremos que la magnitud de los coeficientes sea lo más pequeña posible; en otras palabras, todas las entradas de w deben ser cercanas a cero. Intuitivamente, esto significa que cada característica debe tener el menor efecto posible en el resultado (lo que se traduce en tener una pendiente pequeña), sin dejar de predecir bien. Esta restricción es un ejemplo de lo que se llama regularización. Regularización significa restringir explícitamente un modelo para evitar el sobreajuste. El tipo particular utilizado por la regresión de cresta se conoce como regularización L2.7

La regresión de crestas se implementa en linear_model.Ridge. Veamos su rendimiento en el conjunto de datos ampliado de Boston Housing:

En[31]:

```
de sklearn.linear_model importar Ridge

cresta = Ridge().fit(X_train, y_train) print("Puntaje del
conjunto de entrenamiento: {:.2f}".format(ridge.score(X_train, y_train))) print(" Puntaje del conjunto de prueba:
{:.2f}".format(ridge.score(X_test, y_test)))
```

Salida[31]:

```
Puntuación del conjunto de entrenamiento: 0,89
Puntuación del conjunto de prueba: 0,75
```

Como puede observar, la puntuación del conjunto de entrenamiento de Ridge es menor que la de la regresión lineal, mientras que la del conjunto de prueba es mayor. Esto coincide con nuestra expectativa. Con la regresión lineal, estábamos sobreajustando nuestros datos. Ridge es un modelo más restringido, por lo que es menos probable que sobreajustemos. Un modelo menos complejo implica un peor rendimiento en el conjunto de entrenamiento, pero una mejor generalización. Dado que solo nos interesa el rendimiento de la generalización, deberíamos elegir el modelo Ridge en lugar del modelo de regresión lineal .

---

Matemáticamente , Ridge penaliza la norma L2 de los coeficientes, o la longitud euclidiana de w.

# 1 Introducción

## 1.1 ¿Qué es el aprendizaje automático?

Para resolver un problema en una computadora, necesitamos un algoritmo. Un algoritmo es una secuencia de instrucciones que deben ejecutarse para transformar la entrada en salida. Por ejemplo, se puede diseñar un algoritmo para Ordenación. La entrada es un conjunto de números y la salida es su orden. lista. Para la misma tarea, puede haber varios algoritmos y podemos estar Interesado en encontrar el más eficiente, que requiera el menor número de instrucciones o memoria o ambas.

Sin embargo, para algunas tareas no tenemos un algoritmo; por ejemplo, Para distinguir los correos spam de los legítimos. Sabemos cuál es la entrada: Un documento de correo electrónico que, en el caso más simple, es un archivo de caracteres. saber cuál debería ser la salida: una salida de sí/no que indique si El mensaje es spam o no. No sabemos cómo transformar la entrada. A la salida. Lo que se puede considerar spam cambia con el tiempo y desde de individuo a individuo.

Lo que nos falta en conocimiento, lo compensamos con datos. Podemos fácilmente... Recopilar miles de mensajes de ejemplo, algunos de los cuales sabemos que son spam y lo que queremos es “aprender” qué constituye spam a partir de ellos. En otras palabras, nos gustaría que la computadora (máquina) extrajera automáticamente el algoritmo para esta tarea. No es necesario aprender a ordenar. números, ya tenemos algoritmos para eso; pero hay muchas aplicaciones para las que no tenemos un algoritmo pero sí tenemos ejemplos datos.

Con los avances en la tecnología informática, actualmente tenemos la capacidad de Almacenar y procesar grandes cantidades de datos, así como acceder a ellos desde ubicaciones físicamente distantes a través de una red informática. La mayoría de los procesos de adquisición de datos...

Los dispositivos ahora son digitales y registran datos fiables. Pensemos, por ejemplo, en una cadena de supermercados con cientos de tiendas en todo un país que vende miles de productos a millones de clientes. Los terminales de punto de venta registran los detalles de cada transacción: fecha, código de identificación del cliente, productos comprados y su importe, dinero total gastado, etc. Esto suele representar gigabytes de datos diarios. Lo que la cadena de supermercados busca es predecir quiénes son los posibles clientes de un producto. De nuevo, el algoritmo para ello no es evidente; cambia con el tiempo y la ubicación geográfica. Los datos almacenados solo resultan útiles cuando se analizan y se convierten en información que podemos utilizar, por ejemplo, para hacer predicciones.

