Árboles de decisión

UCO ONLINE

Árboles de decisión

Introducción

Los árboles de decisión se consideran uno de los métodos más populares para la representación de clasificadores. Un árbol de decisión es un clasificador expresado como particiones recursivas del espacio de entrada. El árbol se constituye por un conjunto de nodos, comenzando con un nodo llamado "raíz", donde se realiza la partición inicial. Se diferencia también entre los nodos internos o de decisión (aquellos de los que salen ramificaciones para nuevos nodos en niveles inferiores), y los nodos hoja o terminales.

En un árbol de decisión, cada nodo divide los datos de entrada en dos o más subgrupos, de acuerdo a una función discreta de los atributos de entrada; en el caso más simple y frecuente, cada nodo considera un único atributo. En el caso de atributos numéricos, las condiciones se refieren a rangos de valores. Por otro lado, cada nodo hoja se asigna a una clase, coincidiendo con la que mejor representada esté entre las instancias de los datos disponibles en dicho nodo. Esta asignación podría convertirse también en una probabilidad de pertenencia.

Para predecir una nueva instancia, habría que navegar en el árbol a través de sus particiones, siguiendo las ramas que correspondan según el valor de los datos de la instancia concreta. Al llegar a una hoja, se clasifica la instancia según la clase que indique dicha hoja. En la Figura 1 se puede observar un ejemplo de árbol de decisión, en el problema del tiempo atmosférico. Para una instancia dada, y en base a su atributo *outlook*, se seguirá un camino u otro: si está soleado, por la rama de la izquierda; si llueve, por la rama de la derecha; y si está soleado, por la rama central. En caso de que el tiempo sea nublado, se puede tomar directamente una decisión, ya que en los datos de entrenamiento existían 4 instancias para las que estaba nublado y se podía jugar, y en ninguna no se podía jugar. Suponiendo que el tiempo era soleado, encontramos una nueva partición: si la humedad es menor o igual a 75, la decisión será que sí se puede jugar, mientras que, si la humedad es superior a 75, la decisión será que no. Nótese que en este último caso, en los datos de entrenamiento existían 2 instancias para las que el tiempo estaba soleado y la humedad por encima de 75 en que no se jugó, por 1 que sí; por tanto, a una nueva instancia de los datos se asigna la clase mayoritaria.

Como se puede observar, los árboles de decisión son altamente interpretables. No solo proporcionan la predicción de una clase (o su confianza de pertenecer a cada clase) para cada nuevo patrón, sino que muestran de una forma sencilla los pasos seguidos para llegar hasta dicha conclusión. Así, proporcionan mucha más información al experto que otros modelos de caja negra o no interpretables. Por ello, es también importante controlar la complejidad de los árboles, que

puede medirse en número total de nodos, número total de nodos hoja, profundidad del árbol, o número de atributos utilizados.

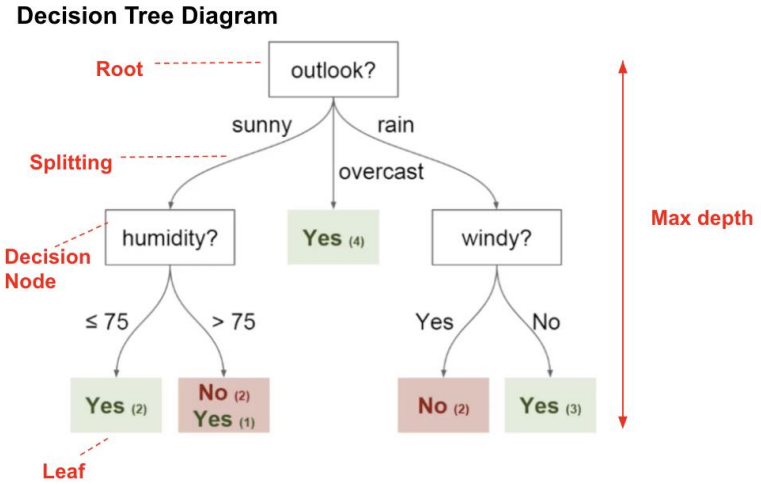

Figura 1. Ejemplo de árbol de decisión para el problema del tiempo atmosférico.

Inducción de árboles de decisión

Se llama inductores de árboles de decisión a los algoritmos capaces de construir dichos árboles a partir de un conjunto de datos. Por lo general, el objetivo es encontrar el árbol de decisión óptimo que minimice el error de generalización. Sin embargo, también se pueden definir otros objetivos, como minimizar el número de nodos o minimizar la profundidad máxima o media de las ramas.

