Muller - Guido

Outline breve (págs. 84-97)

- **Árboles de decisión: idea y construcción**
    - Se introducen como jerarquías de preguntas condicionales (“20 Preguntas”) para clasificar cuatro animales usando tres atributos binarios (Fig. 2-22).
    - Ejemplo *two-moons*: el algoritmo elige cortes sucesivos (profundidades 1-2-9) y muestra los límites de decisión crecientemente detallados (Figs. 2-24 a 2-26).
- **Control de complejidad (prepoda)**
    - scikit-learn solo soporta poda previa; limitar `max_depth`, `max_leaf_nodes` o `min_samples_leaf` reduce sobreajuste.
    - Caso *Breast-Cancer*: árbol sin restricciones memoriza (100 % train, 0.94 test); con `max_depth=4` baja el error de test a ~0.95, equilibrio más sano.
- **Interpretación y análisis**
    - Exportación a Graphviz permite inspeccionar nodos y rutas (Fig. 2-27).
    - `feature_importances_`: bar plot revela que *worst radius* domina la predicción en el dataset de cáncer (Fig. 2-28).
- **Árboles para regresión**
    - Mismo mecanismo; particularidad: no extrapolan fuera del rango de entrenamiento. Ejemplo de precios de RAM: árbol clava el último valor conocido, mientras la regresión lineal proyecta tendencia log-lineal (Fig. 2-32).
- **Ensamblados de árboles (No entra)**
    - Motivación: los árboles individuales sobreajustan.
        - **Random Forest**: promedia muchos árboles entrenados en subconjuntos aleatorios de datos y atributos, reduciendo varianza.
- **Fortalezas y limitaciones clave**
        - Interpretables para árboles pequeños; invariantes al escalado; manejan mixto binario-continuo.
    - – Propensos a sobreajuste y alta varianza; incapaces de extrapolar; por ello, en práctica se prefiere usar ensembles para generalizar mejor.

Geron

Outline breve (págs. 315 – 331)

- **Introducción al Cap. 6 – Árboles de decisión**
    - Algoritmos versátiles para clasificación, regresión y salidas múltiples; base de los Random Forests.
- **Entrenamiento y visualización inicial**
    - Ejemplo con `DecisionTreeClassifier` entrenado sobre Iris (long./ancho de pétalo); exportación a Graphviz y árbol de 2 niveles (Fig. 6-1).
    - Fronteras de decisión 2-D con `max_depth = 2` (Fig. 6-2).
- **Mecánica de predicción y probabilidades**
    - Recorrido de nodos, campos `samples`, `value` y **gini**; nodo puro ⇒ gini = 0.
    - `predict_proba` estima proporciones de clase; ej. pétalo 5 × 1,5 cm → 90,7 % versicolor.
- **Impureza y criterios de división**
    - **Gini** (Ec. 6-1) por defecto; alternativa `criterion="entropy"`.
    - Gini ≈ entropía; Gini es más rápido y tiende a aislar la clase mayoritaria, entropía produce árboles algo más equilibrados.
- **Algoritmo CART**
    - División binaria (feature, threshold) que minimiza el coste J de impureza ponderada (Ec. 6-2).
    - Complejidad: entrenamiento O(n × m log m), predicción O(log m).
- **Regularización del árbol**
    - Hiperparámetros clave: `max_depth`, `min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf`, `max_leaf_nodes`, `max_features`.
    - Ejemplo *moons*: sin regularizar sobreajusta; con `min_samples_leaf = 5` mejora la frontera y la precisión test (0,898 → 0,92) (Fig. 6-3).
    - Árboles son modelos **no paramétricos**; mayor libertad implica riesgo de sobreajuste, mitigado con límites de tamaño.
    - **Poda posterior**: eliminación de nodos vía prueba χ^2 cuando la ganancia no es estadísticamente significativa.
- **Árboles de regresión**

    - `DecisionTreeRegressor` minimiza MSE (Ec. 6-4); ejemplo con datos cuadráticos y `max_depth = 2` (Fig. 6-4, 6-5).
    - Sobreajuste grave sin regularizar; `min_samples_leaf = 10` produce predicciones más suaves (Fig. 6-6).
- **Síntesis práctica**
    - Árboles son **modelos caja blanca**: interpretación directa de reglas, sin requisitos de escalado y predicciones O(log m).
    - Debilidad principal: sobreajuste, contra el que se actúa con hiperparámetros de parada/poda o con ensamblados (Random Forest, Boosting) introducidos en el capítulo siguiente (no entra).

Alppaydin

Outline breve (págs. 225 – 236)

- **Marco general y motivación**
    - Contraste entre estimación paramétrica (un solo modelo global) y métodos no paramétricos; los árboles dividen recursivamente el espacio y son eficientes para clasificación o regresión.
- **Estructura de un árbol de decisión**
    - Nodos internos con pruebas f_m(x), ramas etiquetadas por el resultado y nodos hoja que asignan la salida. Estrategia “divide y vencerás” que localiza la región correcta en \log_2 b pasos y permite convertir el árbol en reglas SI-ENTONCES interpretables.
- **Árboles univariados**
    - Cada prueba usa **un** único atributo:
        - Discreto → división n-vías por valor.
        - Numérico → comparación binaria x_j > w_{m0}.
    - Ejemplo bidimensional y representación gráfica (Fig. 9.1).
- **Construcción (ID3/C4.5, CART) — algoritmo goloso**
    - En cada nodo se evalúan todas las divisiones posibles y se elige la que **minimiza la impureza**; pseudocódigo C4.5 mostrado (Fig. 9.3).
- **Medidas de impureza en clasificación**
    - Entropía, Índice de Gini y error de clasificación; propiedades deseables (máximo con clases balanceadas, mínimo en nodos puros) y fórmula de impureza total tras la división.
- **Atributos de alta cardinalidad y sobreajuste**

    - Atributos con muchos valores pueden reducir artificialmente la impureza; se proponen penalizaciones y parámetros de complejidad (p. ej. tamaño mínimo de hoja, \theta_l).

- **Árboles de regresión**
    - Sustituyen la impureza por el **error cuadrático medio**; cada hoja almacena la media (o mediana) del objetivo y la división busca la mayor reducción de SSE. Posibilidad de modelos lineales en las hojas para suavizar.

- **Pre-poda vs. post-poda**
    - **Pre-poda:** detener el crecimiento si el nodo es casi puro o contiene pocas instancias.
    - **Post-poda:** crecer el árbol completo y luego reemplazar subárboles por hojas evaluando un conjunto de poda; suele producir árboles más precisos aunque requiere más cómputo (Figs. 9.4–9.5).

- **Conclusión operativa**
    - Los árboles ofrecen interpretabilidad y flexibilidad, pero necesitan controlar tamaño mediante parámetros de parada o poda para evitar sobreajuste; la elección de la métrica de impureza y del umbral de complejidad determina la eficacia final.

Marsland

Outline breve (págs. 270 – 284)

- **Motivación y ventajas de los árboles**
    - Estructura “20 Preguntas” para decidir acciones nocturnas; coste de consulta O(log N) y transparencia frente a modelos “caja negra”.

- **Algoritmo ID3**
    - Selección codiciosa basada en **entropía** y **ganancia de información**; sesgo inductivo hacia árboles pequeños (Occam/MDL).
    - Implementación en Python con diccionarios y recursión; maneja ruido y valores faltantes omitiendo ramas.

- **Sobreajuste y poda**
    - **Prepoda:** detener crecimiento por pureza mínima, nº de instancias o validación temprana.
    - **Pospoda (C4.5):** convertir el árbol a reglas y eliminar condiciones que no mejoran precisión.

- **Variables continuas**
    - Estrategias: discretizar o buscar el mejor punto de corte entre pares de ejemplos; solo se realiza una división por atributo continuo para contener el coste.

- **Complejidad computacional**
    - Construcción ≈ $O(d N^2 \log N)$ en el peor caso (árboles desbalanceados); inferencia $O(\log N)$.
- **Divisiones univariadas vs. multivariadas**
    - Árboles típicos eligen un atributo por vez; divisiones ortogonales se visualizan como cortes axis-aligned (Figs. 12.4-12.5).
- **CART y Gini**
    - Árboles binarios que usan **impureza de Gini** en lugar de entropía; extensión natural a regresión minimizando SSE en las hojas.
- **Ejemplo paso a paso**
    - Cálculo manual de entropías y ganancias para elegir la raíz y ramas (no entra); ilustra la mecánica de ID3 en un problema de ejemplo.

Fortalezas, debilidades y parámetros.

MultinomialNB y BernoulliNB tienen un único parámetro, alfa, que controla la complejidad del modelo. El algoritmo añade al alfa de los datos muchos puntos de datos virtuales con valores positivos para todas las características. Esto resulta en un suavizado de las estadísticas. Un alfa alto implica un mayor suavizado, lo que resulta en modelos menos complejos. El rendimiento del algoritmo es relativamente robusto al establecer el alfa, lo que significa que establecerlo no es crucial para un buen rendimiento. Sin embargo, ajustarlo suele mejorar ligeramente la precisión.

GaussianNB se utiliza principalmente en datos de muy alta dimensión, mientras que las otras dos variantes de Naive Bayes se utilizan ampliamente para datos de recuento dispersos, como texto. MultinomialNB suele tener un mejor rendimiento que BinaryNB, especialmente en conjuntos de datos con un número relativamente grande de características distintas de cero (es decir, documentos extensos).

Los modelos bayesianos ingenuos comparten muchas de las fortalezas y debilidades de los modelos lineales. Son muy rápidos de entrenar y predecir, y el procedimiento de entrenamiento es fácil de entender. Funcionan muy bien con datos dispersos de alta dimensión y son relativamente robustos a los parámetros. Los modelos bayesianos ingenuos son excelentes modelos de referencia y se utilizan a menudo en conjuntos de datos muy grandes, donde el entrenamiento incluso de un modelo lineal podría ser demasiado largo.

Árboles de decisión

Los árboles de decisión son modelos ampliamente utilizados en tareas de clasificación y regresión. En esencia, aprenden una jerarquía de preguntas condicionales (if/else), lo que lleva a una decisión.

Estas preguntas son similares a las preguntas que podrías hacer en un juego de 20 preguntas.
Imagina que quieres distinguir entre los siguientes cuatro animales: osos, halcones, pingüinos y delfines. Tu objetivo es llegar a la respuesta correcta haciendo la menor cantidad posible de preguntas condicionales. Podrías empezar preguntando si el animal tiene plumas, una pregunta que reduce tus posibles animales a solo dos. Si la respuesta es "sí", puedes hacer otra pregunta que podría ayudarte a distinguir entre halcones y pingüinos. Por ejemplo, podrías preguntar si el animal puede volar. Si el animal no tiene plumas, tus posibles opciones son delfines y osos, y tendrás que hacer una pregunta para distinguir entre estos dos animales, por ejemplo, preguntar si el animal tiene aletas.

Esta serie de preguntas se puede expresar como un árbol de decisiones, como se muestra en la Figura 2-22.

En[56]:

```
mglearn.plots.plot_animal_tree()
```

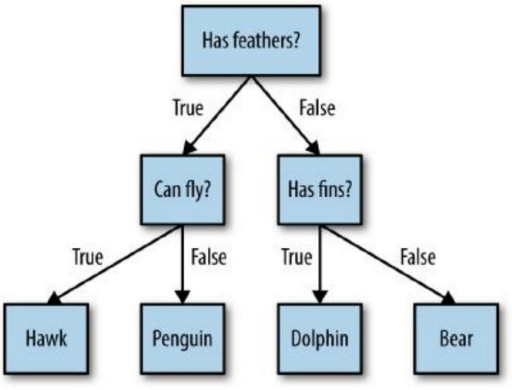

Figura 2-22. Un árbol de decisión para distinguir entre varios animales.

En esta ilustración, cada nodo del árbol representa una pregunta o un nodo terminal (también llamado hoja) que contiene la respuesta. Los bordes conectan las respuestas a una pregunta con la siguiente pregunta que se formularía.

En el lenguaje del aprendizaje automático, construimos un modelo para distinguir entre cuatro clases de animales (halcones, pingüinos, delfines y osos) utilizando las tres características "tiene plumas", "puede volar" y "tiene aletas". En lugar de construir estos modelos manualmente, podemos aprenderlos a partir de datos mediante aprendizaje supervisado.

Creación de árboles de

decisión. Analicemos el proceso de creación de un árbol de decisión para el conjunto de datos de clasificación 2D que se muestra en la Figura 2-23. El conjunto de datos consta de dos medialunas, y cada clase consta de 75 puntos de datos. Nos referiremos a este conjunto de datos como two_moons.

Aprender un árbol de decisión implica aprender la secuencia de preguntas condicionales que nos lleva a la respuesta verdadera más rápidamente. En el aprendizaje automático, estas preguntas se denominan pruebas (no deben confundirse con el conjunto de pruebas, que son los datos que utilizamos para comprobar la generalización de nuestro modelo). Normalmente, los datos no se presentan en forma de características binarias de sí/no, como en el ejemplo del animal, sino como características continuas, como en el conjunto de datos 2D que se muestra en la Figura 2-23. Las pruebas que se utilizan con datos continuos son del tipo "¿Es la característica i mayor que el valor a?".

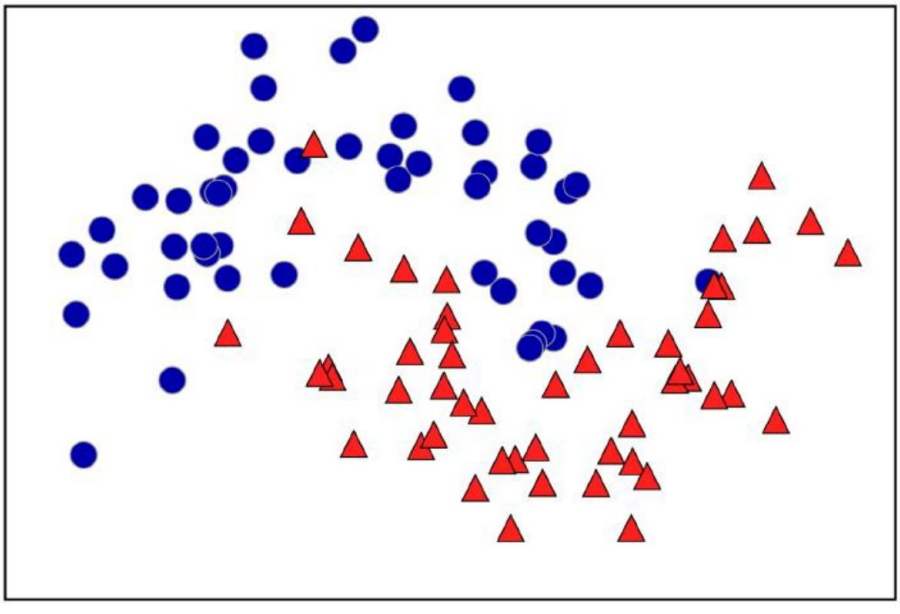

Figura 2-23. Conjunto de datos de dos lunas sobre el que se construirá el árbol de decisión.

Para construir un árbol, el algoritmo busca entre todas las pruebas posibles y encuentra la que ofrece mayor información sobre la variable objetivo. La Figura 2-24 muestra la primera prueba seleccionada. Dividir el conjunto de datos verticalmente en x[1]=0,0596 proporciona la mayor cantidad de información; separa mejor los puntos de la clase 1 de los de la clase 2. El nodo superior, también llamado raíz, representa el conjunto de datos completo, compuesto por 75 puntos de la clase 0 y 75 de la clase 1. La división se realiza comprobando si x[1] <= 0,0596, indicado por una línea negra. Si la prueba es verdadera, se asigna un punto al nodo izquierdo, que contiene 2 puntos pertenecientes a la clase 0 y 32 puntos pertenecientes a la clase 1. De lo contrario, el punto se asigna al nodo derecho, que contiene 48 puntos pertenecientes a la clase 0 y 18 puntos pertenecientes a la clase 1. Estos dos nodos corresponden a las regiones superior e inferior que se muestran en la Figura 2-24. Aunque la primera división hizo un buen trabajo al separar las dos clases, la región inferior aún contiene puntos pertenecientes a la clase 0 y la región superior aún contiene puntos pertenecientes a la clase 1. Podemos construir un modelo más preciso repitiendo el proceso de búsqueda de la mejor prueba en ambas regiones.

La figura 2-25 muestra que la siguiente división más informativa para la región izquierda y derecha se basa en x[0].

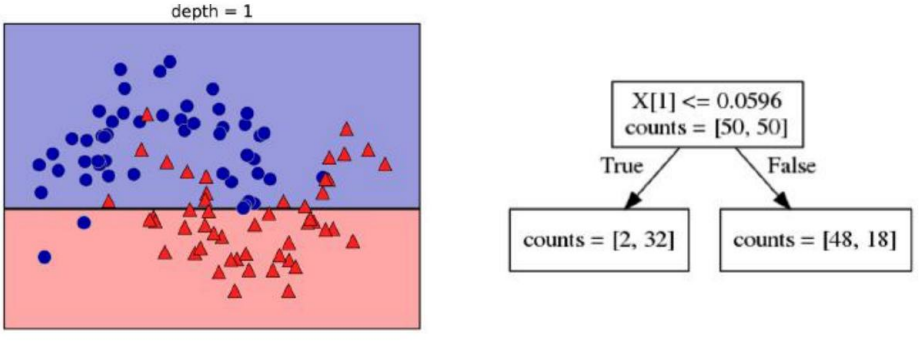

Figura 2-24. Límite de decisión del árbol con profundidad 1 (izquierda) y árbol correspondiente (derecha)

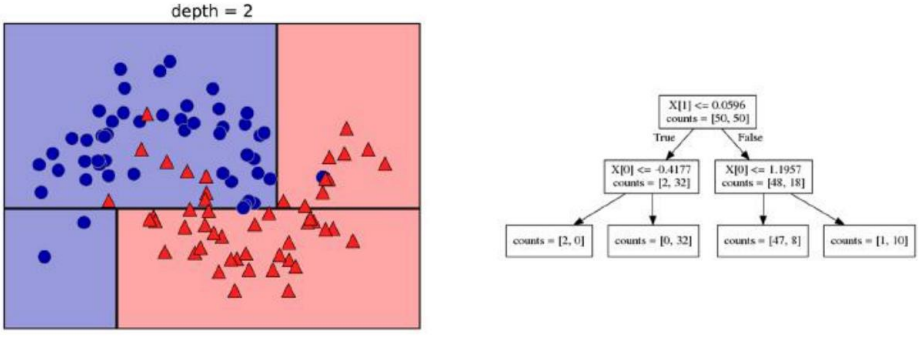

Figura 2-25. Límite de decisión del árbol con profundidad 2 (izquierda) y árbol de decisión correspondiente (derecha)

Este proceso recursivo genera un árbol binario de decisiones, donde cada nodo contiene una prueba. Alternativamente, se puede considerar que cada prueba divide la parte de los datos que se está considerando a lo largo de un eje. Esto permite visualizar el algoritmo como una partición jerárquica. Como cada prueba se refiere a una sola característica, las regiones de la partición resultante siempre tienen límites paralelos a los ejes.

La partición recursiva de los datos se repite hasta que cada región de la partición (cada hoja del árbol de decisión) contenga un único valor objetivo (una sola clase o un único valor de regresión). Una hoja del árbol que contiene puntos de datos que comparten el mismo valor objetivo se denomina pura. La partición final de este conjunto de datos se muestra en la Figura 2-26.