No sabemos con exactitud qué personas probablemente comprarán este sabor de helado, el próximo libro de este autor, verán esta nueva película, visitarán esta ciudad o harán clic en este enlace. Si lo supiéramos, no necesitaríamos analizar los datos; simplemente escribiríamos el código. Pero como no lo sabemos, solo podemos recopilar datos y esperar extraer de ellos las respuestas a estas y otras preguntas similares.

Creemos que existe un proceso que explica los datos que observamos.
Aunque desconocemos los detalles del proceso subyacente a la generación de datos —por ejemplo, el comportamiento del consumidor—, sabemos que no es completamente aleatorio. La gente no va al supermercado a comprar cosas al azar. Cuando compran cerveza, compran patatas fritas; compran helado en verano y especias para el Glühwein en invierno. Existen ciertos patrones en los datos.

Quizás no podamos identificar el proceso por completo, pero creemos que podemos construir una aproximación buena y útil. Esta aproximación quizá no lo explique todo, pero sí podría explicar parte de los datos. Creemos que, aunque no sea posible identificar el proceso completo, sí podemos detectar ciertos patrones o regularidades. Este es el nicho del aprendizaje automático. Dichos patrones pueden ayudarnos a comprender el proceso o podemos usarlos para hacer predicciones: suponiendo que el futuro, al menos el futuro cercano, no será muy diferente del pasado cuando se recopilaron los datos de muestra, también cabe esperar que las predicciones futuras sean correctas.

La aplicación de métodos de aprendizaje automático a grandes bases de datos se denomina minería de datos. La analogía es que se extrae un gran volumen de tierra y materia prima de una mina, que al procesarse produce una pequeña cantidad de material muy valioso; de igual manera, en la minería de datos, se procesa un gran volumen de datos para construir un modelo simple con un uso valioso.

Por ejemplo, tener una alta precisión predictiva. Sus áreas de aplicación son
Abundante: Además del comercio minorista, en finanzas los bancos analizan sus datos pasados
para construir modelos para usar en solicitudes de crédito, detección de fraudes y
Mercado de valores. En la industria manufacturera, los modelos de aprendizaje se utilizan para la
optimización, el control y la resolución de problemas. En medicina, los programas de aprendizaje son...
Se utiliza para diagnóstico médico. En telecomunicaciones, los patrones de llamadas se analizan
para optimizar la red y maximizar la calidad del servicio.
En ciencia, se pueden procesar grandes cantidades de datos en física, astronomía y biología.
solo puede ser analizado con la suficiente rapidez por computadoras. La World Wide Web es enorme;
Está en constante crecimiento y la búsqueda de información relevante no puede ser
hecho manualmente

Pero el aprendizaje automático no es sólo un problema de base de datos; también es una parte
de inteligencia artificial. Para ser inteligente, un sistema que está en constante cambio
El entorno debe tener la capacidad de aprender. Si el sistema puede aprender y
Para adaptarse a tales cambios, el diseñador del sistema no necesita prever ni proporcionar
Soluciones para todas las situaciones posibles.

El aprendizaje automático también nos ayuda a encontrar soluciones a muchos problemas en
visión, reconocimiento de voz y robótica. Tomemos el ejemplo del reconocimiento de rostros: es una
tarea que realizamos sin esfuerzo; todos los días reconocemos...
familiares y amigos mirando sus caras o sus fotografías, a pesar de las diferencias en pose,
iluminación, peinado, etc.
Pero lo hacemos inconscientemente y somos incapaces de explicar cómo lo hacemos. Como no
podemos explicar nuestra experiencia, no podemos escribir el programa de computadora. Al mismo
tiempo, sabemos que la imagen de un rostro no es solo...
Colección aleatoria de píxeles; un rostro tiene estructura. Es simétrico.
son los ojos, la nariz, la boca, situados en determinados lugares de la cara.
El rostro de cada persona es un patrón compuesto por una combinación particular
de estos. Al analizar imágenes de muestra del rostro de una persona, un programa de aprendizaje
captura el patrón específico de esa persona y luego lo reconoce por
Comprobando este patrón en una imagen dada. Este es un ejemplo de patrón.
reconocimiento.

