0.1. Arboles de decisión. ID3-C4.5

En esta sección se explica el aprendizaje inductivo mediante árboles de decisión. Más concretamente, los asuntos más relevantes de los conocidos métodos denominados ID3 [Quinlan, 1986], su sucesor C4.5 [Quinlan, 1993] y algunas de las ampliaciones realizadas hasta la fecha. En la primera parte de este tema se comentarán los elementos básicos más relacionados con la versión ID3 y en sucesivas secciones se irán analizando las limitaciones y mejoras, entre ellas las incorporadas por C4.5.

Motivación

Aunque no todas las tareas inductivas (p.ej., problemas de regresión lineal) son tratables por el tipo de algoritmos aquí tratados, éstos han resultado ser unos de los más utilizados en el campo del aprendizaje automático. Ello se debe a sus destacables ventajas: aprenden funciones de valores discretos, admiten ejemplos con ruido y obtienen un conjunto de expresiones fácilmente transformable en un conjunto de reglas, que forman opciones alternativas (*disyuntivas*) que cubren el espacio de ejemplos presentados.

Por otro lado, las estrategias que se estudiarán en estos apartados tienen un alto rendimiento y se han aplicado a los más variados dominios (p.ej., diagnóstico médico, predicción de fraudes en créditos bancarios, organización personalizada de eventos en una agenda, etc.). En este sentido, cabe destacar la aplicaciones realizadas en el campo de la minería de datos, tal y como se refleja en la versión comercial denominada C5.0[1].

Objetivos

El objetivo básico, al igual que en otros tantos algoritmos inductivos, es clasificar los ejemplos presentados en sus respectivas clases o categorías, manteniendo un alto grado de predicción sobre los ejemplos no presentados. En concreto, en este caso se trata de obtener un árbol de decisión *simple* que clasifique correctamente los ejemplos presentados y sea capaz de *predecir correctamente* la clases de futuros ejemplos.

En concreto, un árbol de decisión es una estructura que pueden tener los

[1] http://www.rulequest.com/

siguientes tipos de nodos:

- *Nodo interno* (o de *decisión*): Consiste en una pregunta o test relativa al valor de un atributo. De cada nodo interno parten tantas ramas como respuestas haya a la pregunta, que normalmente equivale al número de posibles valores que puede tener el atributo en cuestión.

- *Nodos hoja*: En cada nodo hoja sólo puede haber instancias con un único valor de clase (aunque esto se puede generalizar en presencia de ruido como se tratará en 0.1).

Con este planteamiento, el objetivo podría ser la búsqueda exhaustiva de un árbol lo más *simple* y *predictivo* posible. Sin embargo, este es un problema cuya solución óptima no puede garantizarse para la mayoría de los casos, dado que su complejidad es NP. Por ejemplo, para un problema en el que los ejemplos estén descritos por n atributos booleanos existirían 2^n combinaciones posibles, lo cual supondría 2^{2^n} funciones posibles. Si el número de atributos fuera 5 habría $4{,}3 \times 10^9$ funciones. Frente a este problema, la gran ventaja de ID3 y sus sucesores (como C4.5) es que son capaces de obtener una buena solución que considera ambas cuestiones, aunque el algoritmo no garantice que la solución sea la óptima.

Entradas y salidas

Las entradas son un conjunto de ejemplos descritos mediante una serie de pares *atributo-valor*. El conjunto de atributos es igual para todos los ejemplos presentados y sus valores pueden ser discretos o numéricos. El tratamiento de estos últimos se estudiará más adelante al estudiar las ampliaciones del algoritmo básico (sec. 0.1).

- **Entradas:**
  - A, conjunto de atributos
  - V, conjunto de valores posibles de los atributos
  - C, conjunto de clases
  - E, conjunto de instancias de entrenamiento descritas en términos de A, V y C

- **Salidas:** un árbol de decisión que separa a los ejemplos de acuerdo a las clases a las que pertenecen.

