Árboles de decisión

Índice

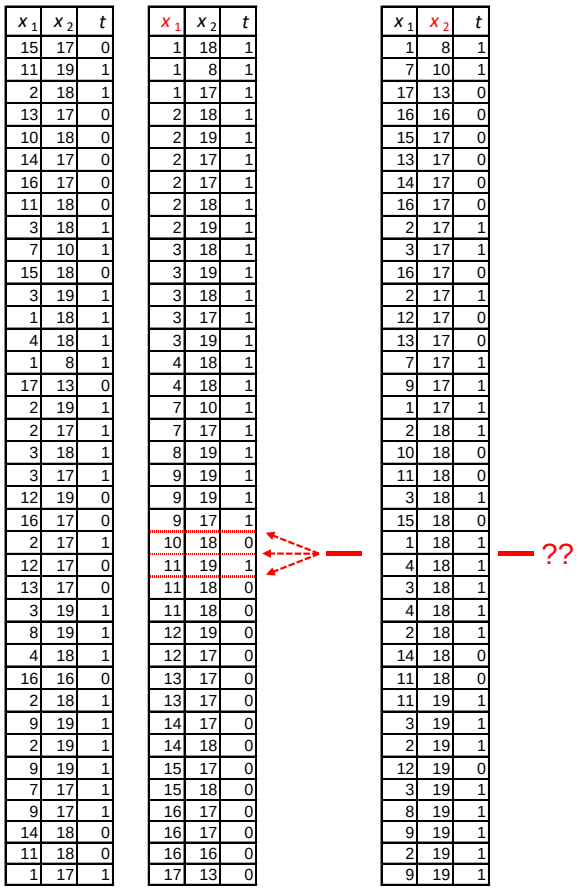
Referencia principal

Cap.6 de "Hands On Machine Learning with Scikit Learn and TensorFlow"

Al terminar este tema entenderemos las propiedades y expresividad de los árboles de decisión, así como los criterios con los que se construyen.

1. Intuición

En la tabla de la izquierda de Figura 1 se muestran 38 instancias $\mathbf{x} = (x_1, x_2)$ de ceros y unos tal y como se obtuvieron en el ejemplo-guía de la semana 2. A continuación se muestra dicha tabla ordenada ascendentemente, primero según x_1 (centro) y después según x_2(derecha).



x_1	x_2	t
15	17	0
11	19	1
2	18	1
13	17	0
10	18	0
14	17	0
16	17	0
11	18	0
3	18	1
7	10	1
15	18	0
3	19	1
1	18	1
4	18	1
1	8	1
17	13	0
2	19	1
2	17	1
3	18	1
3	17	1
12	19	0
16	17	0
2	17	1
12	17	0
13	17	0
3	19	1
8	19	1
4	18	1
16	16	0
2	18	1
9	19	1
2	19	1
9	19	1
7	17	1
9	17	1
14	18	0
11	18	0
1	17	1

x_1	x_2	t
1	18	1
1	8	1
1	17	1
2	18	1
2	19	1
2	17	1
2	17	1
2	18	1
2	19	1
3	18	1
3	19	1
3	18	1
3	17	1
3	19	1
4	18	1
4	18	1
7	10	1
7	17	1
8	19	1
9	19	1
9	19	1
9	17	1
10	18	0
11	19	1
11	18	0
11	18	0
12	19	0
12	17	0
13	17	0
13	17	0
14	17	0
14	18	0
15	17	0
15	18	0
16	17	0
16	17	0
16	16	0
17	13	0

x_1	x_2	t
1	8	1
7	10	1
17	13	0
16	16	0
15	17	0
13	17	0
14	17	0
16	17	0
2	17	1
3	17	1
16	17	0
2	17	1
12	17	0
13	17	0
7	17	1
9	17	1
1	17	1
2	18	1
10	18	0
11	18	0
3	18	1
15	18	0
1	18	1
4	18	1
3	18	1
4	18	1
2	18	1
14	18	0
11	18	0
11	19	1
3	19	1
2	19	1
12	19	0
3	19	1
8	19	1
9	19	1
2	19	1
9	19	1

Figura 1: 'Algoritmo' para decidir por dónde dividir. [Fuente: Original de A. Cuesta]

Cuando ordenamos por x_1 podemos ver que prácticamente todas las instancias con etiqueta $t = 1$ satisfacen que $x_1 \leq 9$. Vamos a contemplar esa posibilidad frente a otras dos.

- Si dividimos el conjunto por $x_1 \leq 9$, entonces el subconjunto 'de arriba' sólo tiene etiquetas $t = 1$, mientras que el 'de abajo' sólo tiene etiquetas $t = 0$ EXCEPTO en la instancia $\mathbf{x} = (11, 19)$, que tiene etiqueta $t = 1$.