El aprendizaje automático consiste en programar computadoras para optimizar su rendimiento.
Criterio basado en datos de ejemplo o experiencia pasada. Tenemos un modelo definido.
hasta algunos parámetros, y el aprendizaje es la ejecución de un programa de computadora para
optimizar los parámetros del modelo utilizando los datos de entrenamiento o
Experiencia pasada. El modelo puede ser predictivo para hacer predicciones en el
futuro, o descriptivo para obtener conocimiento a partir de los datos, o ambos.

El aprendizaje automático utiliza la teoría de la estadística para construir modelos matemáticos.
modelos, porque la tarea principal es hacer inferencias a partir de una muestra.

El papel de la informática es doble: primero, en la formación, necesitamos profesionales eficientes.
Algoritmos para resolver el problema de optimización, así como para almacenar y procesar la enorme
cantidad de datos que generalmente tenemos. En segundo lugar, una vez que un modelo...
Se aprende, su representación y solución algorítmica para las necesidades de inferencia.
Para ser eficiente también. En ciertas aplicaciones, la eficiencia del algoritmo de aprendizaje o
inferencia, es decir, su complejidad espacial y temporal, puede
ser tan importante como su precisión predictiva.

Analicemos ahora algunas aplicaciones de ejemplo con más detalle para obtener más información.
Más información sobre los tipos y usos del aprendizaje automático.

## 1.2 Ejemplos de aplicaciones de aprendizaje automático

### 1.2.1 Asociaciones de aprendizaje

En el caso del comercio minorista (por ejemplo, una cadena de supermercados), una aplicación
Una de las principales ventajas del aprendizaje automático es el análisis de cestas, que consiste en encontrar
asociaciones entre los productos comprados por los clientes: si las personas que compran X normalmente también...
compra Y, y si hay un cliente que compra X y no compra Y,
o ella es un cliente potencial de Y. Una vez que encontramos a dichos clientes, podemos
orientarlos para la venta cruzada.

regla de asociación

Para encontrar una regla de asociación, nos interesa aprender una regla condicional.
probabilidad de la forma P(Y|X) donde Y es el producto que nos gustaría obtener
condición sobre X, que es el producto o el conjunto de productos que
saber que el cliente ya ha comprado.

Digamos que, revisando nuestros datos, calculamos que P(patatas fritas|cerveza) = 0,7.
Luego podemos definir la regla:

El 70 por ciento de los clientes que compran cerveza también compran patatas fritas.

Quizás queramos hacer una distinción entre los clientes y en relación con esto,
estimar P(Y|X,D) donde D es el conjunto de atributos del cliente, por ejemplo, género, edad, estado
civil, etc., suponiendo que tenemos acceso
A esta información. Si se trata de una librería en lugar de un supermercado, los productos pueden
ser libros o autores. En el caso de un portal web, los elementos corresponden a enlaces a páginas
web, y podemos estimar los enlaces que probablemente reciba un usuario.
Para hacer clic y utilizar esta información para descargar dichas páginas con antelación
acceso más rápido.

### 1.2.2 Clasificación

Un crédito es una cantidad de dinero prestada por una institución financiera, por ejemplo un banco, para ser devuelta con intereses, generalmente en cuotas.
Es importante que el banco pueda predecir con antelación el riesgo asociado a un préstamo, es decir, la probabilidad de que el cliente incumpla y no pague la totalidad del importe. Esto sirve tanto para garantizar que el banco obtenga beneficios como para no perjudicar al cliente con un préstamo que exceda su capacidad financiera.