En definitiva, el algoritmo responde a un esquema clásico de clasificación en el que se distinguen dos requisitos para las clases:

- *Clases predefinidas*: Se parte de un problema de *aprendizaje supervisado* en el que el atributo que hace las veces de clase está perfectamente identificado de antemano.

- *Clases discretas*: Se exige que haya un conjunto preciso y discreto de clases que sirven para clasificar claramente todos los ejemplos presentados. El caso de las clases con valores continuos se estudia en el apartado dedicado al algoritmo M5 (sec. 0.2).

Por otro lado, se supone que el número de ejemplos presentados tiene que ser muy superior al de posibles clases. Esto se debe a que el proceso de generalización está basado en un análisis estadísitico que no podría distinguir patrones de interés a partir de coincidencias aleatorias. Un problema real puede requerir cientos o miles de ejemplos de entrenamiento.

La esencia del algoritmo es considerar como estructura base para clasificar los ejemplos un árbol de decisión (AD). Existen toda una familia de algoritmos que utilizan esta estructura entre los que se puede citar [Quinlan, 1986, Kononenko and Simec, 1995, Quinlan, 1993], consistente en un árbol con dos tipos distintos de nodos y una definición recursiva.

Por ejemplo, retomando el problema de personalización de una agenda comentado previamente (sec. **??**), un árbol de decisión sencillo que reflejaría el conocimiento aprendido podría ser el de la figura 1. En el mismo se pueden distinguir dos nodos de decisión que se corresponden con tres propiedades de los ejemplos presentados (si la reunión es o no individual y quién es el asistente a la reunión) y cuatro nodos hojas que reflejan las clases; en este caso, los distintos días de la semana en que tienen lugar las reuniones correspondientes.

TAREA

Si se analiza el método ID3 desde el punto de vista de la búsqueda realizada, éste se puede caracterizar como una búsqueda en un espacio de árboles