Se ha demostrado ampliamente que obtener un árbol de decisión óptimo es un problema complejo, por lo que es necesario utilizar heurísticas para obtener dichos árboles en tiempo razonable. Generalmente, los métodos pueden dividirse en dos grupos: inductores *top-down* (de arriba a abajo) y *bottom-up* (de abajo a arriba), donde los primeros suelen ser preferibles. SSon algoritmos de tipo *top-down* ampliamente conocidos: ID3 [Qui86], C4.5 [Qui93], y CART [Bre84]. El primero de ellos se construye utilizando una técnica de crecimiento del árbol (*growing*), mientras que los otros dos también incluyen una técnica de podado (*pruning*). Estos algoritmos suelen ser voraces (o *greedy*), construyendo el árbol de forma recursiva. En cada iteración, se escoge la partición del conjunto de entrenamiento usando alguna métrica de particionado en base a los atributos de entrada. Tras seleccionar una partición apropiada, cada nodo se subdivide en subconjuntos más pequeños, hasta que no existe una división que obtenga una ganancia suficiente, o hasta que se cumpla un criterio de parada.

CART

El método *Classification and regression trees* (CART) fue propuesto por Breiman *et al.* en 1984 [Bre84]. Los árboles de decisión producidos por CART son estrictamente binarios, conteniendo únicamente dos ramas en cada nodo de decisión. CART particiona recursivamente las instancias en el conjunto de entrenamiento en subconjuntos de instancias con valores similares para la variable de salida.

En cada nodo de decisión, CART realiza una búsqueda exhaustiva de todas las variables y posibles particiones, seleccionando el particionado óptimo de acuerdo al siguiente criterio. Siendo $\Phi(s|t)$ la métrica de bondad de un particionado s en un nodo t, el particionado óptimo será aquel que maximiza el valor de $\Phi(s|t)$ de entre todos los posibles particionados:

$$\Phi(s|t) = 2P_L P_R \sum_{j=1}^{\#clases} |P(j|t_L) - P(j|t_R)|$$

donde:

$$t_L = \text{hijo izquierdo del nodo } t$$

$$t_R = \text{hijo derecho del nodo } t$$

$$P_L = \frac{\text{Número de instancias en } t_L}{\text{Número de instancias en el conjunto de entrenamiento}}$$

$$P_R = \frac{\text{Número de instancias en } t_R}{\text{Número de instancias en el conjunto de entrenamiento}}$$

$$P(j|t_L) = \frac{\text{Número de instancias de la clase } j \text{ en } t_L}{\text{Número de instancias en } t_L}$$

$$P(j|t_R) = \frac{\text{Número de instancias de la clase } j \text{ en } t_R}{\text{Número de instancias en } t_R}$$

Supongamos un ejemplo con el conjunto de datos que aparece en la Tabla 1, donde se pretende construir un árbol de decisión para predecir si un cliente dado se puede clasificar con un riesgo de crédito bueno o malo. En principio, las 8 instancias de los datos entran en el nodo raíz del árbol. Como CART solo permite particiones binarias, el algoritmo evaluará las particiones iniciales que se indican en la Tabla 2. Nótese que aunque *Income* es una variable continua, CART identifica una

lista finita de las posibles particiones en base a los diferentes valores que toma esa variable en el dataset; también podría convertirse dicha variable continua en categórica tomando un menor conjunto de valores.

Tabla 1. Dataset de ejemplo para CART.

Customer	Savings	Assets	Income ($1000s)	Credit Risk
1	Medium	High	75	Good
2	Low	Low	50	Bad
3	High	Medium	25	Bad
4	Medium	Medium	50	Good
5	Low	Medium	100	Good
6	High	High	25	Good
7	Low	Low	25	Bad
8	Medium	Medium	75	Good

Tabla 2. Posibles particiones para el nodo raíz en el ejemplo de CART.