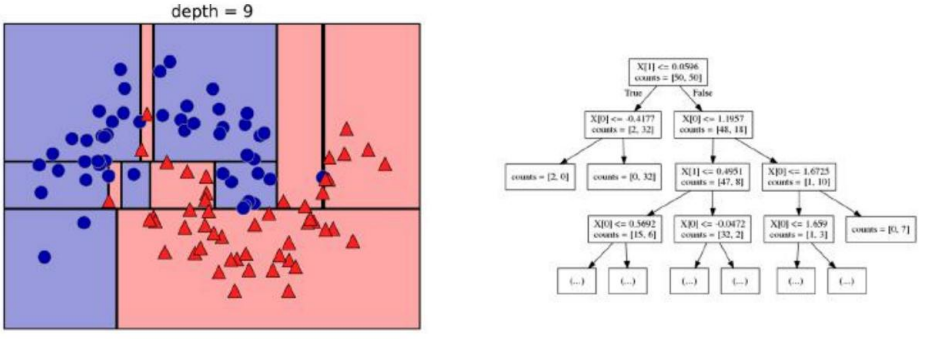

Figura 2-26. Límite de decisión de un árbol con profundidad 9 (izquierda) y parte del árbol correspondiente (derecha); el árbol completo es bastante grande y difícil de visualizar.

Se realiza una predicción sobre un nuevo punto de datos comprobando en qué región de la partición del espacio de características se encuentra dicho punto y, a continuación, prediciendo el objetivo mayoritario (o el objetivo único en el caso de hojas puras) en esa región. La región se puede encontrar recorriendo el árbol desde la raíz y desplazándose hacia la izquierda o la derecha, según se cumpla la prueba.

También es posible utilizar árboles para tareas de regresión, utilizando exactamente la misma técnica. Para realizar una predicción, recorremos el árbol basándonos en las pruebas de cada nodo y encontramos la hoja donde se ubica el nuevo punto de datos. El resultado de este punto de datos es el objetivo medio de los puntos de entrenamiento en esta hoja.

Controlar la complejidad de los árboles de

decisión Normalmente, construir un árbol como se describe aquí y continuar hasta que todas las hojas sean puras conduce a modelos que son muy complejos y altamente sobreajustados a los datos de entrenamiento. La presencia de hojas puras significa que un árbol es 100% preciso en el conjunto de entrenamiento; cada punto de datos en el conjunto de entrenamiento está en una hoja que tiene la clase mayoritaria correcta. El sobreajuste se puede ver a la izquierda de la Figura 2-26. Puede ver las regiones determinadas para pertenecer a la clase 1 en el medio de todos los puntos que pertenecen a la clase 0. Por otro lado, hay una pequeña franja predicha como clase 0 alrededor del punto que pertenece a la clase 0 a la derecha. Así no es como uno imaginaría que se vería el límite de decisión, y el límite de decisión se centra mucho en puntos atípicos individuales que están lejos de los otros puntos de esa clase.

Hay dos estrategias comunes para evitar el sobreajuste: detener la creación del árbol de forma anticipada (también llamado poda previa) o construir el árbol pero luego eliminar o colapsar los nodos que contienen poca información (también llamado poda posterior o simplemente poda).
Los criterios posibles para la poda previa incluyen limitar la profundidad máxima del árbol, limitar el número máximo de hojas o requerir un número mínimo de puntos en un nodo para seguir dividiéndolo.

Los árboles de decisión en scikit-learn se implementan en las clases DecisionTreeRegressor y DecisionTreeClassifier . scikit-learn solo implementa la poda previa, no la poda posterior.

Analicemos con más detalle el efecto de la poda previa en el conjunto de datos de cáncer de mama. Como siempre, importamos el conjunto de datos y lo dividimos en una parte de entrenamiento y otra de prueba. A continuación, construimos un modelo con la configuración predeterminada de desarrollo completo del árbol (haciendo crecer el árbol hasta que todas las hojas sean puras). Corregimos el valor de random_state en el árbol, que se utiliza para el desempate interno:

En[58]:

```
desde sklearn.tree importar DecisionTreeClassifier

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=42)
árbol = DecisionTreeClassifier(random_state=0) árbol.fit(X_train,
y_train) print("Precisión en el conjunto
de entrenamiento: {:.3f}".format(tree.score(X_train, y_train))) print("Precisión en el conjunto de prueba:
{:.3f}".format(tree.score(X_test, y_test)))
```

Salida[58]:

```
Precisión en el conjunto de entrenamiento: 1.000
Precisión en el conjunto de prueba: 0.937
```

Como era de esperar, la precisión del conjunto de entrenamiento es del 100 %. Dado que las hojas son puras, el árbol creció lo suficiente como para memorizar perfectamente todas las etiquetas de los datos de entrenamiento. La precisión del conjunto de prueba es ligeramente inferior a la de los modelos lineales analizados anteriormente, que tenían una precisión cercana al 95 %.

Si no restringimos la profundidad de un árbol de decisión, este puede adquirir una profundidad y complejidad arbitrarias. Por lo tanto, los árboles sin podar son propensos al sobreajuste y a no generalizar adecuadamente a nuevos datos. Ahora, apliquemos la poda previa al árbol, lo que detendrá su desarrollo antes de que se ajuste perfectamente a los datos de entrenamiento. Una opción es detener la construcción del árbol una vez alcanzada cierta profundidad. En este caso, establecemos max_depth=4, lo que significa que solo se pueden formular cuatro preguntas consecutivas (véanse las figuras 2-24 y 2-26). Limitar la profundidad del árbol reduce el sobreajuste. Esto conlleva una menor precisión en el conjunto de entrenamiento, pero una mejora en el conjunto de prueba.

En[59]:

```
árbol = DecisionTreeClassifier(profundidad_máxima=4, estado_aleatorio=0)
árbol.fit(entrenamiento_X, entrenamiento_y)

print("Precisión en el conjunto de entrenamiento: {:.3f}".format(tree.score(X_train, y_train))) print("Precisión en el
conjunto de prueba: {:.3f}".format(tree.score(X_test, y_test)))
```

Salida[59]:

```
Precisión en el conjunto de entrenamiento: 0,988
Precisión en el conjunto de prueba: 0,951
```

Análisis de árboles de

decisión Podemos visualizar el árbol utilizando la función export_graphviz del módulo árbol .
Esto escribe un archivo en el formato de archivo .dot, que es un formato de archivo de texto para almacenar gráficos.
Establecemos una opción para colorear los nodos para reflejar la clase mayoritaria en cada nodo y pasamos los
nombres de clase y características para que el árbol pueda etiquetarse correctamente:

En[61]:

```
de sklearn.tree importar export_graphviz
export_graphviz(tree, archivo_de_salida="tree.dot", nombres_de_clase=["maligno", "benigno"],
                feature_names=cáncer.feature_names, impureza=Falso, relleno=Verdadero)
```

Podemos leer este archivo y visualizarlo, como se ve en la Figura 2-27, usando el módulo graphviz (o puede usar
cualquier programa que pueda leer archivos .dot):

En[61]:

```
importar graphviz

con open("tree.dot") como f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

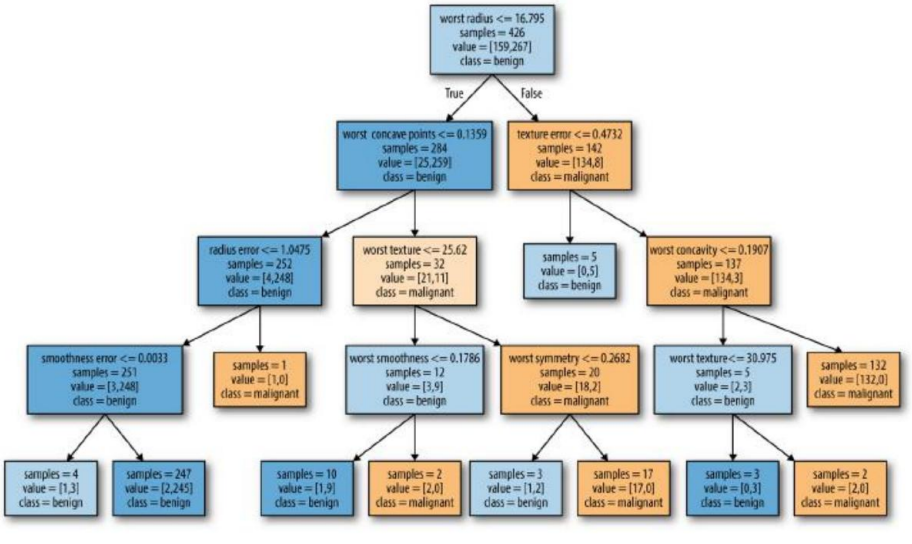

Figura 2-27. Visualización del árbol de decisiones basado en el conjunto de datos de cáncer de mama.

La visualización del árbol proporciona una excelente visión en profundidad de cómo funciona el algoritmo.
Hace predicciones y es un buen ejemplo de un algoritmo de aprendizaje automático que es
Fácil de explicar para quienes no son expertos. Sin embargo, incluso con un árbol de profundidad cuatro, como se ve aquí,
El árbol puede resultar un poco abrumador. Árboles más profundos (una profundidad de 10 no es infrecuente)
mon) son aún más difíciles de comprender. Un método para inspeccionar el árbol que puede ser útil
es averiguar qué ruta sigue realmente la mayoría de los datos. Las n_muestras que se muestran en
cada nodo en la Figura 2-27 proporciona el número de muestras en ese nodo, mientras que el valor pro
Proporciona el número de muestras por clase. Siguiendo las ramas a la derecha, vemos
ese peor radio <= 16.795 crea un nodo que contiene solo 8 benignos pero 134
muestras malignas. El resto de este lado del árbol utiliza distinciones más sutiles.
Para separar estas 8 muestras benignas restantes. De las 142 muestras que fueron a la
Justo en la división inicial, casi todos ellos (132) terminan en la hoja de la extrema derecha.

Tomando a la izquierda en la raíz, para el peor radio > 16.795 terminamos con 25 malignos
y 259 muestras benignas. Casi todas las muestras benignas terminan en la segunda hoja.
desde la derecha, con la mayoría de las otras hojas conteniendo muy pocas muestras.

Importancia de las características en los árboles

En lugar de mirar el árbol completo, lo cual puede resultar agotador, hay algunas propuestas útiles:
propiedades que podemos derivar para resumir el funcionamiento del árbol. Las más comunes
El resumen usado es la importancia de las características, que califica la importancia de cada característica para
La decisión que toma un árbol. Es un número entre 0 y 1 para cada característica, donde 0
significa “no se usa en absoluto” y 1 significa “predice perfectamente el objetivo”. La característica
Las importancias siempre suman 1:

En[62]:

```
print(" Importancias de las características:\n{}".format(tree.feature_importances_))
```

Salida[62]:

```
Importancia de las características:
[ 0.      0.      0.      0.      0.      0.      0.      0.      0.      0.01
  0.048 0. 0. 0.    0.      0.002 0. 0. 0.018 0.122 0.012 0.0.      0.      0,727 0,046
                    0.014 0.                                        ]
```

Podemos visualizar la importancia de las características de una manera similar a como las visualizamos.
izar los coeficientes en el modelo lineal (Figura 2-28):

En[63]:

```
def plot_feature_importances_cancer(modelo):
    n_características = cáncer.datos.forma[1]
    plt.barh(rango(n_características), modelo.características_importancias_, align='centro')
    plt.yticks(np.arange(n_características), cáncer.nombres_de_las_características)
    plt.xlabel(" Importancia de la característica")
    plt.ylabel("Característica")

plot_feature_importances_cancer(árbol)
```

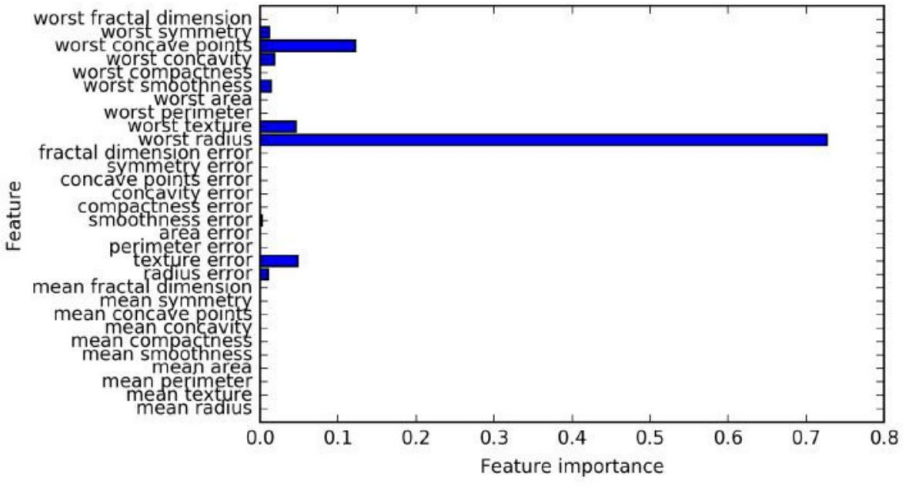

Figura 2-28. Importancia de las características calculada a partir de un árbol de decisión aprendido en el conjunto de datos de cáncer de mama.

Aquí vemos que la característica utilizada en la división superior (el peor radio) es, con diferencia, la más importante. Esto confirma nuestra observación al analizar el árbol: el primer nivel ya separa bastante bien las dos clases.

Sin embargo, si una característica tiene una importancia_de_característica baja , esto no significa que sea poco informativa. Simplemente significa que el árbol no la seleccionó, probablemente porque otra característica codifica la misma información.

A diferencia de los coeficientes en los modelos lineales, la importancia de las características siempre es positiva y no indica la clase a la que corresponde una característica. La importancia de las características nos indica que el "peor radio" es importante, pero no si un radio alto indica que una muestra es benigna o maligna. De hecho, podría no existir una relación tan simple entre las características y la clase, como se puede observar en el siguiente ejemplo (Figuras 2-29 y 2-30):

En[64]:

```
árbol = mglearn.plots.plot_tree_not_monotone() mostrar(árbol)
```

Salida[64]:

```
Importancia de las características: [ 0. 1.]
```

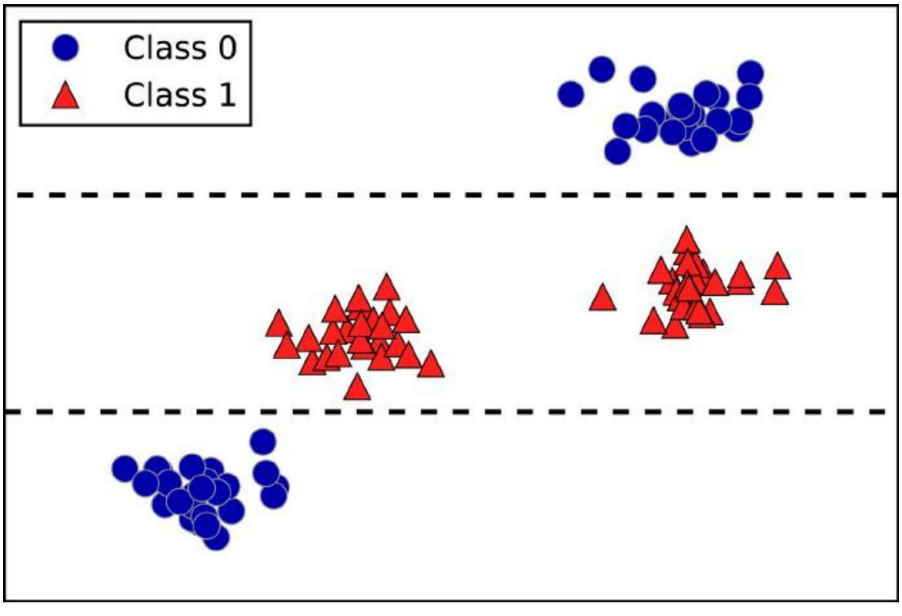

Figura 2-29. Un conjunto de datos bidimensional en el que la característica en el eje y tiene una relación no monótona con la etiqueta de clase, y los límites de decisión encontrados por un deci-
árbol de sion

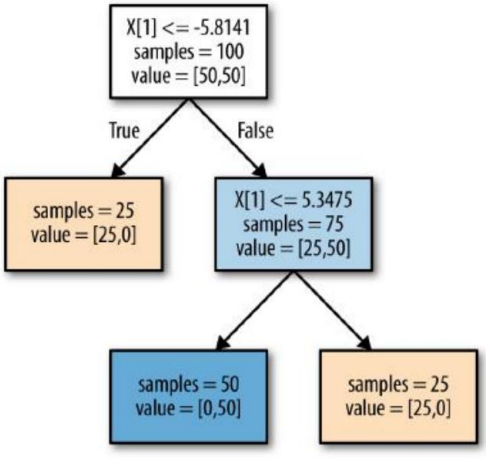

Figura 2-30. Árbol de decisión aprendido con los datos de la Figura 2-29

El gráfico muestra un conjunto de datos con dos características y dos clases. Aquí, toda la información está contenida en X[1], y X[0] no se utiliza en absoluto. Sin embargo, la relación entre X[1] y

La clase de salida no es monótona, lo que significa que no podemos decir “un valor alto de X[0] significa clase 0 y un valor bajo significa clase 1” (o viceversa).

Si bien centramos nuestra discusión en los árboles de decisión para la clasificación, todo lo anterior es igualmente válido para los árboles de decisión para la regresión, tal como se implementan en Decision TreeRegressor. El uso y análisis de los árboles de regresión es muy similar al de los árboles de clasificación. Sin embargo, hay una propiedad particular del uso de modelos basados en árboles para la regresión que queremos destacar: DecisionTreeRegressor (y todos los demás modelos de regresión basados en árboles) no puede extrapolar ni realizar predicciones fuera del rango de los datos de entrenamiento.

Analicemos esto con más detalle utilizando un conjunto de datos de precios históricos de memorias de computadora (RAM). La Figura 2-31 muestra el conjunto de datos, con la fecha en el eje x y el precio de un megabyte de RAM en ese año en el eje y:

En[65]:

```
importar pandas como
pd ram_prices = pd.read_csv("data/ram_price.csv")

plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Año")
plt.ylabel("Precio en $/Mbyte")
```

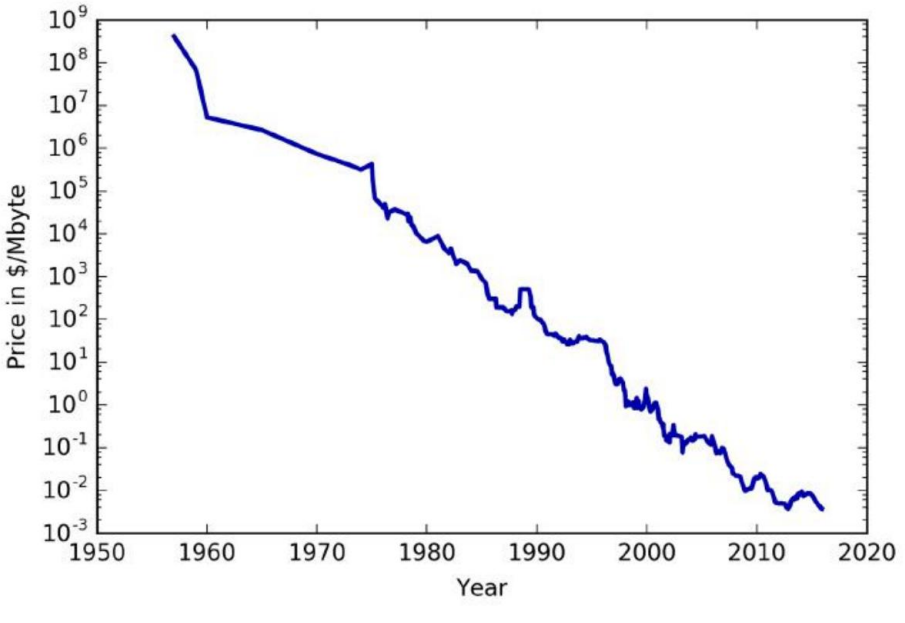

Figura 2-31. Evolución histórica del precio de la RAM, representada en escala logarítmica.

Observe la escala logarítmica del eje y. Al graficar logarítmicamente, la relación parece ser bastante lineal y, por lo tanto, debería ser relativamente fácil de predecir, salvo algunas irregularidades.

Realizaremos un pronóstico para los años posteriores al 2000 utilizando los datos históricos hasta ese momento, con la fecha como único parámetro. Compararemos dos modelos simples: un Regresor de Árbol de Decisiones y una Regresión Lineal. Reescalamos los precios mediante un logaritmo, de modo que la relación sea relativamente lineal. Esto no supone una diferencia para el Regresor de Árbol de Decisiones, pero sí una gran diferencia para la Regresión Lineal (lo abordaremos con más detalle en el Capítulo 4). Tras entrenar los modelos y realizar predicciones, aplicamos el mapa exponencial para deshacer la transformación logarítmica. Realizamos predicciones sobre todo el conjunto de datos para fines de visualización, pero para una evaluación cuantitativa solo consideraríamos el conjunto de datos de prueba:

En[66]:

```
de sklearn.tree import DecisionTreeRegressor # usa datos
históricos para pronosticar precios después del año 2000 data_train =
ram_prices[ram_prices.date < 2000] data_test =
ram_prices[ram_prices.date >= 2000]

# predecir precios basados en la fecha
X_train = data_train.date[:, np.newaxis] # usamos
una transformación logarítmica para obtener una relación más simple entre los datos y el
objetivo y_train = np.log(data_train.price)

árbol = DecisionTreeRegressor().fit(X_train, y_train) linear_reg =
LinearRegression().fit(X_train, y_train)

# predecir todos los datos
X_all = ram_prices.date[:, np.newaxis]

pred_tree = árbol.predict(X_all) pred_lr
= lineal_reg.predict(X_all)

# deshacer la
transformación logarítmica price_tree
= np.exp(pred_tree) price_lr = np.exp(pred_lr)
```

La figura 2-32, creada aquí, compara las predicciones del árbol de decisión y el modelo de regresión lineal con la verdad fundamental:

En[67]:

```
plt.semilogy(data_train.date, data_train.price, label="Datos de entrenamiento")
plt.semilogy(data_test.date, data_test.price, label=" Datos de prueba")
plt.semilogy(ram_prices.date, price_tree, label=" Predicción de árbol")
plt.semilogy(ram_prices.date, price_lr, label="Predicción lineal") plt.legend()
```

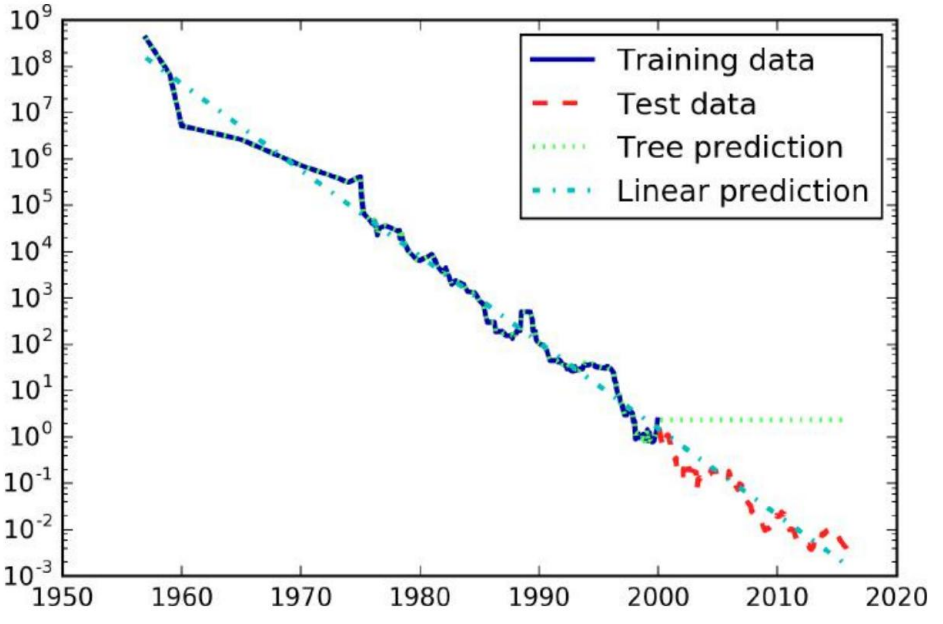

Figura 2-32. Comparación de las predicciones realizadas mediante un modelo lineal y un árbol de regresión sobre los datos de precios de RAM.

La diferencia entre los modelos es bastante notable. El modelo lineal aproxima los datos con una línea, como ya sabíamos. Esta línea proporciona un pronóstico bastante bueno para los datos de prueba (años posteriores al 2000), a la vez que omite algunas de las variaciones más sutiles tanto en los datos de entrenamiento como en los de prueba. El modelo de árbol, por otro lado, realiza predicciones perfectas con los datos de entrenamiento; no restringimos la complejidad del árbol, por lo que aprendió todo el conjunto de datos de memoria. Sin embargo, una vez que salimos del rango de datos para el que el modelo tiene datos, este simplemente continúa prediciendo el último punto conocido. El árbol no tiene la capacidad de generar respuestas "nuevas", más allá de las observadas en los datos de entrenamiento. Esta deficiencia se aplica a todos los modelos basados en árboles.⁹

Fortalezas, debilidades y parámetros. Como

se mencionó anteriormente, los parámetros que controlan la complejidad del modelo en los árboles de decisión son los parámetros de prepoda que detienen la construcción del árbol antes de que esté completamente desarrollado. Por lo general, se elige una de las estrategias de prepoda (establecer...

De hecho , es posible realizar pronósticos muy acertados con modelos basados en árboles (por ejemplo, al intentar predecir si un precio subirá o bajará). El objetivo de este ejemplo no era demostrar que los árboles son un mal modelo para las series temporales, sino ilustrar una propiedad específica de cómo los árboles realizan predicciones.

max_depth, max_leaf_nodes o min_samples_leaf—es suficiente para evitar el sobreajuste.

Los árboles de decisión presentan dos ventajas sobre muchos de los algoritmos que hemos analizado hasta ahora: el modelo resultante puede ser fácilmente visualizado y comprendido por personas sin experiencia (al menos para árboles pequeños), y los algoritmos son completamente invariables al escalado de los datos. Dado que cada característica se procesa por separado y las posibles divisiones de los datos no dependen del escalado, no se requiere preprocesamiento como la normalización o estandarización de características para los algoritmos de árboles de decisión. En particular, los árboles de decisión funcionan bien cuando se tienen características con escalas completamente diferentes, o una combinación de características binarias y continuas.

La principal desventaja de los árboles de decisión es que, incluso con la poda previa, tienden a sobreajustarse y a ofrecer un rendimiento de generalización deficiente. Por lo tanto, en la mayoría de las aplicaciones, los métodos de conjunto que analizaremos a continuación suelen utilizarse en lugar de un único árbol de decisión.

Conjuntos de árboles de decisión = Random Forest. (Cultura gral., no entra)

Los conjuntos son métodos que combinan múltiples modelos de aprendizaje automático para crear modelos más potentes. Existen muchos modelos en la literatura sobre aprendizaje automático que pertenecen a esta categoría, pero hay dos modelos de conjunto que han demostrado ser eficaces en una amplia gama de conjuntos de datos para la clasificación y la regresión. Ambos utilizan árboles de decisión como componentes básicos: bosques aleatorios y árboles de decisión potenciados por gradiente.

Bosques aleatorios

Como acabamos de observar, una de las principales desventajas de los árboles de decisión es que tienden a sobreajustar los datos de entrenamiento. Los bosques aleatorios son una forma de abordar este problema. Un bosque aleatorio es esencialmente un conjunto de árboles de decisión, donde cada árbol es ligeramente diferente de los demás. La idea detrás de los bosques aleatorios es que cada árbol puede tener una buena capacidad de predicción, pero probablemente sobreajustará parte de los datos. Si construimos muchos árboles, todos con buen rendimiento y sobreajustados de diferentes maneras, podemos reducir el sobreajuste promediando sus resultados. Esta reducción del sobreajuste, manteniendo la capacidad de predicción de los árboles, puede demostrarse mediante matemáticas rigurosas.

Para implementar esta estrategia, necesitamos construir varios árboles de decisión. Cada árbol debe predecir el objetivo de forma aceptable y, además, ser diferente de los demás. Los bosques aleatorios reciben su nombre porque inyectan aleatoriedad en la construcción del árbol para garantizar que cada uno sea diferente. Hay dos maneras de aleatorizar los árboles de un bosque aleatorio: seleccionando los puntos de datos utilizados para construir un árbol y seleccionando las características en cada prueba dividida. Analicemos este proceso con más detalle.

Capítulo 6. Árboles de decisión

Los árboles de decisión son algoritmos versátiles de aprendizaje automático que pueden realizar tareas de clasificación y regresión, e incluso tareas multisalida. Son algoritmos potentes, capaces de ajustar conjuntos de datos complejos. Por ejemplo, en el capítulo 2, entrenó un modelo DecisionTreeRegressor con el conjunto de datos de vivienda de California, ajustándolo perfectamente (de hecho, sobreajustándolo).

Los árboles de decisión también son los componentes fundamentales de los bosques aleatorios (véase el Capítulo 7), que se encuentran entre los algoritmos de aprendizaje automático más potentes disponibles en la actualidad.

En este capítulo, comenzaremos explicando cómo entrenar, visualizar y realizar predicciones con árboles de decisión. A continuación, repasaremos el algoritmo de entrenamiento CART utilizado por Scikit-Learn y exploraremos cómo regularizar árboles y utilizarlos en tareas de regresión. Finalmente, analizaremos algunas de las limitaciones de los árboles de decisión.

Entrenamiento y visualización de un árbol de decisiones

Para comprender los árboles de decisión, construyamos uno y veamos cómo genera predicciones. El siguiente código entrena un DecisionTreeClassifier en el conjunto de datos iris (véase el Capítulo 4):

```
desde sklearn.datasets importar load_iris desde
sklearn.tree importar DecisionTreeClassifier

iris = cargar_iris(como_marco=Verdadero)
X_iris = iris.data[[" longitud del pétalo (cm)", "ancho del pétalo (cm)"]].values y_iris
= iris.target

árbol_clf = ClasificadorDeÁrbolDeDecisiones(profundidad_máxima=2,
estado_aleatorio=42) árbol_clf.fit(X_iris, y_iris)
```

Puede visualizar el árbol de decisiones entrenado utilizando primero la función export_graphviz() para generar un archivo de definición de gráfico llamado iris_tree.dot:

```
desde sklearn.tree importar export_graphviz

export_graphviz( tree_clf,

        out_file="iris_tree.dot",
        features_names=[" longitud del pétalo (cm)", "ancho del pétalo (cm)"],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

Luego puedes usar graphviz.Source.from_file() para cargar y mostrar el archivo en un cuaderno Jupyter:

```
desde graphviz import Fuente

Fuente.from_file("iris_tree.dot")
```

Visualización gráfica Es un paquete de software de visualización de gráficos de código abierto. También...

Incluye una herramienta de línea de comandos para convertir archivos .dot a una variedad de formatos, como PDF o PNG.

Su primer árbol de decisiones se parece a la Figura 6-1.

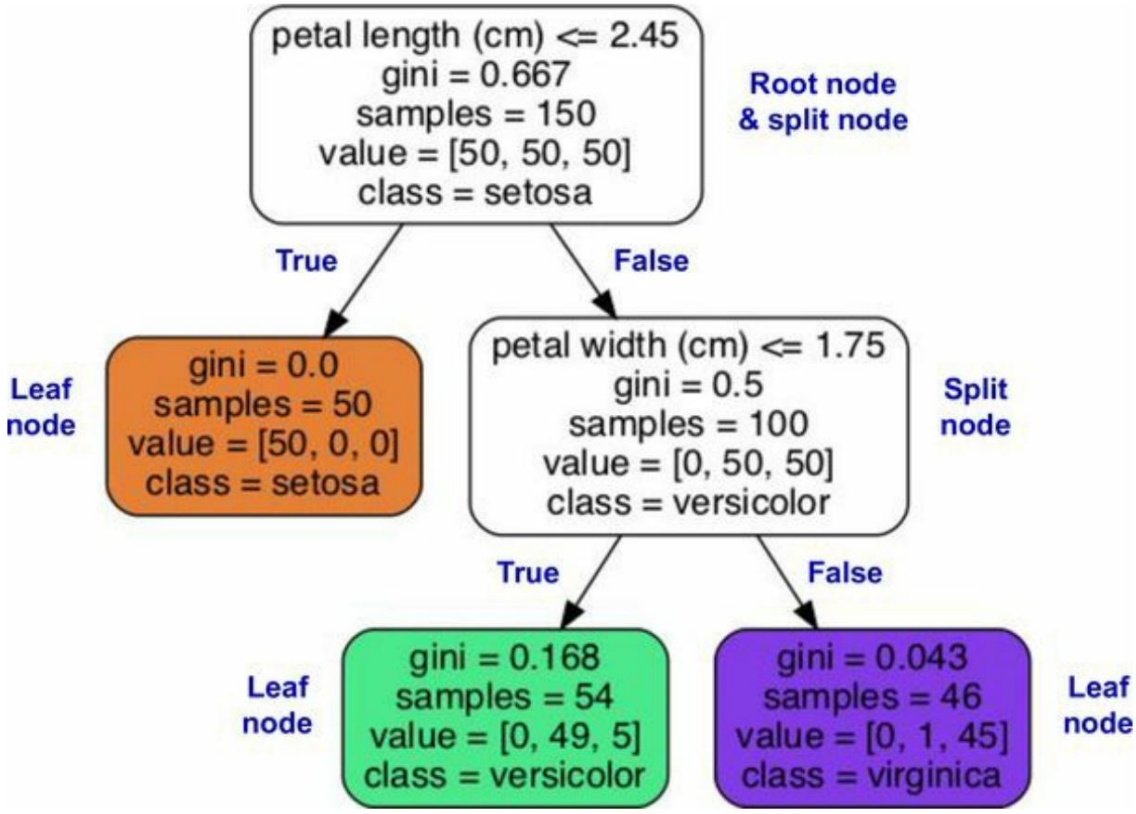

Figura 6-1. Árbol de decisión de Iris

Haciendo predicciones

Veamos cómo el árbol representado en la Figura 6-1 hace predicciones. Supongamos que encuentra una flor de iris y desea clasificarla según sus pétalos. Comienza en el nodo raíz (profundidad 0, en la parte superior): este nodo pregunta si la longitud del pétalo de la flor es menor de 2,45 cm. Si lo es, entonces se mueve hacia abajo al nodo hijo izquierdo de la raíz (profundidad 1, izquierda). En este caso, es un nodo hoja (es decir, no tiene ningún nodo hijo), por lo que no hace ninguna pregunta: simplemente mira la clase predicha para ese nodo, y el árbol de decisión predice que tu flor es una Iris setosa (clase=setosa).

Ahora supongamos que encuentras otra flor, y esta vez la longitud del pétalo es mayor de 2,45 cm. Comienzas de nuevo en la raíz, pero ahora desciendes hasta su nodo hijo derecho (profundidad 1, derecha). Este no es un nodo de hoja, sino un nodo dividido, por lo que plantea otra pregunta: ¿el ancho del pétalo es menor de 1,75 cm? Si es así, lo más probable es que tu flor sea un Iris versicolor (profundidad 2, izquierda). Si no, probablemente sea un Iris virginica (profundidad 2, derecha). Es así de simple.

> NOTA
>
> Una de las muchas cualidades de los árboles de decisión es que requieren muy poca preparación de datos. De hecho, no requieren escalamiento ni centrado de funciones en absoluto.

El atributo de muestras de un nodo cuenta a cuántas instancias de entrenamiento se aplica. Por ejemplo, 100 instancias de entrenamiento tienen una longitud de pétalo mayor a 2,45 cm (profundidad 1, derecha), y de esas 100, 54 tienen un ancho de pétalo menor a 1,75 cm (profundidad 2, izquierda). El atributo de valor de un nodo indica a cuántas instancias de entrenamiento de cada clase se aplica este nodo: por ejemplo, el nodo inferior derecho se aplica a 0 Iris setosa, 1 Iris versicolor y 45 Iris virginica. Finalmente, el atributo Gini de un nodo mide su impureza de Gini: un nodo es "puro" (gini=0) si todas las instancias de entrenamiento a las que se aplica pertenecen a la misma clase. Por ejemplo, dado que el nodo izquierdo de profundidad 1 se aplica solo a las instancias de entrenamiento de Iris setosa , es puro y su impureza de Gini es 0. La ecuación 6-1 muestra cómo el algoritmo de entrenamiento ca

el

Impureza de Gini G_i del nodo i^{el}. El nodo izquierdo de profundidad 2 tiene una impureza de Gini igual a $1 – (0/54)^2 – (49/54)^2 – (5/54)^2 \approx 0,168$.

Ecuación 6-1. Impureza de Gini

$$G_i = 1 - \sum_{k=1} n p_{i,k}^2$$

En esta ecuación:

- G_i es la impureza de Gini del nodo i^{el}.
- p es la relación de instancias de clase k entre las instancias de entrenamiento en el nodo i^{el} $_{i,k}$.

> NOTA
>
> Scikit-Learn utiliza el algoritmo CART, que produce únicamente árboles binarios; es decir, árboles donde los nodos divididos siempre tienen exactamente dos hijos (es decir, las preguntas solo tienen respuestas de sí/no). Sin embargo, otros algoritmos, como ID3, pueden producir árboles de decisión con nodos que tienen más de dos hijos.

La Figura 6-2 muestra los límites de decisión de este árbol de decisión. La línea vertical gruesa representa el límite de decisión del nodo raíz (profundidad 0): longitud del pétalo = 2,45 cm. Dado que el área izquierda es pura (solo Iris setosa), no se puede dividir más. Sin embargo, el área derecha es impura, por lo que el nodo derecho de profundidad 1 la divide en la anchura del pétalo = 1,75 cm (representada por la línea discontinua). Dado que max_depth se estableció en 2, el árbol de decisión se detiene ahí. Si se establece max_depth en 3, los dos nodos de profundidad 2 añadirían cada uno otro límite de decisión (representado por las dos líneas verticales de puntos).