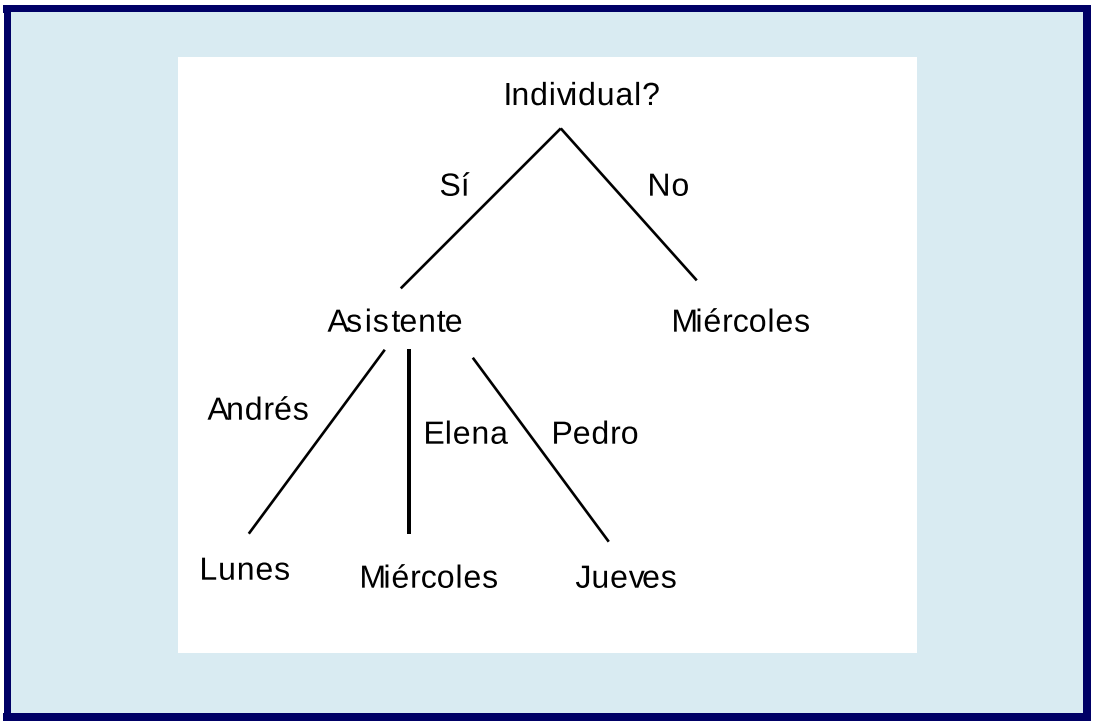

Figura 1: Árbol sencillo de decisión del día de una reunión donde `Miércoles, ...` son las clases, `Individual` y `Asistente` son los atributos relevantes según ID3 para este ejemplo, y `Sí, No, Andrés, ...` son los valores de los atributos considerados.

de decisión. El algoritmo realiza una *búsqueda en escalada* en dicho espacio que comienza con el conjunto vacío y que avanza recursivamente en la elaboración de una hipótesis que consiste en un árbol de decisión que clasifique adecuadamente los ejemplos analizdos.

Para los atributos dados, el espacio de hipótesis de ID3 de todos los árboles de decisión es un *espacio completo* de funciones discretas finitas. Esto es así gracias a que cualquier función discreta finita puede ser representada por algún árbol de decisión.

En concreto, la búsqueda realizada sigue el principio de *"divide y vencerás"*, que en este caso se centra en la división recursiva del árbol en subárboles en los que se busca una mayor homogeneidad en las clases existentes, de tal forma que el proceso se realiza hasta que cada partición contenga ejemplos que pertenezcan a un única clase o hasta que no haya posibilidad de

realizar nuevas particiones.

La búsqueda es un proceso recursivo basado en una decisión, en la que se determina, en cada momento, cuál es el atributo que origina subárboles más homogéneos; entendiendo por homogeneidad la creación de grupos de ejemplos que pertenezcan a una sola clase. El proceso de búsqueda se puede definir como sigue:

- Conjunto de estados: cada estado es un árbol de decisión, en el que los nodos intermedios son preguntas sobre valores de los atributos, las ramas son los distintos valores posibles[2] de dichos atributos y los nodos hoja identifican un único valor de clase.

- Conjunto de operadores: el único operador es "introducir en un nodo la pregunta del atributo correspondiente". Esto origina la expansión de un nodo intermedio del árbol en construcción en varios sucesores, uno por cada posible valor del atributo. Este operador no se puede aplicar cuando no hay ningún caso para analizar en dicho nodo o cuando no quedan más atributos por seleccionar en la rama correspondiente.

- Estado inicial: árbol de decisión vacío.

- Meta: árbol de decisión que separa los ejemplos de entrenamiento dependiendo de su clase.

- Heurística: elegir en cada nodo de decisión aquel atributo que tenga mayor capacidad de discriminación sobre los ejemplos asociados al nodo. Más exactamente, aquél que genere conjuntos disjuntos en los que se maximice la homogenedidad (o minimice la variación) interna con respecto a los valores de clase. Esta heurística tiene la característica también de intentar encontrar el árbol de decisión más pequeño (con menos nodos).

  Para ello se utiliza una función básica de la teoría de la información que mide el grado de desorden, o impureza, de cada una de las particiones generadas. De forma genérica, dicha función determina el *contenido de información* o *entropía* de una fuente de información. Así, si se consideran los mensajes generados por la fuente, la información transmitida

[2]Más adelante se estudiarán los problemas que puede ocasionar que todos los valores de atributo se consideren relevantes (ver sec. 0.1).

por un mensaje dado depende de su probabilidad y puede medirse en bits mediante el logaritmo en base 2 de su probabilidad[3]. Por ejemplo, si hubiera 16 mensajes equiprobables, la información transmitida por cualquiera de ellos sería $-\log_2(\frac{1}{16})$, lo que supone 4 bits de información.