En la calificación crediticia (Hand 1998), el banco calcula el riesgo dado el monto del crédito y la información sobre el cliente. La información...
La información sobre el cliente incluye datos a los que tenemos acceso y que son relevantes para Calcular su capacidad financiera, es decir, ingresos, ahorros, garantías, profesión, edad, historial financiero, etc. El banco cuenta con un registro de préstamos anteriores que contiene datos del cliente e indica si el préstamo fue devuelto o no. A partir de estos datos de solicitudes específicas, el objetivo es inferir una regla general que codifica la asociación entre los atributos de un cliente y su riesgo. Es decir, el sistema de aprendizaje automático ajusta un modelo a los datos históricos para calcular el riesgo de una nueva solicitud y, en consecuencia, decide aceptarla o rechazarla.

clasificación

Este es un ejemplo de un problema de clasificación donde existen dos clases: clientes de bajo riesgo y clientes de alto riesgo. La información sobre un cliente constituye la entrada del clasificador, cuya tarea es asignar la entrada a una de las dos clases.

Después del entrenamiento con los datos pasados, una regla de clasificación aprendida puede tener la forma

SI ingresos > θ1 Y ahorros > θ2 ENTONCES riesgo bajo SI NO riesgo alto

Para valores adecuados de θ1 y θ2 (véase la figura 1.1). Este es un ejemplo de discriminante ; es una función que separa los ejemplos de diferentes clases.

predicción

Con una regla como esta, la principal aplicación es la predicción: una vez que tenemos una regla que se ajusta a los datos pasados, si el futuro es similar al pasado, podemos hacer predicciones correctas para nuevas instancias. Dada una nueva aplicación con ciertos ingresos y ahorros, podemos determinar fácilmente si es de bajo o alto riesgo.

En algunos casos, en lugar de tomar una decisión de tipo 0/1 (riesgo bajo/riesgo alto), es posible que queramos calcular una probabilidad, es decir, P(Y|X), donde
X son los atributos del cliente e Y es 0 o 1 respectivamente para riesgo bajo

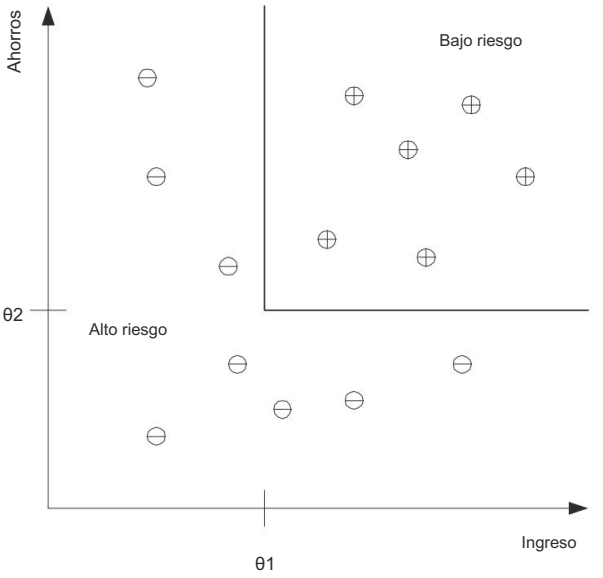


Figura 1.1 Ejemplo de un conjunto de datos de entrenamiento donde cada círculo corresponde a uno instancia de datos con valores de entrada en los ejes correspondientes y su signo indica la clase. Para simplificar, solo dos atributos del cliente, ingresos y ahorros, se toman como entrada y las dos clases son de bajo riesgo ('+') y de alto riesgo ('−'). Un También se muestra un ejemplo discriminante que separa los dos tipos de ejemplos.

y de alto riesgo. Desde esta perspectiva, podemos ver la clasificación como el aprendizaje de una asociación entre X e Y. Entonces, para un X = x dado, si tenemos $P(Y = 1|X = x) = 0,8$, decimos que el cliente tiene una probabilidad del 80 por ciento de ser de alto riesgo, o equivalentemente una probabilidad del 20 por ciento de ser de bajo riesgo. Luego decidimos si aceptamos o rechazamos el préstamo dependiendo sobre las posibles ganancias y pérdidas.