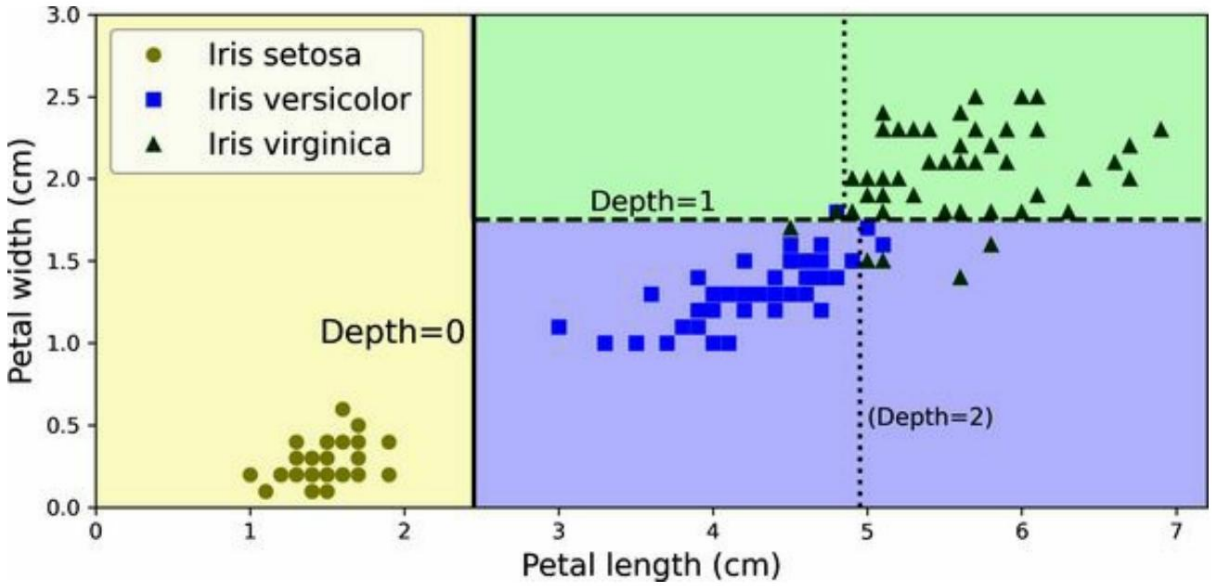

Figura 6-2. Límites de decisión del árbol de decisión

CONSEJO

La estructura de árbol, que incluye toda la información mostrada en la Figura 6-1, está disponible mediante el atributo tree_ del clasificador. Escriba help(tree_clf.tree_) para obtener más detalles y consulte el cuaderno de notas de este capítulo para ver un ejemplo.

INTERPRETACIÓN DEL MODELO: CAJA BLANCA VERSUS CAJA NEGRA CAJA

Los árboles de decisión son intuitivos y sus decisiones son fáciles de interpretar. Estos modelos suelen denominarse modelos de caja blanca. En cambio, como verá, los bosques aleatorios y las redes neuronales suelen considerarse modelos de caja negra. Realizan predicciones excelentes, y puede comprobar fácilmente los cálculos que realizaron para obtenerlas; sin embargo, suele ser difícil explicar de forma sencilla por qué se realizaron. Por ejemplo, si una red neuronal indica que una persona en particular aparece en una imagen, es difícil saber qué contribuyó a esta predicción: ¿Reconoció el modelo los ojos de esa persona? ¿Su boca? ¿Su nariz?

¿Sus zapatos? ¿O incluso el sofá en el que estaban sentados? Por el contrario, los árboles de decisión proporcionan reglas de clasificación sencillas y útiles que incluso pueden...

Se puede aplicar manualmente si es necesario (p. ej., para la clasificación de flores). El campo del aprendizaje automático interpretable busca crear sistemas de aprendizaje automático que expliquen sus decisiones de forma comprensible para los humanos. Esto es importante en muchos ámbitos; por ejemplo, para garantizar que el sistema no tome decisiones injustas.

Estimación de Probabilidades de Clase. Un árbol de decisión también puede estimar la probabilidad de que una instancia pertenezca a una clase k en particular. Primero, recorre el árbol para encontrar el nodo hoja de esta instancia y, a continuación, devuelve la proporción de instancias de entrenamiento de la clase k en este nodo. Por ejemplo, supongamos que ha encontrado una flor cuyos pétalos miden 5 cm de largo y 1,5 cm de ancho. El nodo hoja correspondiente es el nodo izquierdo de profundidad 2, por lo que el árbol de decisión genera las siguientes probabilidades: 0 % para Iris setosa (0/54), 90,7 % para Iris versicolor (49/54) y 9,3 % para Iris virginica (5/54). Y si le pides que prediga la clase, da como resultado Iris versicolor (clase 1) porque tiene la mayor probabilidad. Comprobémoslo:

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
matriz([[0, 0.907, 0.093]])
>>> tree_clf.predict([[5, 1.5]])
matriz([1])
```

¡Perfecto! Observe que las probabilidades estimadas serían idénticas en cualquier otro lugar del rectángulo inferior derecho de la Figura 6-2; por ejemplo, si los pétalos midieran 6 cm de largo y 1,5 cm de ancho (aunque parece obvio que en este caso lo más probable es que se trate de un Iris virginica).

El algoritmo de entrenamiento CART de Scikit-Learn utiliza el algoritmo de Árbol de Clasificación y Regresión (CART) para entrenar árboles de decisión (también llamados árboles de crecimiento). El algoritmo funciona dividiendo primero el conjunto de entrenamiento en dos subconjuntos utilizando una única característica k y un umbral t (p. ej., «longitud del pétalo ≤ 2,45 cm»). ¿Cómo elige k y t ? Busca el par (k, t) que produce los subconjuntos más puros, ponderados por su tamaño. La ecuación 6-2 proporciona la función de coste que el algoritmo intenta minimizar.

Ecuación 6-2. Función de costo CART para la clasificación

$J(k, t_k) = \frac{m_{izquierda}}{m} G_{izquierda} + \frac{m_{derecha}}{m} G_{derecha}$ donde $G_{izquierda/derecha}$ mide la impureza del subconjunto izquierdo/derecho $m_{izquierda/derecha}$ es el número de instancias en el subconjunto izquierdo/derecho

Una vez que el algoritmo CART divide correctamente el conjunto de entrenamiento en dos, divide los subconjuntos utilizando la misma lógica, luego los subconjuntos, y así sucesivamente, recursivamente. Detiene la recursividad al alcanzar la profundidad máxima (definida por el hiperparámetro max_depth) o si no encuentra una división que reduzca la impureza. Otros hiperparámetros (que se describen a continuación) controlan condiciones de detención adicionales: min_samples_split, min_samples_leaf, min_weight_fraction_leaf y max_leaf_nodes.

ADVERTENCIA

Como puede ver, el algoritmo CART es voraz: busca con avidez una división óptima en el nivel superior y repite el proceso en cada nivel subsiguiente. No comprueba si la división resultará en la menor impureza posible varios niveles más abajo. Un algoritmo voraz suele producir una solución razonablemente buena, pero no se garantiza que sea óptima.

Desafortunadamente, encontrar el árbol óptimo es un problema NP-completo [1] . Requiere un tiempo de $O(\exp(m))$, lo que lo hace insoluble incluso para conjuntos de entrenamiento pequeños. Por eso, al entrenar árboles de decisión, debemos conformarnos con una solución razonablemente buena.

Complejidad computacional: Realizar predicciones requiere recorrer el árbol de decisión desde la raíz hasta una hoja. Los árboles de decisión generalmente están aproximadamente equilibrados, por lo que recorrerlos requiere recorrer aproximadamente $O(\log_2(m))$ nodos, donde $\log_2(m)$ es el logaritmo binario de m, igual a log(m) / log(2). Dado que cada nodo solo requiere verificar el valor de una característica, la complejidad total de la predicción es $O(\log_2(m))$, independientemente del número de características. Por lo tanto, las predicciones son muy rápidas, incluso al trabajar con grandes conjuntos de entrenamiento.

El algoritmo de entrenamiento compara todas las características (o menos si se establece max_features) de todas las muestras en cada nodo. Esta comparación resulta en una complejidad de entrenamiento de $O(n \times m \log_2(m))$.

¿Impureza de Gini o entropía?

De forma predeterminada, la clase DecisionTreeClassifier utiliza la medida de impureza de Gini, pero se puede seleccionar la medida de impureza de entropía estableciendo el hiperparámetro criterio en "entropía". El concepto de entropía se originó en la termodinámica como una medida del desorden molecular: la entropía tiende a cero cuando las moléculas están quietas y bien ordenadas. Posteriormente, la entropía se extendió a una amplia variedad de dominios, incluyendo la teoría de la información de Shannon, donde mide el contenido promedio de información de un mensaje, como vimos en el Capítulo 4. La entropía es cero cuando todos los mensajes son idénticos. En aprendizaje automático, la entropía se utiliza frecuentemente como una medida de impureza: la entropía de un conjunto es cero cuando contiene instancias de una sola clase. La ecuación 6-3 muestra la definición de la entropía del i- ésimo nodo. Por ejemplo, el nodo izquierdo de profundidad 2 en la Figura 6-1 tiene una entropía igual a $-(49/54) \log_2(49/54) - (5/54) \log_2(5/54) \approx 0{,}445$.

Ecuación 6-3. Entropía

$$H_i = -\sum_{k=1,\, p_{i,k} \neq 0}^{n} p_{i,k} \log_2(p_{i,k})$$

Entonces, ¿debería usar la impureza de Gini o la entropía? Lo cierto es que, en la mayoría de los casos, no hay una gran diferencia: generan árboles similares. La impureza de Gini se calcula ligeramente más rápido, por lo que es una buena opción predeterminada. Sin embargo, cuando difieren, la impureza de Gini tiende a aislar la clase más frecuente en su propia rama del árbol, mientras que la entropía tiende a producir árboles ligeramente más equilibrados.

Hiperparámetros de regularización Los árboles de

decisión hacen muy pocas suposiciones sobre los datos de entrenamiento (a diferencia de los modelos lineales, que suponen que los datos son lineales, por ejemplo). Si no se restringe, la estructura de árbol se adaptará automáticamente a los datos de entrenamiento, ajustándolos con precisión; de hecho, es muy probable que los sobreajuste. Este tipo de modelo se suele denominar modelo no paramétrico, no porque no tenga parámetros (a menudo tiene muchos), sino porque el número de parámetros no se determina antes del entrenamiento, por lo que la estructura del modelo puede ajustarse estrictamente a los datos. Por el contrario, un modelo paramétrico, como un modelo lineal, tiene un número predeterminado de parámetros, por lo que su grado de libertad es limitado, lo que reduce el riesgo de sobreajuste (pero aumenta el riesgo de subajuste).

Para evitar el sobreajuste de los datos de entrenamiento, es necesario restringir la libertad del árbol de decisión durante el entrenamiento. Como ya sabe, esto se denomina regularización. Los hiperparámetros de regularización dependen del algoritmo utilizado, pero generalmente se puede restringir la profundidad máxima del árbol de decisión. En Scikit-Learn, esto se controla mediante el hiperparámetro max_depth. El valor predeterminado es None, lo que significa ilimitado. Reducir max_depth regularizará el modelo y, por lo tanto, reducirá el riesgo de sobreajuste.

La clase DecisionTreeClassifier tiene algunos otros parámetros que restringen de manera similar la forma del árbol de decisión:

características máximas

Número máximo de características que se evalúan para dividir en cada nodo

máximo de nodos de hoja

Número máximo de nodos de hoja

división de muestras mínimas

Número mínimo de muestras que debe tener un nodo antes de poder dividirse

min_muestras_hoja

Número mínimo de muestras que debe tener un nodo hoja para ser creado

fracción de peso mínimo de la hoja

Igual que min_samples_leaf pero expresado como una fracción del número total de instancias ponderadas

Aumentar los hiperparámetros min_* o reducir los hiperparámetros max_* regularizará el modelo.

> NOTA
>
> Otros algoritmos funcionan entrenando primero el árbol de decisión sin restricciones y luego podando (eliminando) los nodos innecesarios. Un nodo cuyos hijos son todos nodos hoja se considera innecesario si la mejora de pureza que proporciona no es estadísticamente significativa. Se utilizan pruebas estadísticas estándar, como la prueba χ^2 (prueba de chi-cuadrado), para estimar la probabilidad de que la mejora sea puramente resultado del azar (lo que se denomina hipótesis nula). Si esta probabilidad, denominada valor p, es superior a un umbral determinado (normalmente el 5 %, controlado por un hiperparámetro), el nodo se considera innecesario y se eliminan sus hijos. La poda continúa hasta que se hayan podado todos los nodos innecesarios.

Probemos la regularización en el conjunto de datos de lunas, presentado en el Capítulo 5. Entrenaremos un árbol de decisión sin regularización y otro con min_samples_leaf=5. Aquí está el código; la Figura 6-3 muestra los límites de decisión de cada árbol:

```
desde sklearn.datasets importar make_moons

X_lunas, y_lunas = crear_lunas(n_muestras=150, ruido=0,2, estado_aleatorio=42)

árbol_clf1 = ClasificadorÁrbolDeDecisiones(estado_aleatorio=42)
árbol_clf2 = ClasificadorÁrbolDeDecisiones(min_muestras_hoja=5,
estado_aleatorio=42)
árbol_clf1.fit(X_lunas, y_lunas) árbol_clf2.fit(X_lunas, y_lunas)
```

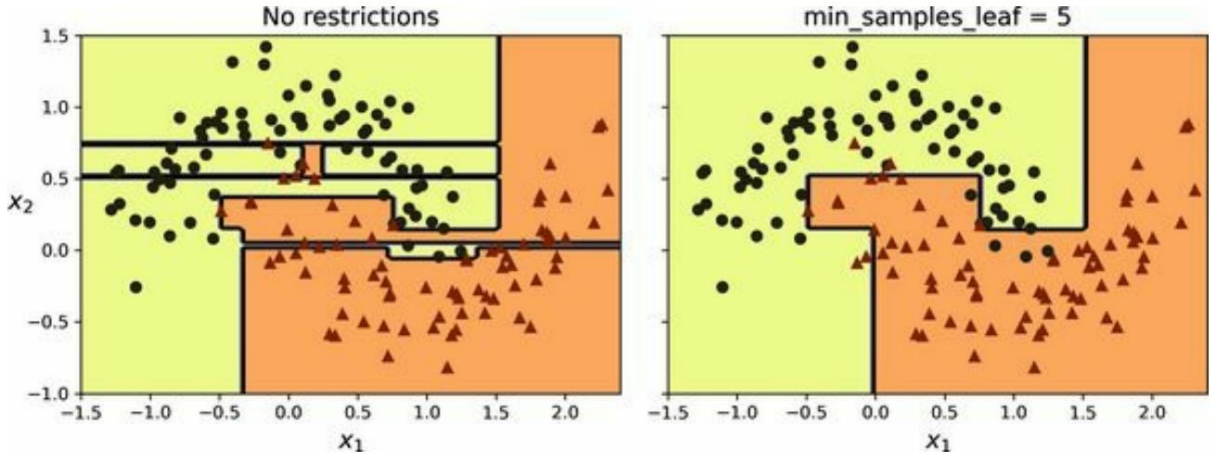

Figura 6-3. Límites de decisión de un árbol no regularizado (izquierda) y un árbol regularizado (derecha)

El modelo no regularizado de la izquierda presenta un claro sobreajuste, y el modelo regularizado de la derecha probablemente generalizará mejor. Podemos verificarlo evaluando ambos árboles en un conjunto de prueba generado con una semilla aleatoria diferente:

```
>>> Prueba_x_luna, prueba_y_luna = crear_luna(n_muestras=1000, ruido=0.2,
...                                           estado_aleatorio=43)
...
>>> tree_clf1.score(prueba_x_lunas, prueba_y_lunas) 0.898

>>> tree_clf2.score(prueba_de_luna_X, prueba_de_luna_y)
0.92
```

De hecho, el segundo árbol tiene una mayor precisión en el conjunto de prueba.

Los árboles de

decisión de regresión también pueden realizar tareas de regresión. Construyamos un árbol de regresión usando la clase DecisionTreeRegressor de Scikit-Learn, entrenándolo con un conjunto de datos cuadráticos con ruido y una profundidad máxima de 2:

```
importar numpy como np
desde sklearn.tree importar DecisionTreeRegressor

np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5 # una única característica de entrada aleatoria
y_quad = X_quad ** 2 + 0.025 * np.random.randn(200, 1)