  El resultado de esta división es que logramos un subconjunto 'sin impurezas' y otro sólo con una 'impureza'. Si pensamos en la tarea de clasificación como la de separar dos clases obteniendo un resultado 'puro' de ambas, entonces parece que el criterio $x_1 \leq 9$ es una muy buena opción.

- Si dividimos por $x_1 \leq 10$ entonces habría una impureza en cada subconjunto.

  No está mal, aunque es preferible tener subconjuntos puros a impuros.

- Si elegimos $x_1 \leq 11$, puesto que hay tres instancias que lo cumplen y dos de ellas tienen etiqueta $t = 0$, el subconjunto de arriba tendría 3 impurezas, pero ahora el de abajo no tendría ninguna.

  Entre esta opción y la primera, esta parece peor porque el subconjunto puro contiene menos ejemplos y el impuro contiene más impurezas.

Si, en vez de x_1, ordenamos por x_2, podemos comprobar que no es posible obtener una división que logre dos subconjuntos tan puros como antes. En definitiva, con este método hemos llegado a una nueva función discriminante:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{si} \quad x_1 \leq 9 \\ 0 & \text{si} \quad x_1 > 9 \end{cases}$$

¿Y si no hubiera sido posible una división tan 'pura'? Suponiendo que, con la primera división hemos conseguido que todas las impurezas estén sólo en uno de los dos subconjuntos, y el otro es puro, entonces volvemos a realizar la operación de división sobre el conjunto impuro. Es decir, probamos de nuevo con cada una de las características buscando dividir el subconjunto en dos y tal que al menos uno de ellos sea puro.

Este método se denomina **árbol de decisión** porque cada vez que hacemos una división en dos subconjuntos creamos un nodo en el que escribimos el criterio, y del cual salen dos arcos que representan la condición cierta o falsa. Si un arco termina en una hoja del árbol, entonces escribimos la etiqueta que se corresponde con la secuencia de decisiones hasta llegar a ella. Si llega a un nuevo nodo significa que vamos a hacer una nueva división. Por ejemplo, la función discriminante de arriba daría lugar al árbol de la Figura 2, con un único nodo y dos hojas.

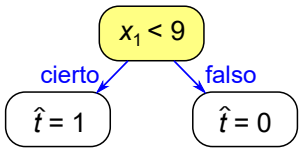


Figura 2: Árbol de decisión con 1 nodo y dos hojas. [Fuente: Original de A. Cuesta]

El proceso finaliza cuando logramos subconjuntos sin impurezas. Sin embargo esto puede dar lugar a sobreajuste, con lo cual podemos imponer criterios de parada, como una profundidad máxima por ejemplo . En la Figura 3 se muestran 3 ejemplos con diferentes profundidades.