Candidate Split	Left Child Node, t_L	Right Child Node, t_R
1	$Savings = low$	$Savings \in \{medium, high\}$
2	$Savings = medium$	$Savings \in \{low, high\}$
3	$Savings = high$	$Savings \in \{low, medium\}$
4	$Assets = low$	$Assets \in \{medium, high\}$
5	$Assets = medium$	$Assets \in \{low, high\}$
6	$Assets = high$	$Assets \in \{low, medium\}$
7	$Income \leq \$25{,}000$	$Income > \$25{,}000$
8	$Income \leq \$50{,}000$	$Income > \$50{,}000$
9	$Income \leq \$75{,}000$	$Income > \$75{,}000$

Para cada posible partición, examinamos el valor de $\Phi(s|t)$. Podemos analizar que dicho valor de bondad alcanza un valor alto cuando cada una de sus componentes ($2P_LP_R$ y $\sum_{j=1}^{\#clases}|P(j|t_L) - P(j|t_R)|$) son valores altos. La componente $\sum_{j=1}^{\#clases}|P(j|t_L) - P(j|t_R)|$ obtiene valores altos cuando las proporciones de instancias en los nodos hijo para cada valor de la clase son lo más diferentes posibles, obteniendo el máximo valor cuando para cada clase los nodos hijo son

completamente uniformes. La componente $2P_L P_R$ se maximiza cuando las proporciones de instancias en ambos nodos hijo son iguales; es decir, busca particiones balanceadas de los nodos, conteniendo similar número de instancias en cada uno.

En el ejemplo (Tabla 3) el mejor particionado sería el número 4, donde $\Phi(s|t) = 0.6248$. Por tanto, del nodo raíz se obtendrán dos nodos hijo: uno donde estarán los patrones cuya variable *assets* tome el valor *low*; y otro donde estarán los patrones cuya variable *assets* tome los valores *medium* o *high*. El nodo izquierdo se convierte a su vez en un nodo terminal, ya que todos los patrones de este nodo (instancias 2 y 7) tienen un riesgo de crédito malo. Sin embargo, el riesgo de crédito en el nodo derecho aún es diverso, por lo que habrá que hacer nuevos particionados recursivos, utilizando exclusivamente los patrones dentro de este nodo (aquellos donde *assets* sea *medium* o *high*). De nuevo, se calcularía la bondad de cada posible particionado en el nuevo nodo, eligiendo en qué atributo particionar en cada nodo hasta solo obtener nodos hoja. El árbol resultante en este ejemplo sería el que se ve en la Figura 2.

Tabla 3. Valores de $\Phi(s|t)$ para cada posible partición en el ejemplo de CART.

Split	P_L	P_R	$P(j\|t_L)$	$P(j\|t_R)$	$2P_L P_R$	$Q(s\|t)$	$\Phi(s\|t)$
1	0.375	0.625	G: .333 B: .667	G: .8 B: .2	0.46875	0.934	0.4378
2	0.375	0.625	G: 1 B: 0	G: 0.4 B: 0.6	0.46875	1.2	0.5625
3	0.25	0.75	G: 0.5 B: 0.5	G: 0.667 B: 0.333	0.375	0.334	0.1253
4	0.25	0.75	G: 0 B: 1	G: 0.833 B: 0.167	0.375	1.667	0.6248
5	0.5	0.5	G: 0.75 B: 0.25	G: 0.5 B: 0.5	0.5	0.5	0.25
6	0.25	0.75	G: 1 B: 0	G: 0.5 B: 0.5	0.375	1	0.375
7	0.375	0.625	G: 0.333 B: 0.667	G: 0.8 B: 0.2	0.46875	0.934	0.4378
8	0.625	0.375	G: 0.4 B: 0.6	G: 1 B: 0	0.46875	1.2	0.5625
9	0.875	0.125	G: 0.571 B: 0.429	G: 1 B: 0	0.21875	0.858	0.1877