tree_reg = DecisionTreeRegressor(profundidad_máxima=2, estado_aleatorio=42)
tree_reg.fit(X_cuadrado, y_cuadrado)
```

El árbol resultante se representa en la Figura 6-4.

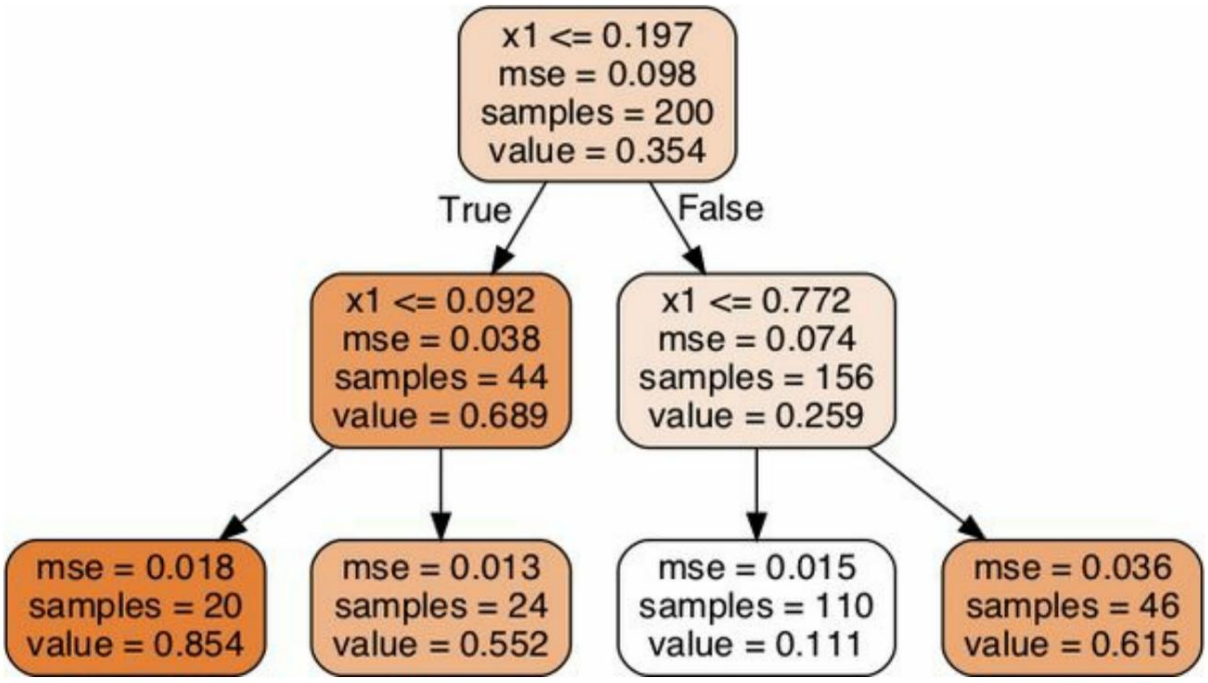

Figura 6-4. Un árbol de decisión para la regresión

Este árbol se parece mucho al árbol de clasificación que construiste anteriormente. La principal diferencia es que, en lugar de predecir una clase en cada nodo, predice un valor. Por ejemplo, supongamos que quieres hacer una predicción para un nuevo...

Instancia con x = 0,12. El nodo raíz pregunta si x ≤ 0,197. Como no lo es, el algoritmo pasa al nodo hijo derecho, que pregunta si x ≤ 0,772. 1
Dado que es así, el algoritmo se dirige al nodo hijo izquierdo. Este es un nodo hoja y predice un valor de 0,111. Esta predicción es el valor objetivo promedio de las 110 instancias de entrenamiento asociadas a este nodo hoja y resulta en un error cuadrático medio de 0,015 para estas 110 instancias.

Las predicciones de este modelo se representan a la izquierda en la Figura 6-5. Si se establece max_depth=3, las predicciones se representan a la derecha. Observe cómo el valor predicho para cada región siempre es el valor objetivo promedio de las instancias en esa región. El algoritmo divide cada región de forma que la mayoría de las instancias de entrenamiento se acerquen lo más posible a ese valor predicho.

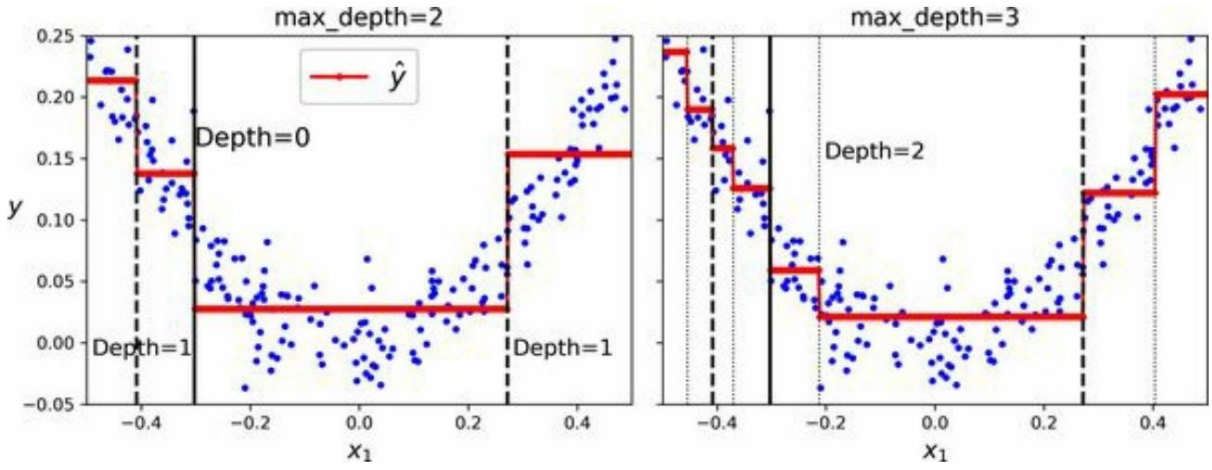

Figura 6-5. Predicciones de dos modelos de regresión de árboles de decisión

El algoritmo CART funciona como se describió anteriormente, excepto que, en lugar de intentar dividir el conjunto de entrenamiento para minimizar las impurezas, ahora intenta dividirlo para minimizar el MSE. La ecuación 6-4 muestra la función de coste que el algoritmo intenta minimizar.

Ecuación 6-4. Función de costo CART para regresión

J(k,tk)=mizquierdamMSEizquierda+mderechamMSEderechadondeMSEnodo=∑i nodo(y^nodo-y(i))2mnodoy^nodo=∑i nodoy(i)mnodo

Al igual que en las tareas de clasificación, los árboles de decisión son propensos al sobreajuste al trabajar con tareas de regresión. Sin regularización (es decir, utilizando los hiperparámetros predeterminados), se obtienen las predicciones de la izquierda en la Figura 6-6.

Estas predicciones obviamente sobreajustan gravemente el conjunto de entrenamiento. Con solo establecer min_samples_leaf=10, se obtiene un modelo mucho más razonable, representado a la derecha en la Figura 6-6.

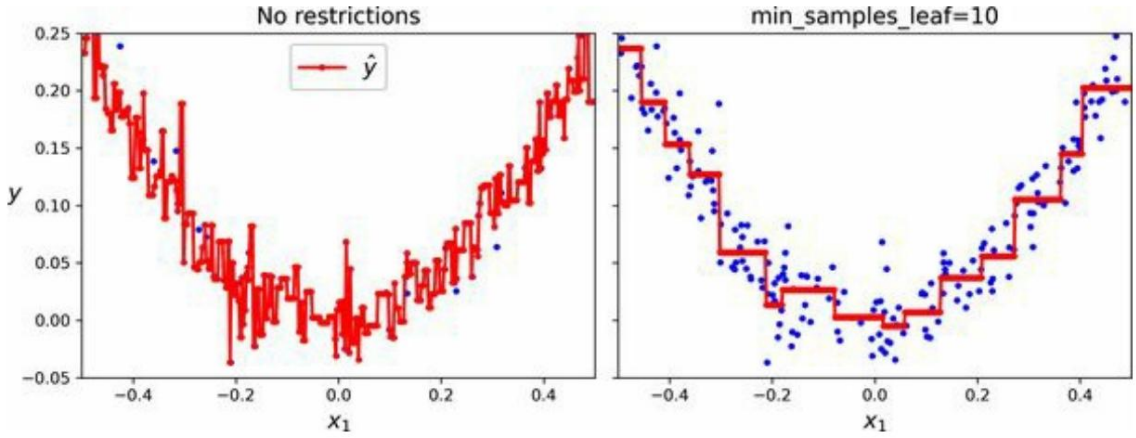

Figura 6-6. Predicciones de un árbol de regresión no regularizado (izquierda) y un árbol regularizado (derecha)

9 árboles de decisión

Un árbol de decisión es una estructura de datos jerárquica que implementa la estrategia de "divide y vencerás". Es un método no paramétrico eficiente, que puede utilizarse tanto para clasificación como para regresión. Analizamos algoritmos de aprendizaje que construyen el árbol a partir de una muestra de entrenamiento etiquetada, así como la conversión del árbol en un conjunto de reglas sencillas y fáciles de entender. Otra posibilidad es aprender una base de reglas directamente.

9.1 Introducción

En la estimación paramétrica, definimos un modelo sobre todo el espacio de entrada y obtenemos sus parámetros de todos los datos de entrenamiento. Posteriormente, utilizamos el mismo modelo y el mismo conjunto de parámetros para cualquier entrada de prueba. En la estimación no paramétrica, dividimos el espacio de entrada en regiones locales, definidas por una medida de distancia como la norma euclidiana, y para cada entrada, utilizamos el modelo local correspondiente, calculado a partir de los datos de entrenamiento en esa región. En los modelos basados en instancias que analizamos en el capítulo 8, dada una entrada, identificar los datos locales que definen el modelo local es costoso; requiere calcular las distancias desde la entrada dada hasta todas las instancias de entrenamiento, que es O(N).

Árbol de decisión. Un árbol de decisión es un modelo jerárquico para el aprendizaje supervisado, donde la región local se identifica en una secuencia de divisiones recursivas en un número menor de pasos. Un árbol de decisión se compone de nodos de decisión internos, un nodo de decisión y hojas terminales (véase la figura 9.1). Cada nodo de decisión m implementa una función de prueba fm(x) con resultados discretos que etiquetan las ramas. Dada una entrada, en cada nodo se aplica una prueba y se toma una de las ramas según el resultado. Este proceso comienza en la raíz y se repite.

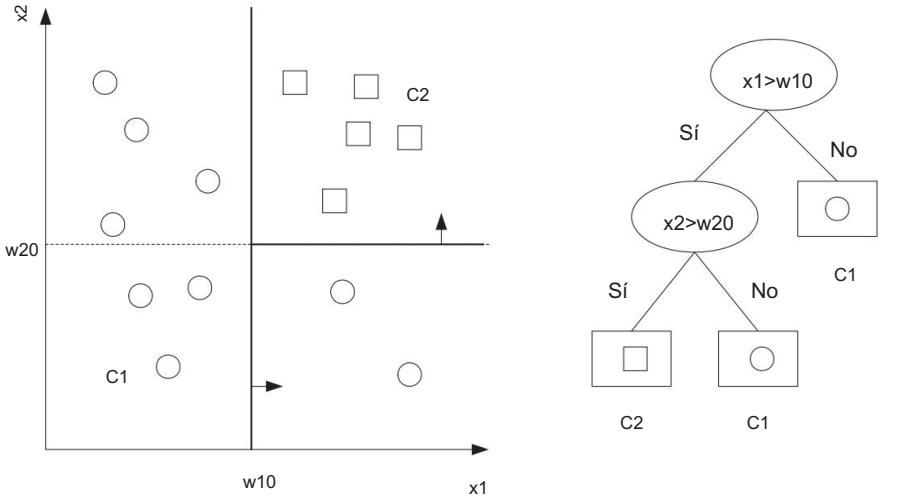

Figura 9.1 Ejemplo de un conjunto de datos y su árbol de decisión correspondiente. Los nodos ovalados son los nodos de decisión y los rectángulos los nodos hoja. El nodo de decisión univariante se divide a lo largo de un eje, y las divisiones sucesivas son ortogonales entre sí. Tras la primera división, {x|x1 < w10} es puro y no se divide más.

nodo hoja recursivamente hasta que se llega a un nodo hoja , momento en el cual el valor escrito en el La hoja constituye la salida.

Un árbol de decisión también es un modelo no paramétrico en el sentido de que no asumimos ninguna forma paramétrica para las densidades de clases y la estructura del árbol no es fija a priori sino que el árbol crece, se agregan ramas y hojas, durante el aprendizaje dependiendo de la complejidad del problema inherente a los datos.

Cada fm(x) define un discriminante en el espacio de entrada de dimensión d, dividiéndolo en regiones más pequeñas que se subdividen a medida que se traza una ruta desde la raíz hacia abajo. fm(·) es una función simple y, al escribirse como un árbol, una función compleja se descompone en una serie de decisiones simples. Los diferentes métodos de árboles de decisión asumen diferentes modelos para fm(·), y la clase del modelo define la forma del discriminante y la forma de las regiones. Cada nodo hoja tiene una etiqueta de salida, que en el caso de la clasificación es el código de clase y en la regresión es un valor numérico. Un nodo hoja define una región localizada en el espacio de entrada donde las instancias que caen en esta región tienen las mismas etiquetas (en la clasificación).

o resultados numéricos muy similares (en regresión). Los límites de las regiones se definen mediante los discriminantes codificados en los nodos internos en la ruta desde el nodo raíz hasta el nodo hoja.

La colocación jerárquica de las decisiones permite una rápida localización de la región que cubre una entrada. Por ejemplo, si las decisiones son binarias, en el mejor de los casos, cada decisión elimina la mitad de los casos. Si hay b regiones, en el mejor de los casos, la región correcta se puede encontrar en log2 b decisiones. Otra ventaja del árbol de decisión es su interpretabilidad. Como veremos en breve, el árbol puede convertirse en un conjunto de reglas SI-ENTONCES fácilmente comprensibles. Por esta razón, los árboles de decisión son muy populares y, en ocasiones, se prefieren a métodos más precisos, pero menos interpretables.

Comenzamos con árboles univariados donde la prueba en un nodo de decisión utiliza Con una sola variable de entrada, vemos cómo se pueden construir estos árboles para clasificación y regresión. Posteriormente, generalizamos esto a árboles multivariados donde todas las entradas se pueden usar en un nodo interno.

9.2 Árboles univariados

Árbol univariante. En un árbol univariante, en cada nodo interno, la prueba utiliza solo una de las dimensiones de entrada. Si la dimensión de entrada utilizada, xj , es discreta y toma uno de los n valores posibles, el nodo de decisión verifica el valor de xj y toma la rama correspondiente, implementando una división de n vías. Por ejemplo, si un atributo es color {rojo, azul, verde}, entonces un nodo en ese atributo tiene tres ramas, cada una correspondiente a uno de los tres valores posibles del atributo.

Un nodo de decisión tiene ramas discretas y debe haber una entrada numérica. Discretizado. Si xj es numérico (ordenado), la prueba es una comparación (9.1)

$$f_m(x) : x_j > w_{m0}$$

Donde wm0 es un valor umbral adecuadamente seleccionado. El nodo de decisión divide el espacio de entrada en dos: $L_m = \{x|x_j > w_{m0}\}$ y $R_m = \{x|x_j \leq$ división binaria $w_{m0}\}$; esto se denomina división binaria. Los nodos de decisión sucesivos en una ruta desde la raíz hasta una hoja dividen estas en dos utilizando otros atributos y generando divisiones ortogonales entre sí. Los nodos hoja definen hiperrectángulos en el espacio de entrada (véase la figura 9.1).

La inducción de árboles es la construcción del árbol dada una muestra de entrenamiento. Para un conjunto de entrenamiento dado, existen muchos árboles que lo codifican sin errores y, para simplificar, nos interesa encontrar el más pequeño entre

ellos, donde el tamaño del árbol se mide como el número de nodos en el árbol y la complejidad de los nodos de decisión. Encontrar el árbol más pequeño es NP-completo (Quinlan 1986), y nos vemos obligados a utilizar procedimientos de búsqueda local basados en heurísticas que dan árboles razonables en un tiempo razonable.

Los algoritmos de aprendizaje de árboles son codiciosos y, en cada paso, comienzan por el raíz con los datos de entrenamiento completos, buscamos la mejor división. Esto divide los datos de entrenamiento en dos o n, dependiendo de si el elegido El atributo es numérico o discreto. Luego, continuamos dividiendo recursivamente. con el subconjunto correspondiente hasta que ya no necesitemos dividir más, en punto en el que se crea y etiqueta un nodo hoja.

9.2.1 Árboles de clasificación

árbol de clasificación En el caso de un árbol de decisión para la clasificación, es decir, un árbol de clasificación

Árbol de medición de impurezas : la bondad de una división se cuantifica mediante una medida de impurezas.

La división es pura si después de la división, para todas las ramas, todas las instancias elegidas Una rama pertenece a la misma clase. Digamos que para el nodo m, Nm es el Número de instancias de entrenamiento que llegan al nodo m. Para el nodo raíz, es N. N_{i_m} de Nm pertenecen a la clase Ci, con $\sum_i N_{i_m} = N_m$. Dado que una instancia llega al nodo m, la estimación de la probabilidad de la clase Ci es

$$\hat{P}(C_i|x,m) \equiv p_{i_m} = \frac{N_{i_m}}{N_m} \tag{9.2}$$

El nodo m es puro si p_{i_m} para todos los i son 0 o 1. Es 0 cuando ninguno de los Las instancias que llegan al nodo m son de clase Ci, y es 1 si todas esas instancias son de Ci. Si la división es pura, no necesitamos dividir más y podemos Agregue un nodo de hoja etiquetado con la clase para la cual p_{i_m} es 1. Una posible

La función de entropía para medir la impureza es la entropía (Quinlan 1986) (ver figura 9.2)

$$I_m = -\sum_{i=1}^{K} p_{i_m} \log_2 p_{i_m} \tag{9.3}$$

donde 0 log 0 ≡ 0. La entropía en la teoría de la información especifica el mínimo Número de bits necesarios para codificar el código de clase de una instancia. En un sistema de dos Problema de clase, si p1 = 1 y p2 = 0, todos los ejemplos son de C1, y hacemos no es necesario enviar nada y la entropía es 0. Si p1 = p2 = 0,5, Es necesario enviar un bit para señalar uno de los dos casos y la entropía es 1. Entre estos dos extremos, podemos idear códigos y utilizar menos de un poco por mensaje al tener códigos más cortos para la clase más probable y

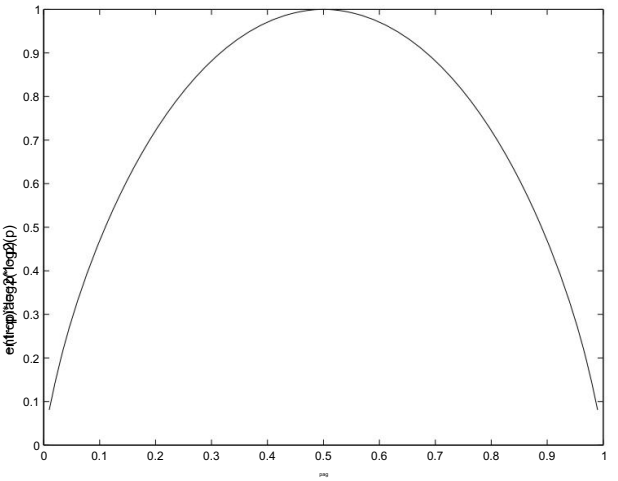

Figura 9.2 Función de entropía para un problema de dos clases.

Códigos más largos para los menos probables. Cuando hay K > 2 clases, lo mismo La discusión se sostiene y la entropía más grande es log2 K cuando pi = 1/K.

Pero la entropía no es la única medida posible. Para un problema de dos clases donde p1 ≡ p y p2 = 1 − p, φ (p, 1 − p) es una función no negativa medir la impureza de una división si satisface las siguientes propiedades (Devroye, Györfi y Lugosi 1996):

φ (1/2, 1/2) ≥ φ (p, 1 − p), para cualquier p [0, 1].

φ (0, 1) = φ (1, 0) = 0.

φ (p, 1−p) aumenta en p en [0, 1/2] y disminuye en p en [1/2, 1].

Algunos ejemplos son:

1. Entropía

(9.4) φ (p, 1 − p) = −p log2 p − (1 − p) log2 (1 − p)

La ecuación 9.3 es la generalización a K > 2 clases.

Índice de Gini 2. Índice de Gini (Breiman et al. 1984)

(9.5) φ (p, 1 − p) = 2p(1 − p)

3. Error de clasificación errónea

$$\varphi(p, 1-p) = 1 - \max(p, 1-p) \tag{9.6}$$

Estos pueden generalizarse a K > 2 clases, y el error de clasificación errónea puede generalizarse al riesgo mínimo dada una función de pérdida (ejercicio 1).
Las investigaciones han demostrado que no existe una diferencia significativa entre estas tres medidas.

Si el nodo m no es puro, entonces las instancias deben dividirse para disminuir impureza, y hay múltiples atributos posibles en los que podemos dividirnos.
Para un atributo numérico, son posibles múltiples posiciones de división. Entre todas, Buscamos la división que minimice la impureza después de la división porque
Se desea generar el árbol más pequeño. Si los subconjuntos después de la división están más cerca... Para ser puro, se necesitarán menos divisiones (si es que hay alguna) después. Por supuesto, esto es localmente óptimos y no tenemos garantía de encontrar la decisión más pequeña árbol.

Digamos que en el nodo m, Nmj de Nm toman la rama j; estos son xt para los cuales La prueba fm(xt) devuelve el resultado j. Para un atributo discreto con n valores, Hay n resultados y, para un atributo numérico, hay dos resultados. (n = 2), en cualquier caso satisfaciendo $\sum_{j=1}^{n} N_{mj} = N_m$. N_{mj} de Nmj pertenecen a clase Ci: $\sum_{i=1}^{K} N_{mj} = N_{mj}$. De manera similar, $\sum_{j=1}^{n} N_{mj} = N_{metro.}$

Entonces, dado que en el nodo m, la prueba devuelve el resultado j, la estimación para La probabilidad de la clase Ci es

$$\hat{P}(C_i|x, m, j) \equiv p_{mj} = \frac{N_{mj}}{N_{mj}} \tag{9.7}$$

y la impureza total después de la división se da como

$$I_{metro} = -\sum_{j=1}^{n} \frac{N_{mj}}{N_{Nuevo\ Méjico}} \sum_{i=1}^{K} p_{mj} \log_2 p_{mj} \tag{9.8}$$

En el caso de un atributo numérico, para poder calcular p_{mj} usando Ecuación 9.1, también necesitamos conocer wm0 para ese nodo. Hay Nm − 1 posible wm0 entre Nm puntos de datos: no necesitamos probar todos (posiblemente infinitos) puntos; basta con probar, por ejemplo, a mitad de camino entre puntos. Tenga en cuenta también que la mejor división siempre es entre puntos adyacentes. puntos pertenecientes a diferentes clases. Así que los probamos, y lo mejor en En términos de pureza se toma como base la pureza del atributo. En el caso de un atributo discreto, no es necesaria dicha iteración.