patrón Reconocimiento

Existen muchas aplicaciones del aprendizaje automático en el reconocimiento de patrones. Uno es el reconocimiento óptico de caracteres, que consiste en reconocer códigos de caracteres. de sus imágenes. Este es un ejemplo donde hay múltiples clases, Tantos como caracteres nos gustaría reconocer. Resulta especialmente interesante cuando los caracteres están escritos a mano, por ejemplo, Para leer códigos postales en sobres o cantidades en cheques. Las personas tienen diferentes estilos de escritura; los caracteres pueden escribirse pequeños o grandes, inclinados, con un bolígrafo o un lápiz, y hay muchas imágenes posibles correspondientes

Al mismo personaje. Aunque la escritura es una invención humana, no tenemos un sistema tan preciso como un lector humano. No tenemos una descripción formal de «A» que abarque todas las «A» y ninguna de las que no lo son. Al no tenerla, tomamos ejemplos de escritores y aprendemos una definición de «A-idad» a partir de estos ejemplos. Pero aunque desconocemos qué hace que una imagen sea una «A», estamos seguros de que todas esas «A» distintas tienen algo en común, que es lo que queremos extraer de los ejemplos. Sabemos que la imagen de un personaje no es solo una colección de puntos aleatorios; es una colección de trazos y tiene una regularidad que podemos capturar mediante un programa de aprendizaje.

Si leemos un texto, un factor que podemos aprovechar es la redundancia en los lenguajes humanos. Una palabra es una secuencia de caracteres, y los caracteres sucesivos no son independientes, sino que están limitados por las palabras del lenguaje. Esto tiene la ventaja de que, incluso si no podemos reconocer un carácter, podemos leer la palabra. Estas dependencias contextuales también pueden ocurrir en niveles superiores, entre palabras y oraciones, a través de la sintaxis y la semántica del lenguaje. Existen algoritmos de aprendizaje automático para aprender secuencias y modelar dichas dependencias.

En el caso del reconocimiento facial, la entrada es una imagen, las clases son las personas a reconocer y el programa de aprendizaje debe aprender a asociar las imágenes faciales con las identidades. Este problema es más complejo que el del reconocimiento óptico de caracteres porque hay más clases, la imagen de entrada es más grande, el rostro es tridimensional y las diferencias en la pose y la iluminación provocan cambios significativos en la imagen. También puede haber oclusión de ciertas entradas; por ejemplo, las gafas pueden ocultar los ojos y las cejas, y la barba puede ocultar la barbilla.

En el diagnóstico médico, los datos de entrada son la información relevante que tenemos sobre el paciente, y las clases son las enfermedades. Estos datos contienen la edad, el sexo, el historial médico y los síntomas actuales del paciente. Es posible que algunas pruebas no se hayan aplicado al paciente, por lo que estos datos podrían faltar. Las pruebas requieren tiempo, pueden ser costosas e incomodar al paciente, por lo que no conviene aplicarlas a menos que creamos que nos proporcionarán información valiosa. En el caso de un diagnóstico médico, una decisión errónea puede resultar en un tratamiento erróneo o nulo, y en caso de duda, es preferible que el clasificador rechace la decisión y la delegue a un experto.

En el reconocimiento de voz, la entrada es acústica y las clases son palabras que se pueden pronunciar. En este caso, la asociación que se aprende es de una señal acústica a una palabra de algún idioma. Diferentes personas, porque...

Las diferencias de edad, género o acento hacen que la misma palabra se pronuncie de forma distinta, lo que dificulta bastante esta tarea. Otra diferencia en el habla es que la entrada es temporal; las palabras se pronuncian en el tiempo como una secuencia de Los fonemas del habla y algunas palabras son más largas que otras.

La información acústica sólo ayuda hasta cierto punto, y al igual que en la óptica, En el reconocimiento de caracteres, la integración de un "modelo de lenguaje" es fundamental Reconocimiento de voz y la mejor manera de crear un modelo de lenguaje Se aprende de nuevo a partir de un gran corpus de datos de ejemplo. Las aplicaciones del aprendizaje automático al procesamiento del lenguaje natural están en constante evolución. aumentando. El filtrado de spam es aquel en el que los generadores de spam por un lado y Los filtros del otro lado siguen encontrando formas cada vez más ingeniosas de se superan mutuamente. Quizás la más impresionante sea la traducción automática. Tras décadas de investigación sobre reglas de traducción codificadas manualmente, ha... Se ha hecho evidente recientemente que la forma más prometedora es proporcionar una una gran cantidad de pares de ejemplos de textos traducidos y un programa que determine automáticamente las reglas para mapear una cadena de caracteres a otro.