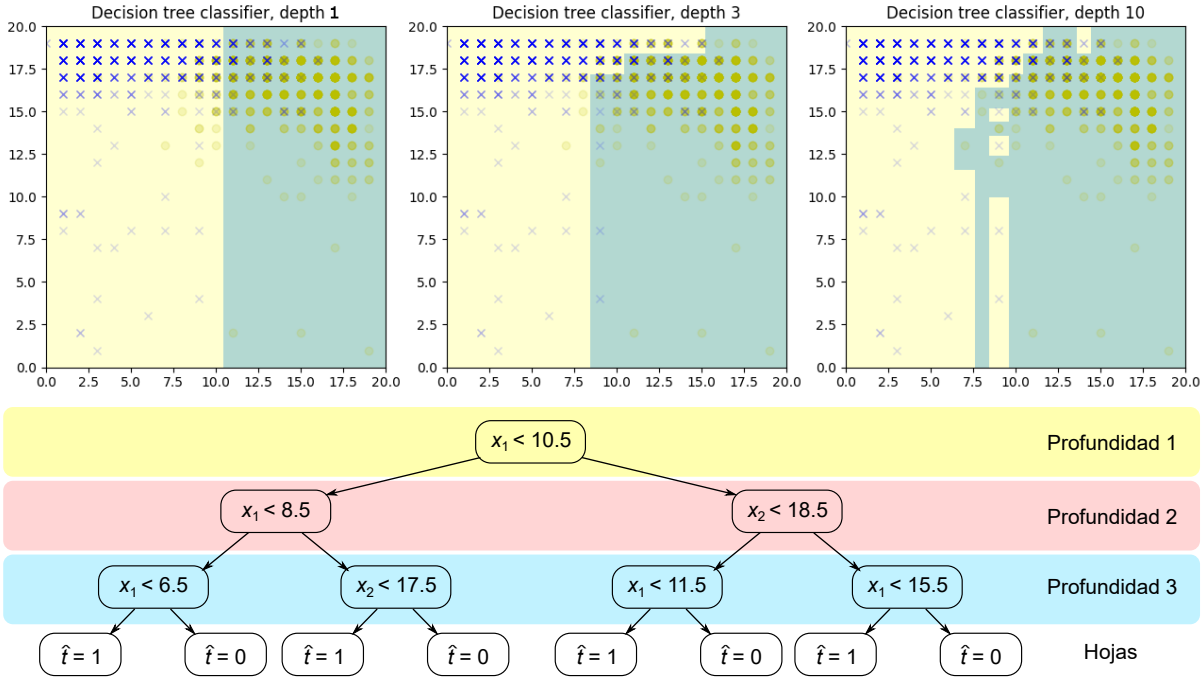


Figura 3: (Arriba) Resultado de tres clasificadores, de profundidad 1, 3 y 10. (Abajo) Árbol de decisión del gráfico de arriba-centro. [Fuente: Original de A. Cuesta]

2. Construcción del árbol

Para construir un nodo del árbol comenzamos asumiendo que:

- Queremos realizar una clasificación binaria (igual que desde el comienzo)
- Tenemos un conjunto de m datos n dimensionales $X = \{\mathbf{x}^{(i)}\}$ y etiquetas $\mathbf{t}^{(i)}$ asociadas.

  Dichos datos pueden ser el conjunto de entrenamiento original o su subconjunto del mismo.
  - Cuando es el conjunto original significa que estamos comenzando el árbol, o sea que es el nodo 'raíz'.
  - Cuando es un subconjunto significa que ya hemos hecho alguna división, por tanto estamos creando un nodo 'interior'.
- Debemos decidir una medida de la 'impureza' de los subconjuntos que se crean a partir del nodo en el que estamos según la condición que impongamos.

  Hay dos medidas importantes:
  - El coeficiente de impureza de Gini
  - La ganancia de información

2.1. Algoritmo CART

El algoritmo *Classification and Regression Tree* (CART) es un método 'avaricioso' o *greedy* de construir un nodo.

Dada la característica x_k, y un umbral θ_k, dividimos el conjunto dado en dos:

$$\begin{cases} \text{A la "izquierda" aquellos } \mathbf{x} : x_k \leq \theta_k \\ \text{A la "derecha" el resto} \end{cases}$$

Para elegir el mejor par (x_k, θ_k) necesitamos una función de coste que minimizar. En el algoritmo CART dicha función es:

$$J(x_k, \theta_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}, \tag{1}$$

donde:

- $m = m_{\text{left}} + m_{\text{right}}$, es el número de ejemplos que se están evaluando el nodo que estamos construyendo.
- m_{left} y m_{right} es el número de ejemplos en el subconjunto izquierdo y derecho respectivamente.
- G_{left} y G_{right} son los coeficientes de impureza de Gini de los subconjuntos izquierdo y derecho respectivamente.

El **coeficiente de impureza de Gini** se calcula con la expresión:

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

donde $p_{i,j}$ es el ratio de impurezas, es decir la proporción de ejemplos de la clase j en el subconjunto i. Para el caso particular de la función de coste (1), de un problema de clasificación binaria, los coeficientes de Gini serían:

$$G_{\text{left}} = 1 - p_{\text{left},0}^2 - p_{\text{left},1}^2, \quad \text{y} \quad G_{\text{right}} = 1 - p_{\text{right},0}^2 - p_{\text{right},1}^2.$$

Si, por ejemplo, el subconjunto izquierdo no tuviera 'impurezas', entonces:

$$G_{\text{left}} = 1 - \left(\frac{0}{m}\right)^2 - \left(\frac{m}{m}\right)^2 = 1 - 0 - 1 = 0,$$

como era de esperar. Evidentemente $0 \leq G_i \leq 1$ siempre.

2.2. Predicción

Una vez construido el árbol, predecir la etiqueta de un nuevo ejemplo es sencillo. Comenzando por la raíz, se evalúa la característica del ejemplo que ha generado ese nodo y según el resultado sea verdadero o falso se elige la rama de la izquierda o de la derecha respectivamente, llegando o bien a un nuevo nodo o bien a una hoja. Si es un nuevo nodo, se repite el proceso con la característica que se indique. Si es una hoja, entonces esta indica la etiqueta estimada.