```
GenerarArbol(X)
    Si NodeEntropy(X)< θ I /* ecuación 9.3 */
        Crear hoja etiquetada por clase mayoritaria en X
        Devolver
    i ←AtributoDividido(X)
    Para cada rama de xi
        Encuentra a Xi cayendo en la rama
        GenerarArbol(Xi)

Atributo dividido (X)
    MinEnt←MAX Para
    todos los atributos i = 1,...,d
        Si xi es discreto con n valores
            Dividir X en X1,..., Xn por xi e ←
            SplitEntropy(X1,..., Xn) /* ecuación 9.8 */ Si e<MinEnt MinEnt ←
            e; bestf ←i
        De lo contrario /* xi es numérico */
            Para todas las posibles divisiones
                Dividir X en X1, X2 en xi
                e←SplitEntropy(X1, X2)
                Si e<MinEnt MinEnt ←e; mejorf ←yo
    Devolver bestf
```

Figura 9.3 Construcción del árbol de clasificación.

Entonces, para todos los atributos, discretos y numéricos, y para un atributo numérico para todas las posiciones divididas, calculamos la impureza y elegimos la que tiene la entropía mínima, por ejemplo, según lo medido por la ecuación 9.8.

Luego, la construcción del árbol continúa recursivamente y en paralelo para todas las ramas de clasificación y regresión no puras, hasta que todas lo sean. Esta es la base del algoritmo de Árboles de Clasificación y Regresión (CART) (Breiman et al., 1984), el algoritmo ID3 (Quinlan, 1986) y su extensión C4.5 (Quinlan, 1993). El pseudocódigo C4.5 del algoritmo se muestra en la figura 9.3.

También se puede decir que en cada paso durante la construcción del árbol, elegimos la división que provoca la mayor disminución de impureza, que es la diferencia entre la impureza de los datos que llegan al nodo m (ecuación 9.3) y la entropía total de los datos que llegan a sus ramas después de la división (ecuación 9.8).

Un problema es que dicha división favorece a los atributos con muchos valores.
Cuando hay muchos valores, hay muchas ramas y la impureza
puede ser mucho menor. Por ejemplo, si tomamos el índice de entrenamiento t como atributo,
la medida de impureza elegirá eso porque entonces la impureza de cada uno
rama es 0, aunque no es una característica razonable. Nodos con muchos
Las ramas son complejas y contradicen nuestra idea de dividir los discriminantes de clase en
decisiones simples. Se han propuesto métodos para penalizar
tales atributos y equilibrar la caída de impurezas y el factor de ramificación.
colina.

Cuando hay ruido, haciendo crecer el árbol hasta que esté más puro, podemos crecer
Un árbol muy grande presenta sobreajuste; por ejemplo, considere el caso de una instancia mal
etiquetada en un grupo de instancias correctamente etiquetadas. Para mitigar dicho sobreajuste,
la construcción del árbol finaliza cuando los nodos se vuelven puros.
suficiente, es decir, un subconjunto de datos no se divide más si $I < \theta_I$. Esto implica que no
requerimos que p_{mj} ser exactamente 0 o 1 pero lo suficientemente cerca,
con un umbral θ_p. En tal caso, se crea un nodo hoja y se etiqueta.
con la clase que tiene el p_{mj} más alto.

θ_I (o θ_p) es el parámetro de complejidad, como h o k de los no paramétricos.
Estimación. Cuando son pequeños, la varianza es alta y el árbol crece.
grandes para reflejar con precisión el conjunto de entrenamiento y, cuando son grandes, la
varianza es menor y un árbol más pequeño representa aproximadamente el conjunto de entrenamiento y
Puede tener un sesgo importante. El valor ideal depende del coste de la clasificación errónea, así
como de los costes de memoria y computación.

En general, se recomienda que en una hoja se almacenen las probabilidades posteriores de
las clases, en lugar de etiquetar la hoja con la clase que tiene la probabilidad.
posterior más alta. Estas probabilidades pueden ser necesarias en pasos posteriores.
Por ejemplo, al calcular riesgos. Tenga en cuenta que no necesitamos almacenar los
instancias que llegan al nodo o los recuentos exactos; basta con las proporciones.

9.2.2 Árboles de regresión (no entra)

árbol de regresión Un árbol de regresión se construye casi de la misma manera que un árbol de clasificación, excepto
que la medida de impureza que es apropiada para
La clasificación se reemplaza por una medida apropiada para la regresión. Veamos
digamos para el nodo m, X_m es el subconjunto de X que llega al nodo m; es decir, es el
conjunto de todos los $x \in X$ que satisfacen todas las condiciones en los nodos de decisión en el
ruta desde la raíz hasta el nodo m. Definimos

$$b_m(x) = \begin{cases} 1 \text{ si } x \in X_m: x \text{ llega al nodo } m \\ 0 \text{ en caso contrario} \end{cases} \quad (9.9)$$

En regresión, la bondad de una división se mide por el cuadrado medio.
Error del valor estimado. Digamos que gm es el valor estimado en
nodo m.

$$E_m = \frac{1}{N_m} \sum_{t} (r^t - g_m)^2 b_m(x^t) \tag{9.10}$$

donde $N_m = |X_m| = \sum_t b_m(x^t)$.

En un nodo, utilizamos la media (mediana si hay demasiado ruido) de la
Salidas requeridas de las instancias que llegan al nodo

$$g_m = \frac{\sum_t b_m(x^t) r^t}{\sum_t b_m(x^t)} \tag{9.11}$$

Entonces la ecuación 9.10 corresponde a la varianza en m. Si en un nodo, la
El error es aceptable, es decir, $E_m < \theta_r$ entonces se crea un nodo hoja y
almacena el valor g_m. Al igual que el regresograma del capítulo 8, esto crea
una aproximación constante por partes con discontinuidades en los límites de las hojas.

Si el error no es aceptable, los datos que llegan al nodo m se dividen aún más.
de modo que la suma de los errores en las ramas sea mínima. Al igual que en la clasificación,
en cada nodo buscamos el atributo (y el umbral de división)
para un atributo numérico) que minimiza el error y luego continuamos
recursivamente.

Definamos X_{mj} como el subconjunto de X_m que toma la rama j: $\bigcup_{j=1}^{n} X_{mj} = X_m$.
Nosotros definimos

$$b_{mj}(x) = \begin{cases} 1 \text{ si } x \in X_{mj} : x \text{ llega al nodo m y toma la rama j} \\ 0 \text{ en caso contrario} \end{cases} \tag{9.12}$$

g_{mj} es el valor estimado en la rama j del nodo m.

$$g_{mj} = \frac{\sum_t b_{mj}(x^t) r^t}{\sum_t b_{mj}(x^t)} \tag{9.13}$$

y el error después de la división es

$$E'_m = \frac{1}{N_m} \sum_{j} \sum_{t} (r^t - g_{mj})^2 b_{mj}(x^t) \tag{9.14}$$

La reducción del error para cualquier división se da como la diferencia entre la ecuación
9.10 y la ecuación 9.14. Buscamos la división tal que esta reducción sea
máximo o, equivalentemente, donde la ecuación 9.14 toma su mínimo. El
El código dado en la figura 9.3 se puede adaptar para entrenar un árbol de regresión mediante

Reemplazar los cálculos de entropía con el error cuadrático medio y las etiquetas de clase con promedios.

El error cuadrático medio es una posible función de error; otra es el peor error posible.

$$E_m = \max_j \max_t |r^t - g_{mj}| b_m(x^t) \tag{9.15}$$

Y usando esto, podemos garantizar que el error para cualquier instancia nunca sea... mayor que un umbral determinado.

El umbral de error aceptable es el parámetro de complejidad; cuando es pequeños, generamos árboles grandes y corremos el riesgo de sobreajuste; cuando es grande, desajustarse y suavizarse demasiado (ver figuras 9.4 y 9.5).

Similar a pasar de la media móvil a la línea móvil en no paramétrico regresión, en lugar de tomar un promedio en una hoja que implementa un ajuste constante, también podemos hacer un ajuste de regresión lineal sobre las instancias que eligen la hoja:

$$g_m(x) = w_m^T x + w_{m0} \tag{9.16}$$

Esto hace que la estimación en una hoja dependa de x y genere valores más pequeños. árboles, pero existe el gasto de computación adicional en un nodo hoja.

9.3 Poda

Con frecuencia, un nodo no se divide más si el número de instancias de entrenamiento llegar a un nodo es menor que un cierto porcentaje del conjunto de entrenamiento— Por ejemplo, el 5 por ciento, independientemente de la impureza o el error. La idea es que cualquier decisión basada en muy pocos casos causa variación y, por lo tanto Error de generalización. Detener la construcción del árbol antes de que esté completo. La prepoda se denomina prepoda del árbol.

pospoda Otra posibilidad para obtener árboles más simples es la poda posterior, que en la práctica funciona mejor que la poda previa. Vimos antes que el crecimiento de los árboles es... codiciosos y en cada paso tomamos una decisión, es decir, generamos una decisión nodo y continuar más adelante, sin retroceder ni probar una alternativa. La única excepción es la pospoda, donde intentamos encontrar y podar subárboles innecesarios.

En la poda posterior, hacemos crecer el árbol hasta que todas las hojas estén puras y No tenemos ningún error de entrenamiento. Luego encontramos subárboles que causan sobreajuste y Los podamos . Del conjunto etiquetado inicial, reservamos un conjunto de poda. No se usó durante el entrenamiento. Para cada subárbol, lo reemplazamos con un nodo hoja.

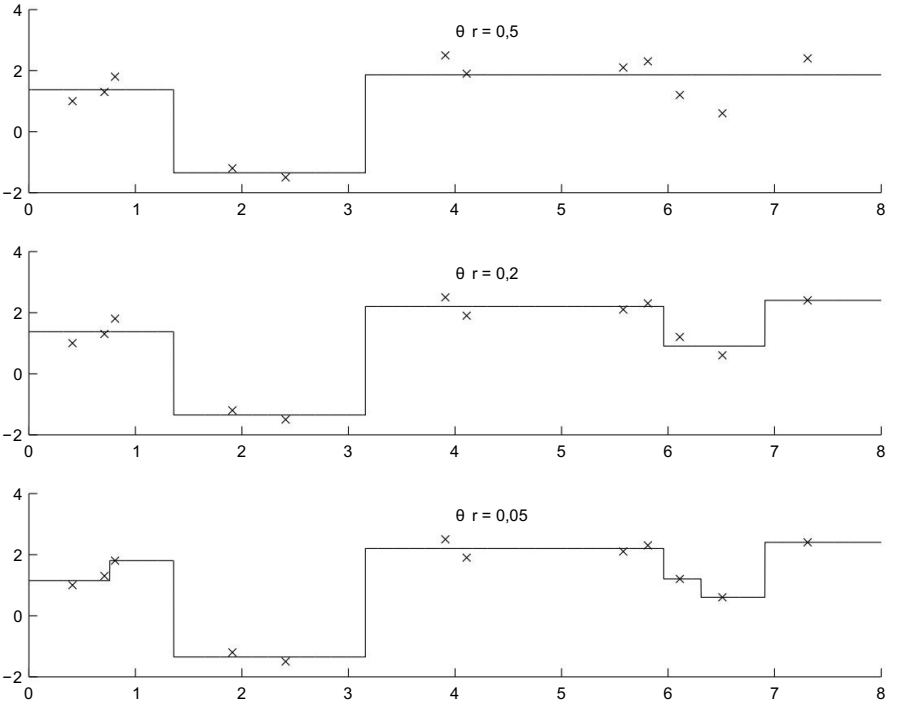

Figura 9.4 Suavizaciones del árbol de regresión para varios valores de θ r . Los correspondientes Los árboles se muestran en la figura 9.5.

etiquetado con las instancias de entrenamiento cubiertas por el subárbol (apropiadamente) para clasificación o regresión). Si el nodo hoja no tiene un peor rendimiento que el subárbol en el conjunto de poda, podamos el subárbol y mantenemos el nodo hoja porque la complejidad adicional del subárbol no está justificada; De lo contrario, mantenemos el subárbol.

Por ejemplo, en el tercer árbol de la figura 9.5, hay un subárbol que comienza con condición x < 6.31. Este subárbol puede ser reemplazado por un nodo hoja de y = 0,9 (como en el segundo árbol) si el error en el conjunto de poda no aumentar durante la sustitución. Tenga en cuenta que el conjunto de poda no debe ser se confunde con (y es distinto de) el conjunto de validación.

Comparando la prepoda y la pospoda, podemos decir que la prepoda es más rápido, pero la poda posterior generalmente conduce a árboles más precisos.

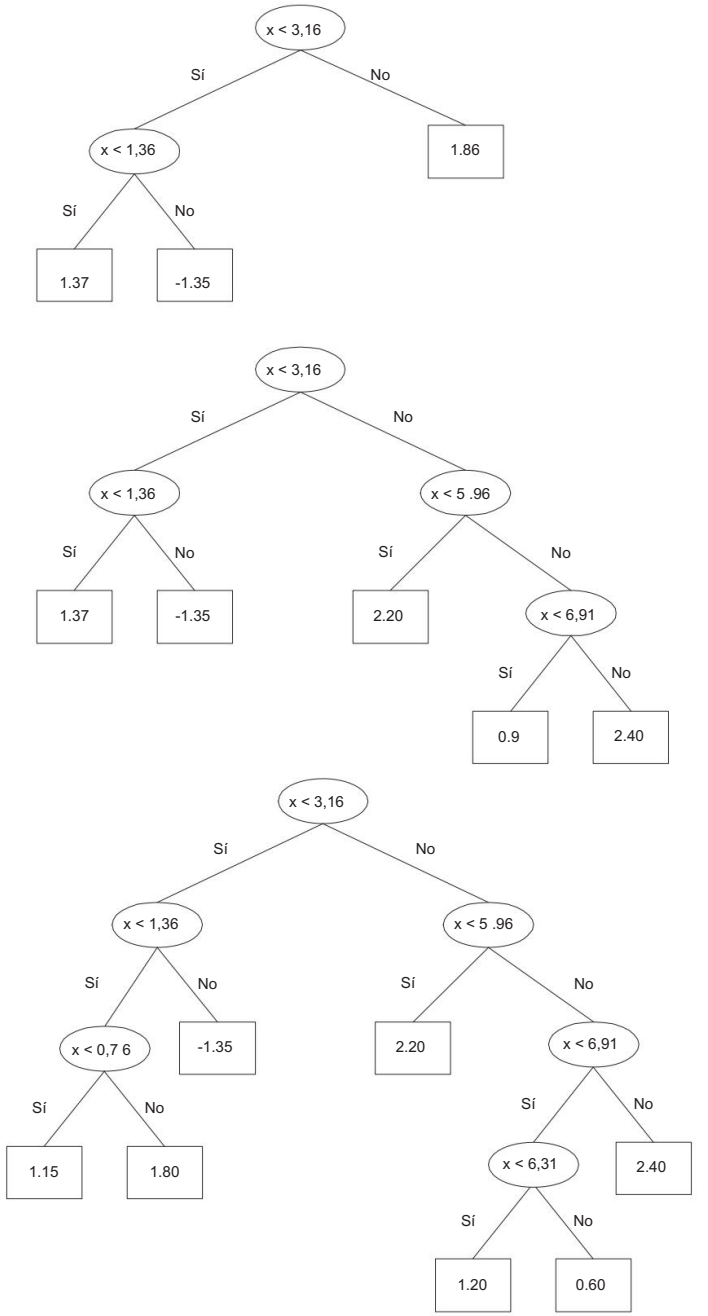

Figura 9.5 Árboles de regresión que implementan los suavizados de la figura 9.4 para varios valores de θ r .

Aprendiendo con árboles

Ahora consideraremos un enfoque bastante diferente para el aprendizaje automático, comenzando con una de las estructuras de datos más comunes y potentes de la informática: el árbol binario. El coste computacional de crear el árbol es bastante bajo, pero el coste de usarlo es aún menor: O(log N), donde N es el número de puntos de datos. Esto es importante para el aprendizaje automático, ya que las consultas al algoritmo entrenado deben ser lo más rápidas posible, ya que ocurren con mayor frecuencia y el resultado suele solicitarse de inmediato. Esto es suficiente para que los árboles resulten atractivos para el aprendizaje automático. Sin embargo, presentan otras ventajas, como su facilidad de comprensión (seguir un árbol para obtener una respuesta de clasificación es transparente, lo que aumenta la confianza en él que obtener una respuesta de una red neuronal de "caja negra").

Por estas razones, la clasificación mediante árboles de decisión ha ganado popularidad en los últimos años. Es muy probable que hayas tenido que lidiar con árboles de decisión si alguna vez has llamado a una línea de ayuda, por ejemplo, por problemas informáticos. Los operadores telefónicos se guían por el árbol de decisión según tus respuestas a sus preguntas.

La idea de un árbol de decisión es descomponer la clasificación en un conjunto de opciones sobre cada característica, comenzando por la raíz (base) del árbol y descendiendo hasta las hojas, donde se recibe la decisión de clasificación. Los árboles son muy fáciles de entender e incluso pueden convertirse en un conjunto de reglas condicionales, adecuadas para un sistema de inducción de reglas.

En términos de optimización y búsqueda, los árboles de decisión utilizan una heurística voraz para realizar la búsqueda, evaluando las posibles opciones en la etapa actual de aprendizaje y seleccionando la que parezca óptima en ese momento. Esto funciona bien en la mayor parte del tiempo.

12.1 USO DE ÁRBOLES DE DECISIÓN

Como estudiante, puede ser difícil decidir qué hacer por la noche. Hay cuatro cosas que realmente disfrutas o tienes que hacer: ir al bar, ver la televisión, ir a una fiesta o incluso (¡qué sorpresa!) estudiar. A veces, la decisión ya está tomada: si tienes una tarea para el día siguiente, necesitas estudiar; si tienes pereza, el bar no es para ti; y si no hay fiesta, no puedes ir. Buscas un algoritmo que te permita decidir qué hacer cada noche sin tener que pensarlo todas las noches. La Figura 12.1 muestra precisamente ese algoritmo.

Cada noche empiezas desde la raíz del árbol y compruebas si alguno de tus amigos sabe de una fiesta esa noche. Si hay una, tienes que ir, sin duda. Solo si no hay fiesta te preocupas por si tienes una entrega de tarea próxima. Si hay una entrega crucial, tienes que estudiar, pero si no hay nada urgente para los próximos días, piensas en cómo te sientes. Un repentino estallido de energía.

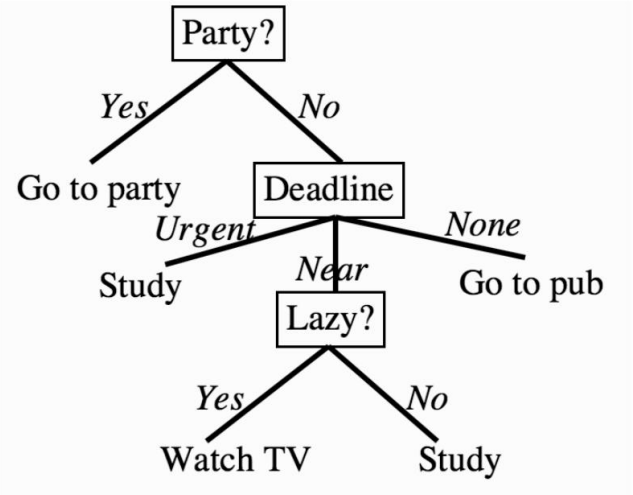

FIGURA 12.1 Un árbol de decisiones simple para decidir cómo pasará la noche.

Puede que te haga estudiar, pero de lo contrario estarás desplomado frente al televisor disfrutando de tu amor secreto por Shortland Street (u otra telenovela de tu elección) en lugar de estudiar.
Por supuesto, cerca del comienzo del semestre, cuando no hay tareas que hacer y te sientes rico, estarás en el pub.

Una de las razones por las que los árboles de decisión son populares es que podemos convertirlos en un conjunto de disyunciones lógicas (reglas si... entonces) que luego van al código del programa de manera muy simple: la primera parte del árbol anterior se puede convertir en: • si hay una fiesta , entonces vaya a ella • si no hay una

fiesta y tiene una fecha límite urgente , entonces
estudie • etc.

Eso es todo lo que hay que saber sobre el uso del árbol de decisión. Compárelo con el uso anterior de estos datos, con el clasificador Naïve Bayes de la sección 2.3.2. Lo más interesante es cómo construir el árbol a partir de los datos, y ese es el tema central de la siguiente sección.

12.2 CONSTRUCCIÓN DE ÁRBOLES DE DECISIÓN

En el ejemplo anterior, las tres características que necesitamos para el algoritmo son tu nivel de energía, la fecha límite más cercana y si hay fiesta esta noche. La pregunta que debemos plantearnos es cómo, basándonos en esas características, podemos construir el árbol. Existen diferentes algoritmos de árboles de decisión, pero casi todos son variantes del mismo principio: construyen el árbol de forma voraz, comenzando desde la raíz, eligiendo la característica más informativa en cada paso. Empezaremos centrándonos en el más común: el ID3 de Quinlan, aunque también mencionaremos su extensión, conocida como C4.5, y otra conocida como CART.

Había una palabra importante oculta en la oración anterior sobre cómo funcionan los árboles, lo cual fue informativo. Elegir qué característica usar a continuación en el árbol de decisión puede considerarse como jugar al juego de las "20 preguntas", donde intentas obtener el objeto en el que tu oponente está pensando haciéndole preguntas sobre él. En cada etapa, eliges la pregunta que te brinde la mayor información posible considerando lo que ya sabes. Por lo tanto, preguntarías "¿Es un animal?" antes de preguntar "¿Es un gato?". La idea es cuantificar esta pregunta de cuánto...

La información se obtiene al conocer ciertos hechos. Codificarla matemáticamente es la tarea de la teoría de la información.

12.2.1 Un breve aparte: La entropía en la teoría de la información

La teoría de la información nació en 1948 cuando Claude Shannon publicó un artículo titulado "Una teoría matemática de la comunicación". En dicho artículo, propuso la medida de la entropía de la información, que describe la cantidad de impureza en un conjunto de características. La entropía H de un conjunto de probabilidades pi es (para quienes tengan conocimientos de física, la relación con la entropía física debería ser clara):

$$\text{Entropía}(p) = -\sum_i pi \log_2 pi\,, \tag{12.1}$$

donde el logaritmo es de base 2 porque imaginamos que codificamos todo usando dígitos binarios (bits), y definimos 0 log 0 = 0. Un gráfico de la entropía se da en la Figura 12.2.
Supongamos que tenemos un conjunto de ejemplos positivos y negativos de una característica (donde la característica solo puede tomar dos valores: positivo y negativo). Si todos los ejemplos son positivos, no obtenemos información adicional al conocer el valor de la característica para ningún ejemplo en particular, ya que, independientemente de su valor, el ejemplo será positivo.
Por lo tanto, la entropía de esa característica es 0. Sin embargo, si la característica separa los ejemplos en 50% positivos y 50% negativos, entonces la entropía es máxima, y conocer dicha característica nos resulta muy útil. El concepto básico es que nos indica cuánta información adicional obtendríamos al conocer el valor de esa característica. Una función para calcular la entropía es muy simple, como la siguiente:

```
def calc_entropía(p):

    si p!=0:
        devuelve -p de * np.log2(p)
    lo contrario:
        devolver 0
```

Para nuestro árbol de decisión, la mejor característica para clasificar ahora es la que proporciona más información, es decir, la que tiene la mayor entropía. Después de usarla, reevaluamos la entropía de cada característica y seleccionamos de nuevo la que tenga la mayor entropía.

La teoría de la información es un tema muy interesante. Es posible descargar el artículo de Shannon de 1948 de internet y también encontrar numerosos recursos que muestran dónde se ha aplicado. Actualmente existen revistas completas dedicadas a la teoría de la información debido a su relevancia en diversas áreas, como las redes informáticas y de telecomunicaciones, el aprendizaje automático y el almacenamiento de datos. Al final del capítulo se ofrecen lecturas adicionales sobre el tema.

12.2.2 ID3

Ahora que tenemos una medida adecuada para elegir la característica a continuación, la entropía, solo tenemos que determinar cómo aplicarla. La idea clave es determinar cuánto disminuiría la entropía de todo el conjunto de entrenamiento si elegimos cada característica en particular para el siguiente paso de clasificación. Esto se conoce como ganancia de información y se define como

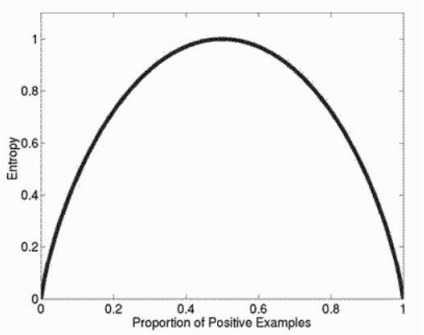

FIGURA 12.2 Un gráfico de entropía, que detalla cuánta información está disponible al encontrar otra pieza de información dado lo que ya sabe.

La entropía del conjunto total menos la entropía al elegir una característica particular. Esto se define por (donde S es el conjunto de ejemplos, F es una característica posible del conjunto de todas las posibles, y |Sf | es el recuento del número de miembros de S que tienen el valor f para la característica F):

$$\text{Ganancia}(S, F) = \text{Entropía}(S) - \sum_{f \in \text{valores}(F)} \frac{|Sf|}{|S|} \text{Entropía}(Sf). \tag{12.2}$$

Por ejemplo, supongamos que tenemos datos (con resultados) S = {s1 = verdadero, s2 = falso, s3 = falso, s4 = falso} y una característica F que puede tener valores {f1, f2, f3}. En el ejemplo, el valor de la característica para s1 podría ser f2, para s2 podría ser f2, para s3, f3 y para s4, f1 . Entonces, podemos calcular la entropía de S como (donde significa verdadero, de lo cual tenemos un ejemplo, y significa falso, de lo cual tenemos tres ejemplos):

$$\text{Entropía}(S) = -p \log_2 p - p \log_2 p = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0{,}5 + 0{,}311 = 0{,}811. \tag{12.3}$$

La función Entropía(Sf) es similar, pero sólo se calcula con el subconjunto de datos donde La característica F tiene valores f.

Si intentabas seguir estos cálculos en una calculadora, quizás te preguntabas cómo calcular $\log_2 p$. La respuesta es usar la identidad $\log_2 p = \ln p / \ln(2)$, donde ln es el logaritmo natural, que tu calculadora puede generar. NumPy tiene la función np.log²().

Ahora queremos calcular la ganancia de información de F, por lo que ahora necesitamos calcular cada |Sf | de los valores dentro de la suma en la ecuación (12.2), las $\overline{\text{Entropía}(S)}$ (en nuestro ejemplo, |S| características son 'Fecha límite', 'Fiesta' y 'Perezoso'):

$$\frac{|Sf1|}{|S|}\text{Entropía}(Sf1) = \frac{1}{4} \times -\left(\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1}\right) = 0 \quad (12.4)$$

$$\frac{|Sf2|}{|S|}\text{Entropía}(Sf2) = \frac{2}{4}\left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right) = \frac{1}{2} \quad (12.5)$$

$$\frac{|Sf3|}{|S|}\text{Entropía}(Sf3) = \frac{1}{4}\left(-\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1}\right) = 0 \quad (12.6)$$

La ganancia de información al agregar esta característica es la entropía de S menos la suma de las tres valores arriba:

$$\text{Ganancia}(S, F) = 0{,}811 - (0 + 0{,}5 + 0) = 0{,}311. \quad (12.7)$$

Esto se puede calcular mediante un algoritmo utilizando la siguiente función (donde gran parte del código es para obtener los datos relevantes):

```
def calc_info_gain(datos,clases,característica): ganancia = 0
    nData =
    len(datos)
    # Enumere los valores que puede tomar la
    característica
    values = [] for datapoint in data:
            if datapoint[feature] not in values:
                valores.append(punto de datos[característica])

    featureCounts = np.zeros(len(valores)) entropía =
    np.zeros(len(valores)) índiceDeValor = 0

    # Encuentre dónde aparecen esos valores en data[feature] y la clase correspondiente para el
    valor
    en values:
            índice de datos = 0
            newClasses = [] para
            el punto de datos en datos: si
                el punto de datos[característica]==valor:
                    featureCounts[valueIndex]+=1
                    newClasses.append(classes[dataIndex]) dataIndex += 1


            # Obtener los valores en newClasses
            classValues = [] para
            aclass en newClasses:
                si classValues.count(aclass)==0:
                    classValues.append(aclass)
```

```
classCounts = np.zeros(len(classValues))
índice de clase = 0
para classValue en classValues:
    para una clase en nuevas clases:
        si una clase == valorClase:
            classCounts[classIndex]+=1 classIndex
    += 1

para classIndex en rango(len(classValues)): entropía[valueIndex]
    += calc_entropy(float(classCounts[classIndex]) /sum(classCounts)) ganancia +=
    float(featureCounts[valueIndex])/
nData * entropía[valueIndex] valueIndex += 1 devuelve ganancia
```

El algoritmo ID3 calcula esta ganancia de información para cada característica y selecciona la que produce el mayor valor. En esencia, esto es todo lo que hace el algoritmo. Busca en el espacio de posibles árboles de forma voraz, seleccionando la característica con la mayor ganancia de información en cada etapa. El resultado del algoritmo es el árbol, es decir, una lista de nodos, aristas y hojas. Como cualquier árbol en informática, se puede construir recursivamente.
En cada etapa, se selecciona la mejor característica y se elimina del conjunto de datos, y el algoritmo se ejecuta recursivamente en el resto. La recursión se detiene cuando solo queda una clase en los datos (en cuyo caso se añade una hoja con esa clase como etiqueta), o cuando no quedan características, y se utiliza la etiqueta más común en los datos restantes.

El algoritmo ID3

- Si todos los ejemplos tienen la misma etiqueta:
  - devolver una hoja con esa etiqueta
- De lo contrario, no quedan funciones para probar:
  - devuelve una hoja con la etiqueta más común
- Demás:
  - elija la característica Fˆ que maximiza la ganancia de información de S para que sea la siguiente nodo usando la ecuación (12.2) –

  agregue una rama desde el nodo para cada valor posible f en Fˆ – para cada rama:

  calcular Sf eliminando Fˆ del conjunto de características llamar recursivamente al algoritmo con Sf , para calcular la ganancia relativa al conjunto actual de ejemplos

---

Debido al enfoque en la clasificación para ejemplos del mundo real, los árboles suelen usarse con características de texto en lugar de valores numéricos. Esto dificulta el uso de NumPy, por lo que la implementación de ejemplo es prácticamente Python puro. Utiliza una característica de Python poco común en otros lenguajes: el diccionario para almacenar el árbol, que utiliza las llaves {, }, y que se describe a continuación antes de analizar la implementación del árbol de decisión.

12.2.3 Implementación de árboles y gráficos en Python

Los árboles son en realidad una versión restringida de los grafos, ya que ambos constan de nodos y aristas entre ellos. Los grafos son una estructura de datos muy útil en diversas áreas de la informática. Hay dos maneras razonables de representar un grafo computacionalmente.
Una es una matriz N ×N , donde N es el número de nodos en la red. Cada elemento de la matriz es un 1 si existe un enlace entre dos nodos y un 0 en caso contrario. La ventaja de este enfoque es que es fácil asignar pesos a los enlaces cambiando los 1 por los valores de los pesos. La alternativa es almacenar una lista de nodos, seguida de cada uno por una lista de los nodos a los que está conectado. Ambas opciones son bastante comunes en Python; la segunda utiliza el diccionario, una estructura de datos básica que no hemos usado mucho, excepto de forma muy simple en el árbol de decisión (Capítulo 12), que consiste en un conjunto de claves y valores. En un grafo, la clave de cada entrada del diccionario es el nombre del nodo, y su valor es una lista de los nodos a los que está conectado, como en este ejemplo:

```
grafo = {'A': ['B', 'C'],'B': ['C', 'D'],'C': ['D'],'D': ['C'],'E': ['F'],
 'F': ['C']}
```

Eso es todo lo que se necesita para crear el diccionario, y su uso no es muy diferente, ya que existen métodos integrados para obtener una lista de claves (keys()) y comprobar si una clave está en un diccionario (in). El código para encontrar una ruta a través del grafo puede escribirse entonces como una función recursiva simple:

```
def findPath(graph, start, end, pathSoFar): pathSoFar =
    pathSoFar + [start] if start == end:

        devolver pathSoFar si el
    inicio no está en el gráfico:
        devolver None
    para el nodo en el gráfico[inicio]:
        Si el nodo no está en pathSoFar:
            newpath = findPath(grafo, nodo, fin, pathSoFar) return newpath return
            None
```

Usando estos métodos ahora podemos ver una implementación en Python del árbol de decisiones, que también tiene una llamada de función recursiva como base.

12.2.4 Implementación del árbol de decisiones La función

make_tree() (que utiliza las funciones calc_entropy() y calc_info_gain() que se describieron anteriormente) se ve así:

```
def make_tree(datos,clases,nombresDeCaracterísticas):
    #Se suprimieron varias inicializaciones
```

```
predeterminado = clases[np.argmax(frecuencia)] si nData==0 o
nFeatures == 0: # Se ha llegado a una rama vacía
    return default elif classes.count(classes[0]) ==
    nData: # Solo queda 1
clase return classes[0] de lo contrario: # Elija qué característica es
    mejor ganancia = np.zeros(nFeatures)
    para característica en

rango(nFeatures): g =
    calc_info_gain(data,classes,feature)
    ganancia[característica] = totalEntropy - g


    mejorCaracterística = np.argmax(ganancia)
    árbol = {featureNames[mejorCaracterística]:{}}
    # Encuentre los posibles valores de
    características para el valor en valores:
        # Encuentre los puntos de datos con cada valor de
        característica para el punto de
            datos en datos: si datapoint[bestFeature]==value:
                si bestFeature==0: punto de
                    datos = punto de datos[1:]
                    nuevosNombres = featureNames[1:]
                elif bestFeature==nFeatures: datapoint =
                    datapoint[:-1] newNames =
                    featureNames[:-1] else: datapoint =

                datapoint[:bestFeature] datapoint.extend(datapoint[bestFeature+1:])
                    newNames = featureNames[:bestFeature]
                    newNames.extend(featureNames[bestFeature+1:])
                    newData.append(datapoint) newClasses.append(classes[index]) index
                += 1 # Ahora recurre al siguiente nivel
                subtree = make_tree(newData,newClasses,newNames)