La biometría es el reconocimiento o autenticación de personas mediante sus características fisiológicas y/o comportamentales que requiere una integración de Entradas de diferentes modalidades. Ejemplos de características fisiológicas son las imágenes del rostro, la huella dactilar, el iris y la palma de la mano; ejemplos de características conductuales son la dinámica de la firma, la voz, la marcha y la pulsación de teclas. A diferencia de los procedimientos de identificación habituales (fotografía, firma impresa o contraseña), cuando hay muchas entradas diferentes (no correlacionadas), Las falsificaciones serían más difíciles y el sistema sería más preciso, ojalá sin demasiadas molestias para los usuarios. El aprendizaje automático se utiliza tanto en los reconocedores separados para estas diferentes modalidades como en la combinación de sus decisiones para obtener un resultado global. decisión de aceptar o rechazar, teniendo en cuenta la fiabilidad de estas diferentes Las fuentes son.

conocimiento extracción

Aprender una regla a partir de los datos también permite la extracción de conocimiento. La regla es un modelo simple que explica los datos, y mirando este modelo tenemos Una explicación sobre el proceso subyacente a los datos. Por ejemplo, una vez Aprendemos el discriminante que separa a los clientes de bajo riesgo y de alto riesgo, Conocemos las propiedades de los clientes de bajo riesgo. Podemos Luego use esta información para dirigirse más a clientes potenciales de bajo riesgo. eficientemente, por ejemplo, a través de la publicidad.

compresión

El aprendizaje también realiza compresión , ya que al ajustar una regla a los datos, Obtenemos una explicación más simple que los datos y que requiere menos memoria.

Oría para almacenar y menos cálculos para procesar. Una vez que se dominan las reglas de la suma, no es necesario recordar la suma de cada par de números posible.

detección de valores atípicos

Otro uso del aprendizaje automático es la detección de valores atípicos, que consiste en encontrar las instancias que no cumplen la regla y constituyen excepciones. En este caso, tras aprender la regla, no nos interesa la regla en sí, sino las excepciones que no cubre, lo que puede implicar anomalías que requieren atención, por ejemplo, fraude.

### 1.2.3 Regresión

Supongamos que queremos un sistema que prediga el precio de un coche usado. Las entradas son los atributos del coche (marca, año, cilindrada, kilometraje y otra información) que creemos que afectan a su valor. La salida es el precio del coche. Los problemas donde la salida es un número son...

problemas de regresión regresión .

Sea X el atributo del vehículo e Y su precio. Al analizar las transacciones pasadas, podemos recopilar datos de entrenamiento y el programa de aprendizaje automático ajusta una función a estos datos para aprender Y como función de X. La figura 1.2 muestra un ejemplo, donde la función ajustada tiene la forma

$$y = wx + w0$$

para valores adecuados de w y w0.

aprendizaje supervisado

Tanto la regresión como la clasificación son problemas de aprendizaje supervisado donde hay una entrada, X, una salida, Y, y la tarea consiste en aprender la correspondencia entre la entrada y la salida. El enfoque del aprendizaje automático consiste en asumir un modelo definido hasta un conjunto de parámetros:

$$y = g(x|\theta)$$

Donde $g(\cdot)$ es el modelo y $\theta$ son sus parámetros. Y es un número en regresión y un código de clase (p. ej., 0/1) en el caso de la clasificación. $g(\cdot)$ es la función de regresión o, en clasificación, la función discriminante que separa las instancias de diferentes clases. El programa de aprendizaje automático optimiza los parámetros, $\theta$, de modo que se minimice el error de aproximación; es decir, nuestras estimaciones se acerquen lo más posible a los valores correctos dados en el conjunto de entrenamiento. Por ejemplo, en la figura 1.2, el modelo es lineal y w y w0 son los parámetros optimizados para un mejor ajuste.