En la Figura 4 se muestra este proceso para un hipotético nuevo ejemplo $\mathbf{x} = (9, 12)$ clasificado según el árbol de decisión de la Figura 3

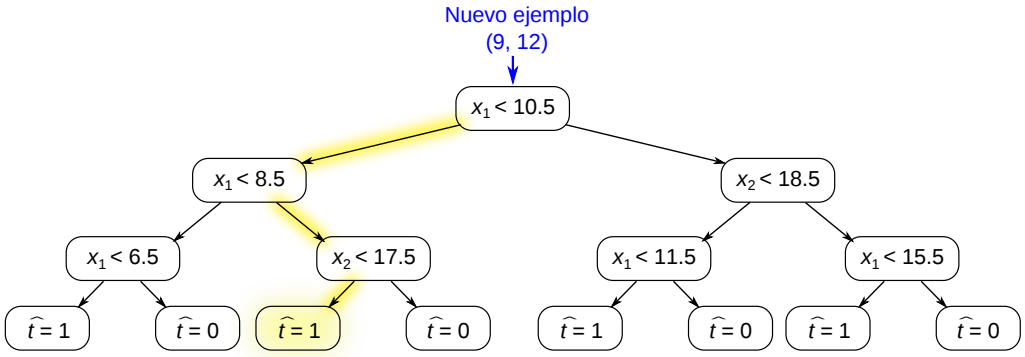


Figura 4: Clasificación con un árbol de decisión [Fuente: Original de A. Cuesta]

3. Comentarios

3.1. Coste computacional

Para entrenar un árbol con el algoritmo CART es necesario recorrer la tabla con dos bucles `for` anidados. El primero se mueve en horizontal, característica tras característica, mientras que el segundo se mueve en vertical, ejemplo tras ejemplo. Sin embargo, según avanza el algoritmo cada vez hay menos ejemplos; de modo que se puede demostrar que el coste final es $O(n \times m \log m)$.

Pero el árbol resultante no es el mejor posible porque CART es un algoritmo *greedy*. La función de coste (1) es válida sólo para los ejemplos que hay en el nodo que se está construyendo. Esto significa que busca la mejor separación en ese nodo. Pero puede darse el caso de que, eligiendo una separación un poco peor, en los nodos siguientes la separación sea mejor, y el resultado global también. Para asegurarnos deberíamos construir todos los árboles posibles, pero esto tiene un coste exponencial en el número de ejemplos, es decir se trata de un problema NP.

Por último, en la Figura 1 se ordenaba el conjunto de datos para explicar mejor el funcionamiento del algoritmo. Ordenar la tabla añade un coste computacional que, si el conjunto de datos es muy grande, hace aún más lento el algoritmo.

3.2. Entropía

La entropía es, en origen, un concepto de la termodinámica estadística, relacionada con la capacidad de un sistema para producir un trabajo. Popularmente se asocia un aumento de entropía con un aumento del desorden de las partículas. Por ejemplo, en un émbolo lleno de aire, si apretamos el pistón las partículas se comprimen produciéndose más choques entre ellas. En el límite, comprimiendo al máximo, las partículas estarían tan inmóviles, como en el cero absoluto de temperatura, con entropía nula (que es inalcanzable). Si lo soltamos el pistón vuelve a subir de modo que se ha producido un trabajo y también un aumento de entropía. Sin embargo, de un émbolo donde las partículas estuvieran separadas y uniformemente distribuidas, sin interacción entre ellas es imposible obtener ningún trabajo y por tanto se puede producir ningún incremento de entropía. Otro ejemplo (macroscópico) es el de los aerogeneradores. Suponiendo que no tuvieran ningún rozamiento, cuando hay viento se mueven produciendo electricidad, pero si el aire está totalmente en reposo no es posible obtener ningún trabajo.

El concepto se ha extendido a otros dominios pero siempre con un significado similar. En ML supervisado, la entropía nula significa que los ejemplos en un nodo ya no tienen la capacidad de producir nuevos nodos hacia abajo. Por tanto la entropía es nula cuando es la tabla de ejemplos etiquetados en un nodo es 'pura' y positiva cuando contiene 'impurezas', igual que Gini. La expresión de la entropía, siguiendo la misma notación que con Gini, es:

$$H_i = -\sum_{j=1}^{n} p_{i,j} \log(p_{i,j}) \ , \quad \text{siempre que} \quad p_{i,j} > 0 \ ;$$

y para clasificación binaría se particulariza en

$$H_{\text{left}} = -p_{\text{left},0} \log(p_{\text{left},0}) - p_{\text{left},1} \log(p_{\text{left},1}), \quad \text{y} \quad H_{\text{right}} = -p_{\text{right},0} \log(p_{\text{right},0}) - p_{\text{right},1} \log(p_{\text{right},1}).$$

¿Cuál usar entonces? En cuanto a resultado, la mayoría de las veces dan prácticamente el mismo árbol pero Gini es más rápido de calcular. cuando difieren, Gini tiende a aislar la clase más frecuente en una rama mientras que la entropía suele generar árboles más equilibrados.

Índice alfabético