        # Y al regresar, agrega el subárbol al árbol
        tree[featureNames[bestFeature]][value] = subtree
    árbol de retorno
```

Vale la pena considerar cómo ID3 generaliza a partir de los ejemplos de entrenamiento al conjunto de todas las entradas posibles. Utiliza un método conocido como sesgo inductivo. La elección de la siguiente característica para añadir al árbol es la que ofrece la mayor ganancia de información, lo que sesga el algoritmo hacia árboles más pequeños, ya que intenta minimizar la cantidad de información restante. Esto es coherente con el principio bien conocido de que las soluciones cortas suelen ser mejores que las largas (no necesariamente cierto, pero las explicaciones más sencillas suelen ser más fáciles de recordar y comprender). Quizás haya oído hablar de este principio como la «navaja de Occam», aunque yo lo prefiero.

Lo denominó KISS (Keep It Simple, Stupid). De hecho, existe una forma sólida, basada en la teoría de la información, de expresar este principio. Se conoce como Longitud Mínima de Descripción (LMD) y fue propuesta por Rissanen en 1989. En esencia, afirma que la descripción más corta, es decir, la más concisa, es la mejor descripción.

Tenga en cuenta que el algoritmo puede gestionar el ruido en el conjunto de datos, ya que las etiquetas se asignan al valor más común del atributo objetivo. Otra ventaja de los árboles de decisión es que pueden gestionar los datos faltantes. Piense en lo que ocurriría si un ejemplo tuviera una característica faltante. En ese caso, podemos omitir ese nodo del árbol y continuar sin él, sumando todos los valores posibles que esa característica podría haber tomado. Esto es prácticamente imposible de hacer con las redes neuronales: ¿cómo se representan los datos faltantes cuando el cálculo se basa en si una neurona se activa o no? En el caso de las redes neuronales, es habitual descartar cualquier punto de datos que tenga datos faltantes o adivinar (de forma más técnica, imputar cualquier valor faltante, ya sea identificando puntos de datos similares y utilizando su valor o utilizando la media o la mediana de los valores de los datos para esa característica). Esto supone que los datos faltantes se distribuyen aleatoriamente dentro del conjunto de datos, no que faltan debido a un proceso desconocido.

Decir que ID3 tiende a favorecer árboles cortos es solo parcialmente cierto. El algoritmo utiliza todas las características que se le asignan, incluso si algunas no son necesarias. Esto, obviamente, conlleva el riesgo de sobreajuste, de hecho, lo hace muy probable. Hay algunas medidas que se pueden tomar para evitarlo; la más sencilla es limitar el tamaño del árbol. También se puede usar una variante de detención temprana mediante un conjunto de validación y comparando el rendimiento del árbol hasta el momento con él. Sin embargo, el enfoque que se utiliza en algoritmos más avanzados (en particular, C4.5, inventado por Quinlan para mejorar ID3) es la poda.

Existen varias versiones de poda, todas basadas en el cálculo del árbol completo y su reducción, evaluando el error en un conjunto de validación. La versión más simple ejecuta el algoritmo del árbol de decisión hasta que se utilizan todas las características, de modo que probablemente esté sobreajustado. Luego, genera árboles más pequeños recorriéndolos, seleccionando cada nodo por turno y reemplazando el subárbol debajo de cada nodo con una hoja etiquetada con la clasificación más común del subárbol. El error del árbol podado se evalúa en el conjunto de validación; este se conserva si el error es igual o menor que el del árbol original; en caso contrario, se rechaza.

C4.5 utiliza un método diferente llamado pospoda de reglas. Este consiste en tomar el árbol generado por ID3, convertirlo en un conjunto de reglas condicionales y, a continuación, podar cada regla eliminando las precondiciones si la precisión de la regla aumenta sin ellas. Las reglas se ordenan según su precisión en el conjunto de entrenamiento y se aplican en orden. Las ventajas de trabajar con reglas son que son más fáciles de leer y su orden en el árbol no importa, solo su precisión en la clasificación.

12.2.5 Manejo de variables continuas

Un aspecto que aún no hemos abordado es cómo manejar variables continuas; solo hemos considerado aquellas con conjuntos discretos de valores característicos. La solución más sencilla es discretizar la variable continua. Sin embargo, también es posible mantenerla continua y modificar el algoritmo. Para una variable continua, no existe un único punto donde dividirla: la variable puede dividirse entre cualquier par de puntos de datos, como se muestra en la Figura 12.3. Por supuesto, también puede dividirse en cualquiera de las infinitas ubicaciones a lo largo de la línea, pero no difieren de este conjunto más pequeño de ubicaciones. Incluso este conjunto más pequeño encarece el algoritmo para variables continuas que para discretas, ya que, además de calcular la ganancia de información de cada variable para elegir la mejor, debe calcularse la ganancia de información de muchos puntos dentro de cada variable. En general, solo se realiza una división para una variable

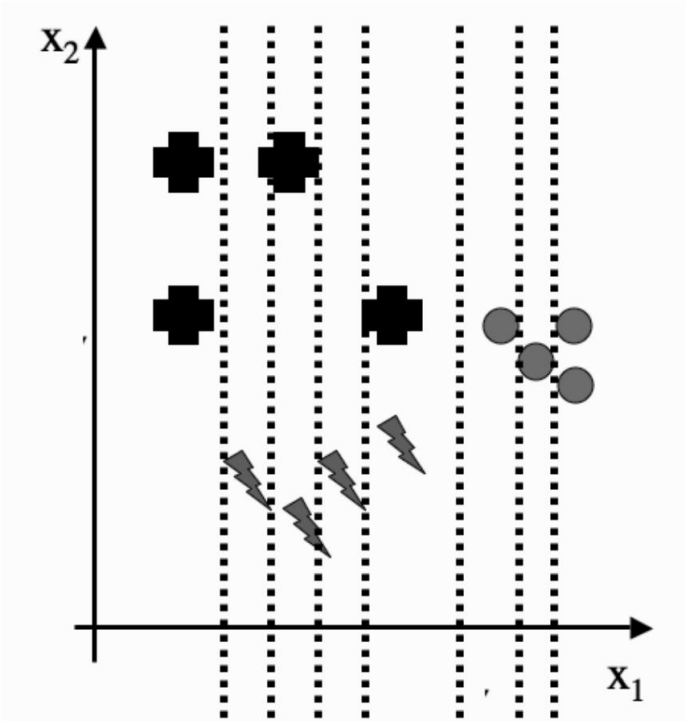

FIGURA 12.3 Posibles lugares para dividir la variable x1, entre cada uno de los puntos de datos a medida que aumenta el valor de la característica.

variable, en lugar de permitir divisiones en tres o más partes, aunque esto se puede hacer si es necesario.

Los árboles que estos algoritmos crean son todos árboles univariados, porque seleccionan una característica (dimensión) a la vez y la dividen según esa. También hay algoritmos que crean árboles multivariados seleccionando combinaciones de características. Esto puede generar árboles considerablemente más pequeños si es posible encontrar líneas rectas que separen bien los datos, pero que no sean paralelas a ningún eje. Sin embargo, los árboles univariados son más simples y tienden a obtener buenos resultados, por lo que no consideraremos más los árboles multivariados. Este hecho de que se seleccione una característica a la vez proporciona otra forma útil de visualizar lo que hace el árbol de decisión. La Figura 12.4 muestra la idea. Dado un conjunto de datos que contiene tres clases, el algoritmo selecciona una característica y un valor para esa característica para dividir los datos restantes en dos. El árbol final que resulta de esto se muestra en la Figura 12.5.

12.2.6 Complejidad computacional

El coste computacional de construir árboles binarios es bien conocido para el caso general, siendo O(N log N) para la construcción y O(log N) para la obtención de una hoja específica, donde N es el número de nodos. Sin embargo, estos resultados corresponden a árboles binarios balanceados, y los árboles de decisión a menudo no lo están. Si bien las medidas de información intentan mantener el árbol balanceado mediante la búsqueda de divisiones que separen los datos en dos partes iguales (ya que esto tendrá la mayor entropía), no hay garantía de que esto suceda. Tampoco son necesariamente binarios, especialmente para ID3 y C4.5, como muestra nuestro ejemplo.

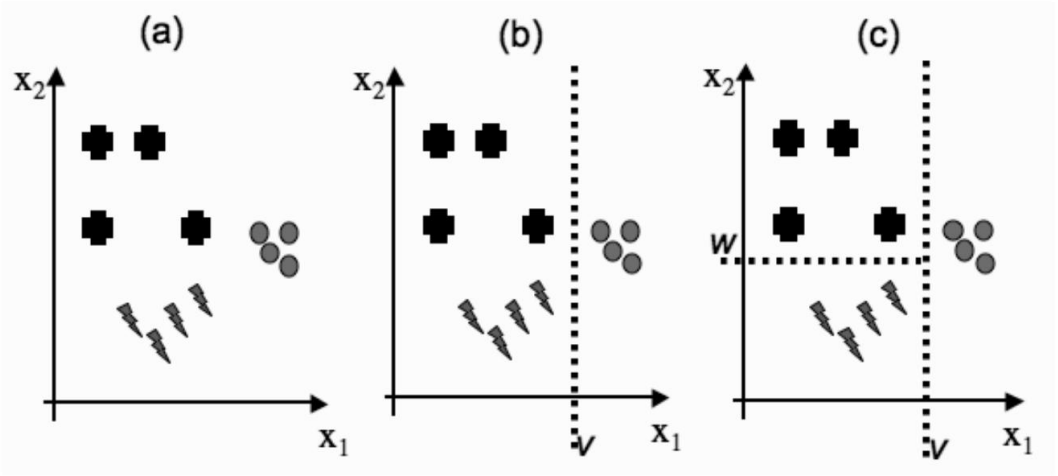

FIGURA 12.4 Efecto de las elecciones en el árbol de decisión. El conjunto de datos bidimensionales mostrado en (a) se divide primero seleccionando la característica x1 (b) y luego x2 (c), lo que separa las tres clases. El árbol final se muestra en la Figura 12.5.

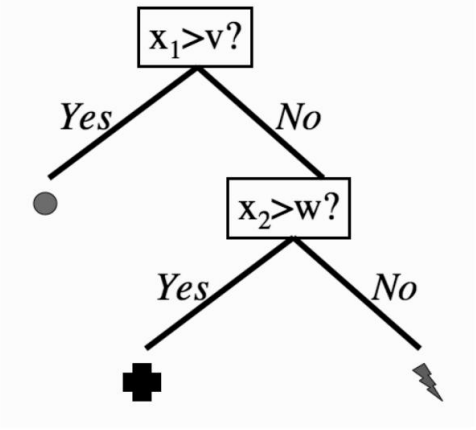

FIGURA 12.5 El árbol final creado por las divisiones en la Figura 12.4.

Si asumimos que el árbol está aproximadamente equilibrado, el coste en cada nodo consiste en buscar entre las d características posibles (aunque este valor disminuye en 1 en cada nivel, lo cual no afecta la complejidad en la notación O(·)) y, a continuación, calcular la ganancia de información del conjunto de datos para cada división. Esto tiene un coste de O(dn log n), donde n es el tamaño del conjunto de datos en ese nodo. Para la raíz, n = N, y si el árbol está equilibrado, n se divide entre 2 en cada etapa descendente del árbol. Al sumar esto sobre los niveles aproximadamente log N del árbol, se obtiene un coste computacional de O(dN² log N).

12.3 ÁRBOLES DE CLASIFICACIÓN Y REGRESIÓN (CART)

Existe otro algoritmo conocido basado en árboles, CART, cuyo nombre indica que puede utilizarse tanto para clasificación como para regresión. La clasificación no es muy diferente en CART, aunque suele limitarse a la construcción de árboles binarios. Esto puede parecer extraño al principio, pero existen sólidas razones informáticas que justifican la utilidad de los árboles binarios, como se sugiere en la discusión anterior sobre el coste computacional, y no constituye una limitación real. Incluso en el ejemplo con el que comenzamos el capítulo, siempre podemos convertir las preguntas en decisiones binarias dividiendo ligeramente la pregunta. Así, una pregunta con tres respuestas (por ejemplo, la pregunta sobre la fecha límite más cercana para su tarea, que puede ser "urgente", "próxima" o "ninguna") puede dividirse en dos preguntas: primero, "¿es urgente la fecha límite?"; y luego, si la respuesta es "no", segundo, "¿está próxima la fecha límite?". La única diferencia real con la clasificación en CART es que se suele utilizar una medida de información diferente. Esto se analiza a continuación, antes de analizar brevemente la regresión con árboles.

12.3.1 Impureza de Gini

La entropía utilizada en ID3 como medida de información no es la única forma de seleccionar características. Otra posibilidad es la impureza de Gini. La "impureza" en el nombre sugiere que el objetivo del árbol de decisión es que cada nodo hoja represente un conjunto de puntos de datos de la misma clase, para evitar discrepancias. Esto se conoce como pureza. Si una hoja es pura, todos sus datos de entrenamiento pertenecen a una sola clase. En ese caso, si contamos el número de puntos de datos en el nodo (o mejor dicho, la fracción del número de puntos de datos) que pertenecen a una clase i (llamémosla N(i)), debería ser 0 para todos los valores de i excepto uno. Supongamos que desea decidir qué característica elegir para una división. El algoritmo recorre las diferentes características y comprueba cuántos puntos pertenecen a cada clase. Si el nodo es puro, entonces N(i) = 0 para todos los valores de i excepto uno en particular.
Entonces, para cualquier característica particular k, puedes calcular:

$$G_k = \sum_{i=1}^{do} \sum_{j=i} N(i)N(j), \tag{12.8}$$

Donde c es el número de clases. De hecho, se puede reducir el esfuerzo algorítmico requerido con observando $\sum_i N(i) = 1$ (ya que debe haber alguna clase de salida) y, por lo tanto, $N(j)_{j=i} = 1$, que 1 − N(i). Entonces, la ecuación (12.8) es equivalente a:

$$G_k = 1 - \sum_{i=1}^{do} N(i)^2. \tag{12.9}$$

En cualquier caso, la impureza de Gini equivale a calcular la tasa de error esperada si la clasificación se seleccionó según la distribución de clases. La ganancia de información puede entonces medirse de la misma manera, restando cada valor Gi de la impureza total de Gini.

La medida de información se puede cambiar de otra manera, que es agregar un peso a
Las clasificaciones erróneas. La idea es considerar el costo de clasificar erróneamente una instancia de clase.
i como clase j (que llamaremos el riesgo en la Sección 2.3.1) y agregar un peso que indica cómo
La importancia de cada punto de datos. Normalmente se etiqueta como λ_{ij} y se presenta como una matriz, con
elemento λ_{ij} que representa el coste de clasificar erróneamente i como j. Su uso es sencillo, modificando el
La impureza de Gini (ecuación (12.8)) será:

$$G_i = \sum_{j=i} \lambda_{ij} N(i) N(j). \qquad (12.10)$$

Veremos en la Sección 13.1 que existe otro beneficio al utilizar estos pesos, que
Consiste en mejorar sucesivamente la capacidad de clasificación otorgando mayor peso a los puntos de datos.
que el algoritmo está equivocado.

12.3.2 Regresión en árboles

La novedad de CART es su aplicación en regresión. Aunque pueda parecer extraño...
Al utilizar árboles para la regresión, resulta que solo se requiere una modificación simple del algoritmo.
Supongamos que las salidas son continuas, por lo que un modelo de regresión es apropiado. Ninguno
Algunas de las medidas de impurezas de nodos que hemos considerado hasta ahora funcionarán. En su lugar, iremos...
Volviendo a nuestro viejo favorito: el error de suma de cuadrados. Para evaluar la elección de qué característica
Para utilizar a continuación, también necesitamos encontrar el valor en el que dividir el conjunto de datos de acuerdo con ese
Característica. Recuerde que la salida es un valor en cada hoja. En general, esto es solo una constante.
valor de la salida, calculado como el promedio de la media de todos los puntos de datos que se encuentran
en esa hoja. Esta es la opción óptima para minimizar el error de suma de cuadrados, pero
También significa que podemos elegir rápidamente el punto de división para una característica determinada, eligiendo
Para minimizar el error de suma de cuadrados, podemos elegir la característica que tiene la división.
punto que proporciona el mejor error de suma de cuadrados y continuar utilizando el algoritmo como para
clasificación.

12.4 EJEMPLO DE CLASIFICACIÓN (cuentas a mano no entran)

En esta sección, trabajaremos con un ejemplo usando ID3. Los datos que usaremos serán...
Continuación del que iniciamos el capítulo, sobre qué hacer por la noche.

Cuando queremos construir el árbol de decisiones para decidir qué hacer por la noche,
Comencemos enumerando todo lo que hemos hecho durante los últimos días para obtener un conjunto de datos adecuado.
(aquí, los últimos diez días):

¿Fecha límite?	¿Hay fiesta?	¿Perezoso?	¿Actividad?
Urgente	Sí		Partido del Sí
Urgente	No		Sí, estudia
Cerca	Sí		Partido del Sí
Ninguno	Sí		Sin fiesta
Ninguno	No		Sí Pub
Ninguno	Sí		Sin fiesta
Cerca	No		Sin estudio
Cerca	No		Sí TV
Cerca	Sí		Partido del Sí
Urgente	No		Sin estudio

Para producir un árbol de decisiones para este problema, lo primero que debemos hacer es trabajar Determinar qué característica usar como nodo raíz. Empezamos calculando la entropía de S:

$$\begin{aligned}\text{Entropía}(S) &= -p\text{party} \log_2 p\text{party} - p\text{study} \log_2 p\text{study} - p\text{pub} \log_2 p\text{pub} \\ &\quad - p\text{TV} \log_2 p\text{TV} \\ &= -\left(\frac{5}{10}\right)\log_2\left(\frac{5}{10}\right) - \left(\frac{3}{10}\right)\log_2\left(\frac{3}{10}\right) - \left(\frac{1}{10}\right)\log_2\left(\frac{1}{10}\right) - \left(\frac{1}{10}\right)\log_2\left(\frac{1}{10}\right) \\ &= 0{,}5 + 0{,}5211 + 0{,}3322 + 0{,}3322 = 1{,}6855\end{aligned} \tag{12.11}$$

y luego encuentre qué característica tiene la máxima ganancia de información:

$$\begin{aligned}\text{de |Surgimiento| (S, Fecha límite)} &= 1{,}6855 - \text{Entropía (Surgimiento)} \\ &\quad - \frac{|\text{Casi}|}{10}\text{Entropía(Snear)} - \frac{|\text{Ninguno}|}{10}\text{Entropía (Snone)} \\ &= 1.6855 - \frac{3}{10}\left(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right) \\ &\quad - \frac{4}{10}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4}\right) \\ &\quad - \frac{3}{10}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right) \\ &= 1{,}6855 - 0{,}2755 - 0{,}6 - 0{,}2755 \\ &= 0{,}5345\end{aligned} \tag{12.12}$$

$$\begin{aligned}\text{Ganancia(S,Partido)} &= 1.6855 - 10\left(\frac{5}{10} - \frac{5}{5}\log_2\frac{5}{5}\right) \\ &\quad - \frac{5}{10}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{1}{5}\log_2\frac{1}{5} - \frac{1}{5}\log_2\frac{1}{5}\right) \\ &= 1{,}6855 - 0 - 0{,}6855 \\ &= 1.0\end{aligned} \tag{12.13}$$

$$\begin{aligned}\text{Ganancia(S, Lazy)} &= 1.6855 - \left(\frac{6}{10} - \frac{3}{6}\log_2\frac{3}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6}\right) \\ &\quad - \frac{4}{10}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right) \\ &= 1{,}6855 - 1{,}0755 - 0{,}4 \\ &= 0{,}21\end{aligned} \tag{12.14}$$

Por lo tanto, el nodo raíz será la característica de la fiesta, que tiene dos valores de característica ('sí' y 'no'), por lo que tendrá dos ramas que salen de ella (véase la Figura 12.6). Al observar la rama 'sí', vemos que en los cinco casos donde había una fiesta, la visitamos, así que simplemente colocamos un nodo hoja allí con el nombre 'fiesta'. Para la rama 'no', de los cinco casos hay tres resultados diferentes, por lo que ahora debemos elegir otra característica. Los cinco casos que analizamos son:

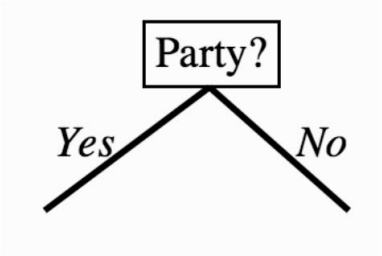

FIGURA 12.6 El árbol de decisión después de un paso del algoritmo.

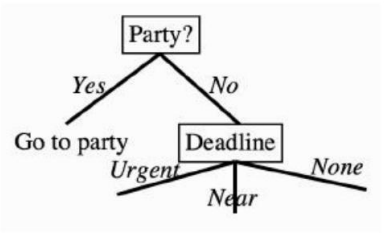

FIGURA 12.7 El árbol tras otro paso.

¿Fecha límite?	¿Hay fiesta?	¿Perezoso?	Actividad
Urgente	Sí	Estudiar	Estudiar
Ninguno	Sí	Pub	Cerca
Urgente	No	Estudiar	Cerca
Sí	Sí	TV	TV

Hemos utilizado la función de fiesta, por lo que solo necesitamos calcular la ganancia de información de los otros dos en estos cinco ejemplos:

$$\begin{aligned} 2\,\text{Ganancia}(S, \text{Fecha límite}) &= 1.371 - \left(\frac{2}{5}\right)\left(-\frac{2}{2}\log_2\frac{2}{2}\right) - \left(\frac{1}{5}\right)\left(-\frac{1}{1}\log_2\frac{1}{1}\right) \\ &\quad - \left(\frac{2}{5}\right)\left(-\frac{1}{1}\log_2\frac{1}{1}\right) - \left(\frac{1}{5}\right)\left(-\frac{1}{1}\log_2\frac{1}{1}\right) \\ &= 1.371 - 0 - 0.4 - 0 \\ &= 0{,}971 \end{aligned} \tag{12.15}$$

$$\begin{aligned} 4\,\text{Ganancia}(S, \text{Perezoso}) &= 1.371 - 5\left(\frac{2}{4}\right)\left(-\frac{2}{2}\log_2\frac{2}{4}\right) - \left(\frac{1}{4}\right)\left(-\frac{1}{1}\log_2\frac{1}{4}\right) - \left(\frac{1}{4}\right)\left(-\frac{1}{1}\log_2\frac{1}{4}\right) \\ &\quad - \left(\frac{1}{5}\right)\left(-\frac{1}{1}\log_2\frac{1}{1}\right) \\ &= 1.371 - 1.2 - 0 \\ &= 0{,}1710 \end{aligned} \tag{12.16}$$

Esto conduce al árbol que se muestra en la Figura 12.7. A partir de aquí, es relativamente sencillo completar el árbol, que conduce al que se muestra en la Figura 12.1.

LECTURAS ADICIONALES

Para obtener más información sobre los árboles de decisión, los dos libros siguientes son de interés:

- JR Quinlan. C4.5: Programas para aprendizaje automático. Morgan Kaufmann, San Francisco Cisco, CA, EE. UU., 1993.
- L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone. Clasificación y regresión Árboles. Chapman & Hall, Nueva York, EE. UU., 1993.