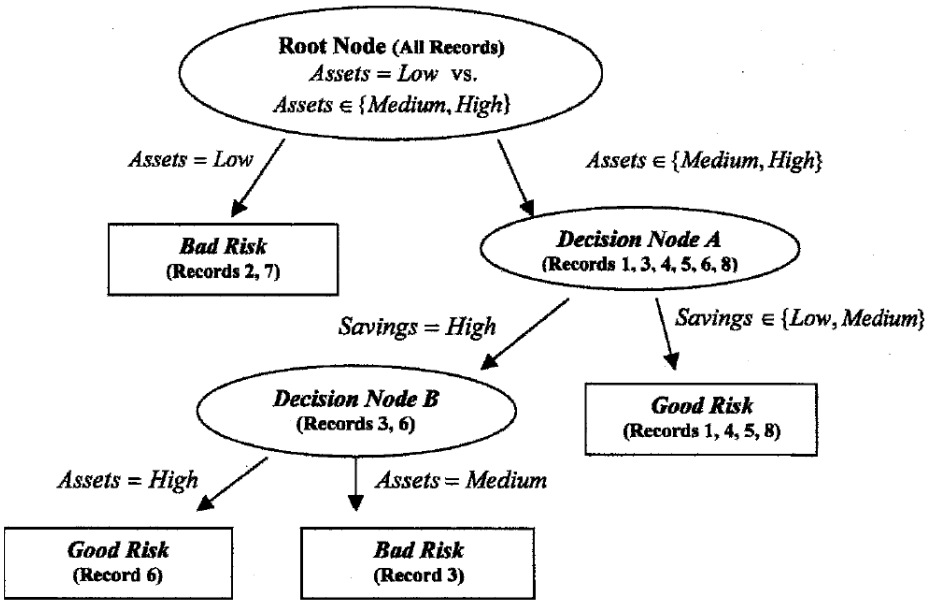

Figura 2. Árbol de decisión completo en el ejemplo de CART.

Este ejemplo se construye el árbol hasta que los nodos hoja tienen instancias de una sola clase (nodos puros), por lo que el error en entrenamiento se ha optimizado. Sin embargo, en algunos casos reales, y dado que los nodos cada vez incluyen menos instancias, podría hacer que el árbol sea demasiado complejo y conlleve un gran sobreentrenamiento. Para ello, se podría realizar un podado del árbol, de modo que se penalice el crecimiento del árbol por tener muchos nodos hoja (y por tanto mucha complejidad). De este modo, podriamos tener un árbol, donde no todos los nodos hoja son puros, clasificando las nuevas instancias en la clase mayoritaria de dicho nodo (como ya se vió en la Figura 1).

C4.5

El algoritmo C4.5 [Qui93] es una extensión del algoritmo ID3 [Qui86] del mismo autor. La construcción del árbol en C4.5 es similar a CART, pero existen algunas diferencias interesantes entre ambos:

- C4.5 no está restringido a particiones binarias. Por tanto, mientras que CART produce solo árboles binarios, C4.5 puede producir árboles de formas variadas.

- Para las variables categóricas, C4.5 produce una rama separada (o nodo hijo distinto) para cada posible valor de la variable por defecto. Esto también podría producir que el árbol tenga más ramificaciones del as deseadas, ya que algunos valores podrían tener una frecuencia de aparición muy baja, o estar naturalmente asociados a otros valores.
- El método para calcular la homogeneidad de los nodos en C4.5 es diferente a como se hace en CART, y se expone a continuación.

C4.5 utiliza el concepto de ganancia de información (*information gain)* o reducción de entropía (*entropy reduction*) para seleccionar el mejor particionado. Supongamos una variable X cuyos k posibles valores tienen probabilidades $p_1, \ldots, p_k$. La entropía de X se define como:

$$H(X) = -\sum_j p_j \log_2 p(j)$$

siendo j cada uno de los posibles valores del atributo en cuestión.

C4.5 utiliza este concepto de entropía de la siguiente manera. Supongamos que tenemos un posible particionado S, que particiona los datos de entrenamiento T en varios subconjuntos $T1, \ldots, Tk$. Se define la ganancia de información como $gain(S) = H(T) - H_S(T)$, donde $H_S(T)$ se calcula como la suma ponderada de las entropías de cada uno de los subconjuntos tras el particionado; en la fórmula, P_i indica la proporción de instancias en el subconjunto i. Posteriormente, C4.5 selecciona el particionado óptimo, es decir, aquel que obtiene una mayor ganancia de información (mayor incremento de información producido por el particionado del conjunto T).

$$H_S(X) = \sum_{i=1}^{k} P_i H(T_i)$$

Supongamos el mismo ejemplo en la Tabla 1. Las posibles particiones de C4.5 en el nodo raíz serían las que se observan en la

Tabla 4. Nótese que el tratamiento de C4.5 para los atributos numéricos sí que es el mismo que en CART. En primer lugar, en el nodo raíz como 5 de las 8 instancias están clasificadas como *buen riesgo de crédito* y 3 como *malo*, la entropía antes de hacer el particionado es:

$$H(T) = -\left(\frac{5}{8}\log_2\left(\frac{5}{8}\right) - \frac{3}{8}\log_2\left(\frac{3}{8}\right)\right) = 0.9544$$

Tabla 4. Posibles particiones para C4.5 en el nodo raíz para el ejemplo propuesto.