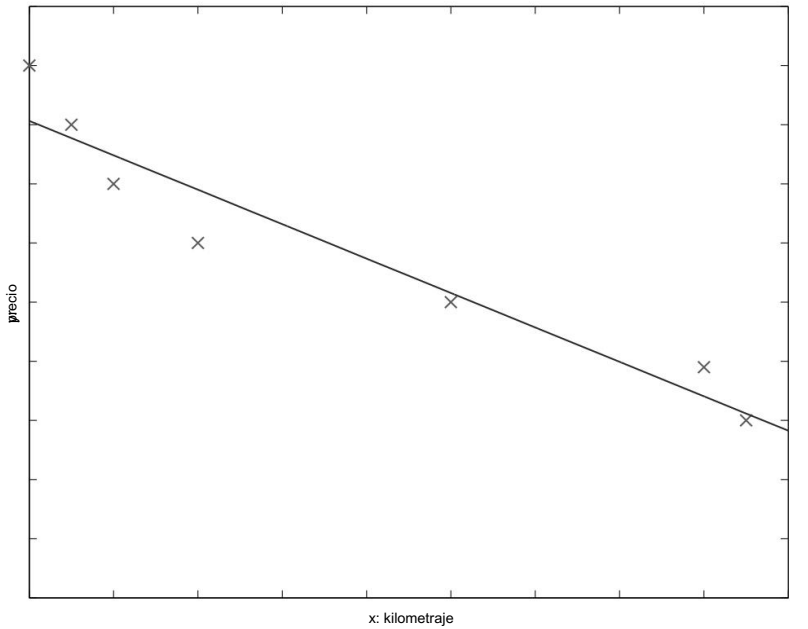


Figura 1.2. Conjunto de datos de entrenamiento de coches usados y la función ajustada. Para simplificar, se toma el kilometraje como único atributo de entrada y se utiliza un modelo lineal.

datos de entrenamiento. En los casos en que el modelo lineal sea demasiado restrictivo, se puede utilice por ejemplo una ecuación cuadrática

$$y = w_2x^2 + w_1x + w_0$$

o un polinomio de orden superior, o cualquier otra función no lineal de la entrada, esta vez optimizando sus parámetros para un mejor ajuste.

Otro ejemplo de regresión es la navegación de un robot móvil, por ejemplo, un automóvil autónomo, donde la salida es el ángulo en el que se mueve el robot.
El volante debe girarse en cada momento, para avanzar sin golpear
Obstáculos y desviaciones de la ruta. En tal caso, las entradas las proporcionan los sensores del vehículo, por ejemplo, una cámara de vídeo, GPS, etc.
adelante. Los datos de entrenamiento se pueden recopilar mediante el seguimiento y registro de la acciones de un conductor humano.

Se pueden imaginar otras aplicaciones de la regresión donde se intenta

Para optimizar una función 1. Supongamos que queremos construir una máquina para tostar café. La máquina tiene muchas entradas que afectan la calidad: diversos ajustes de temperatura, tiempos, tipo de grano, etc. Realizamos varios experimentos y, para diferentes ajustes de estas entradas, medimos la calidad del café, por ejemplo, como la satisfacción del consumidor.
Para encontrar la configuración óptima, ajustamos un modelo de regresión que vincula estas entradas con la calidad del café y seleccionamos nuevos puntos de muestreo cerca del óptimo del modelo actual para buscar una mejor configuración. Muestreamos estos puntos, verificamos la calidad, los agregamos a los datos y ajustamos un nuevo modelo. Esto se conoce generalmente como diseño de superficie de respuesta.

### 1.2.4 Aprendizaje no supervisado

En el aprendizaje supervisado, el objetivo es aprender una correspondencia entre la entrada y la salida, cuyos valores correctos son proporcionados por un supervisor. En el aprendizaje no supervisado, no existe tal supervisor y solo se dispone de datos de entrada.
El objetivo es encontrar las regularidades en la entrada. El espacio de entrada tiene una estructura tal que ciertos patrones ocurren con mayor frecuencia que otros, y queremos ver qué ocurre generalmente y qué no. En estadística, la estimación de densidad se denomina estimación de densidad.