Candidate Split	Child Nodes		
1	*Savings* = *low*	*Savings* = *medium*	*Savings* = *high*
2	*Assets* = *low*	*Assets* = *medium*	*Assets* = *high*
3	*Income* ≤ $25,000		*Income* > $25,000
4	*Income* ≤ $50,000		*Income* > $50,000
5	*Income* ≤ $75,000		*Income* > $75,000

Para calcular la bondad del primer posible particionado calculamos primero la entropía de cada uno de los posibles nodos hijo ($H_s(T_i)$); la entropía del particionado dados los resultados anteriores ($H_s(T)$); y por último la ganancia de información del particionado ($gain(S)$). A modo ilustrativo, téngase en cuenta que en el caso que *savings=low*, 1 de los 3 patrones es de la clase *good*, mientras que los otros dos son de la clase *bad*. En el caso de *savings=medium* todos los patrones son de la clase *good*; nótese que se toma como convención que el valor de $\log_2(0) = 0$.

$$\left(H_{savings=low}(T)\right) = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right) = 0.9183$$
$$\left(H_{savings=medium}(T)\right) = -\frac{3}{3}\log_2\left(\frac{3}{3}\right) - \frac{0}{3}\log_2\left(\frac{0}{3}\right) = 0$$
$$\left(H_{savings=high}(T)\right) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

$$H_{savings}(T) = \frac{3}{8} * 0.9183 + \frac{3}{8} * 0 + \frac{2}{8} * 1 = 0.5944$$

$$gain(savings) = H(T) - H_{savings}(T) = 0.9544 - 0.5944 = 0.3600$$

Si calculamos el resto de los valores de ganancia de información para los distintos particionados, obtenemos:

$$gain(assets) = \mathbf{0.5487}$$
$$gain(income \leq 25000) = 0.1588$$
$$gain(income \leq 50000) = 0.3475$$
$$gain(income \leq 75000) = 0.0923$$

Dados estos resultados, en el nodo raíz C4.5 seleccionaría el segundo posible particionado, es decir, el que involucra al atributo *assets*. Así, del nodo raíz partirán 3 ramas: una para las instancias

donde *assets=low*, otra para *assets=medium*, y por último una rama para las instancias cuyo *assets=high*. Esta partición inicial resulta en la creación de dos nodos hoja (todas las instancias con *assets=low* son de la clase *bad*; mientras que los de *assets=high* son todos de la clase *good*), y un nuevo nodo de decisión que deberá particionarse de nuevo de forma recursiva. El árbol de decisión final se muestra en la Figura 3.

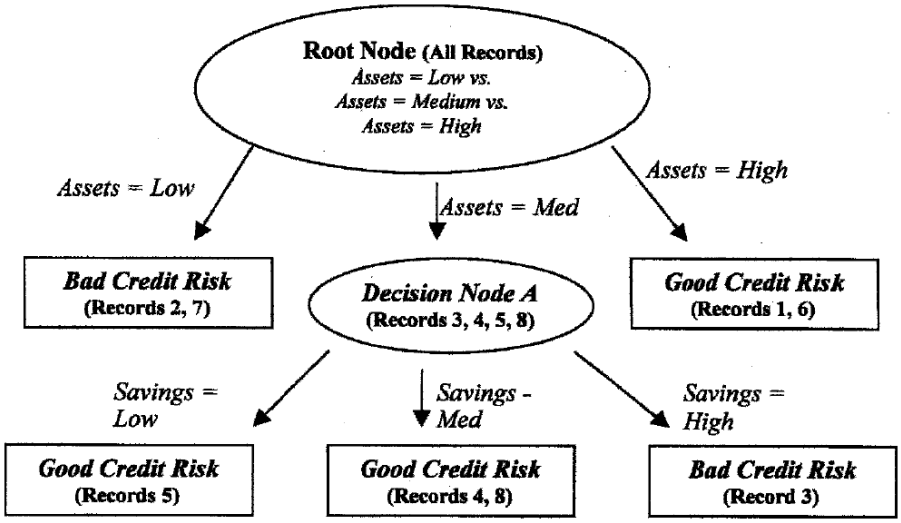

Figura 3. Árbol de decisión construido con C4.5 para el ejemplo.

En los árboles de C4.5 también se propone el uso de podado una vez el árbol se ha construido, aunque no se entrará en mayor detalle en esta lección.

Referencias

[Bre84] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees. Wadsworth Int. Group, 37(15), 237-251.

[Lar14] Larose, D. T., & Larose, C. D. (2014). *Discovering knowledge in data: an introduction to data mining* (Vol. 4). John Wiley & Sons.

[Mai10] Maimon, O., & Rokach, L. (Eds.). (2010). Data mining and knowledge discovery handbook, 2nd edition. *Springer*.

[Qui86] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

[Qui93] Quinlan, J. R. (1993). C4. 5: programs for machine learning. *Morgan Kaufmann*.