agrupamiento

Un método para la estimación de densidad es la agrupación en clústeres , cuyo objetivo es encontrar clústeres o agrupaciones de entrada. En el caso de una empresa con datos de clientes anteriores, estos datos contienen la información demográfica, así como las transacciones pasadas con la empresa, y la empresa podría querer ver la distribución del perfil de sus clientes, para ver qué tipo de clientes ocurren con frecuencia. En tal caso, un modelo de agrupación en clústeres asigna clientes similares en sus atributos al mismo grupo, proporcionando a la empresa agrupaciones naturales de sus clientes; esto se denomina segmentación de clientes. Una vez que se encuentran dichos grupos, la empresa puede decidir estrategias, por ejemplo, servicios y productos, específicos para diferentes grupos; esto se conoce como gestión de relaciones con los clientes. Dicha agrupación también permite identificar a aquellos que son atípicos, es decir, aquellos que son diferentes de otros clientes, lo que puede implicar un nicho en el mercado que la empresa puede explotar aún más.

Una aplicación interesante del agrupamiento es la compresión de imágenes. En este caso, las instancias de entrada son píxeles de imagen representados como valores RGB. Un programa de agrupamiento agrupa píxeles con colores similares en el mismo...

---

1. Me gustaría agradecer a Michael Jordan por este ejemplo.

grupo, y dichos grupos corresponden a los colores que ocurren con frecuencia en la imagen. Si en una imagen, solo hay tonos de un pequeño número de colores, y si codificamos aquellos que pertenecen al mismo grupo con un color, por ejemplo, su promedio, entonces la imagen está cuantificada. Digamos que los píxeles son de 24 bits para representar 16 millones de colores, pero si hay tonos de solo 64 colores principales, para cada píxel necesitamos 6 bits en lugar de 24. Por ejemplo, si la escena tiene varios tonos de azul en diferentes partes de la imagen, y si usamos el mismo azul promedio para todos ellos, perdemos los detalles en la imagen pero ganamos espacio en el almacenamiento y la transmisión. Idealmente, uno querría identificar regularidades de alto nivel mediante el análisis de patrones de imagen repetidos, por ejemplo, textura, objetos, etc. Esto permite una descripción de alto nivel, más simple y más útil de la escena, y por ejemplo, logra una mejor compresión que la compresión a nivel de píxel.

Si hemos escaneado páginas de un documento, no tenemos píxeles aleatorios, sino imágenes de mapa de bits de caracteres. Los datos tienen estructura, y aprovechamos esta redundancia buscando una descripción más breve: el mapa de bits de 16 × 16 de «A» ocupa 32 bytes; su código ASCII solo ocupa 1 byte.

En la agrupación de documentos, el objetivo es agrupar documentos similares. Por ejemplo, las noticias se pueden subdividir en aquellas relacionadas con política, deportes, moda, arte, etc. Comúnmente, un documento se representa como un conjunto de palabras; es decir, se predefine un léxico de N palabras y cada documento es un vector binario N-dimensional cuyo elemento i es 1 si la palabra i aparece en el documento; se eliminan los sufijos "–s" y "–ing" para evitar duplicados, y no se utilizan palabras como "of", "and", etc., que no son informativas. Los documentos se agrupan entonces según el número de palabras compartidas. Por supuesto, aquí es crucial cómo se define el léxico.

es elegido

Los métodos de aprendizaje automático también se utilizan en bioinformática. El ADN en nuestro genoma es el "modelo de la vida" y consiste en una secuencia de bases, a saber, A, G, C y T. El ARN se transcribe a partir del ADN, y las proteínas se traducen a partir del ARN. Las proteínas son lo que el cuerpo vivo es y hace. Así como el ADN es una secuencia de bases, una proteína es una secuencia de aminoácidos (tal como se define por las bases). Un área de aplicación de la informática en biología molecular es la alineación, que consiste en hacer coincidir una secuencia con otra. Este es un problema complejo de coincidencia de cadenas porque las cadenas pueden ser bastante largas, hay muchas cadenas de plantilla con las que hacer coincidir, y puede haber deleciones, inserciones y sustituciones. La agrupación se utiliza para aprender motivos, que son secuencias de aminoácidos que se repiten en las proteínas. Los motivos son de interés porque pueden corresponder a características estructurales o funcionales.