Si la probabilidad de que un ejemplo cualquiera pertenezca a una clase c_k viene dada por la expresión 1, la medida de información *Entropía* realiza una suma ponderada de la información transmitida por cada clase por la proporción de los elementos de la misma, tal y como se concreta en la expresión 2. En esta medida se refleja la variabilidad existente, o el grado de desorden, en el conjunto de ejemplos presentados frente a las diferentes particiones o clases posibles.

$$\frac{n_k}{n} \tag{1}$$

donde n_k es el número de ejemplos de la clase k y n es el número total de ejemplos.

$$\textit{Entropía}(E, A_i, v_j) = -\sum_{k=1}^{C} \frac{n_{ijk}}{n_{ij}} \log_2(\frac{n_{ijk}}{n_{ij}}) \tag{2}$$

donde $\textit{Entropía}(E, A_i, v_j)$ es la entropía de los ejemplos E cuando el atributo A_i tiene el valor v_j, n_{ijk} es el número de ejemplos que tienen el valor v_j del atributo A_i y pertenecen a la clase c_k, n_{ij} es el número de ejemplos que tienen el valor v_j del atributo A_i, y C es el número de clases.

A lo largo del proceso recursivo se elige, en cada nodo de decisión, el atributo que mayor ganancia de información aporte. Para ello, tomando como referencia la anterior medida de entropía, se calcula, para cada atributo, la información esperada requerida si se eligiera dicho atributo como nodo de decisión. En otras palabras, se calcula la reducción en entropía aportada al particionar los ejemplos utilizando los valores del atributo correspondiente. Para ello se realiza una suma ponderada de las entropías resultantes en cada una de las particiones generadas por el atributo. La ponderación considera la proporción de ejemplos de cada rama, tal y como se refeja en la expresión (3).

[3]La unidad de entropía determina la base del logaritmo. En este texto se hablará de bits de información y, por tanto, la base es 2.

$$Ganancia(E, A_i) = Entropía(E) - \sum_{v_j \in \text{Valores}(A_i)} \frac{n_{ij}}{n} \times Entropía(E, A_i, v_j) \tag{3}$$

Dado que se desea elegir el atributo para el que la ganancia es máxima, y, en esta fórmula *Entropía(E)* es una constante con respecto al atributo A_i, es suficiente con elegir aquél atributo que minimice la fórmula de la expresión 4.

$$\sum_{v_j \in \text{Valores}(A_i)} \frac{n_{ij}}{n} \times Entropía(E, A_i, v_j) \tag{4}$$

Para entender la variación de los valores de la entropía pueden considerarse las situaciones en las que sólo hay dos valores de clase. En estos casos, los ejemplos presentados serían los ejemplos y contraejemplos del concepto buscado (tal y como se analizó en el capítulo **??**). Cuando la proporción de ejemplos y contraejemplos sea la misma, entonces la entropía medida por (1) tiene un valor máximo igual a 1, dado que la probabilidad de ambas opciones sería 0.5 y la incertidumbre sobre cuál podría suceder sería máxima. Si sólo hubiera ejemplos de una clase, la entropía tendría un valor mínimo igual a 0, dado que no habría incertidumbre sobre el valor de clase.

Descripción

La estrategia utilizada pertenece a la categoría de los métodos que realizan una búsqueda *voraz* (*greedy*) como escalada. En concreto, se busca un árbol de decisión adecuado entre todos los posibles y no se considera más que una hipótesis (árbol) a lo largo del proceso. Para ello, el método genera dicha hipótesis mediante la selección de un único atributo: aquél que produzca una mejor partición de los ejemplos restantes. Además, no se reconsidera la decisión en pasos sucesivos. No hay, por tanto, ningún tipo de retroceso ni tampoco se consideran las consecuencias de las opciones no elegidas. Por ello, el algoritmo tiene los problemas de los métodos que convergen hacia máximos locales que no sean globalmente óptimos. Más adelante veremos alguna ampliación que resuelve este problema por medio del retroceso (sec. 0.1).

Por otro lado, el algoritmo ID3 aprende un conjunto de disyunciones que se generan cada vez que se divide un nodo. Por ejemplo, en el árbol de

la figura 1, el conocimiento aprendido que determina las reuniones que se celebran los miércoles vendría dado por la disyunción de todas las ramas cuyas hojas terminan en dicho valor de clase (5).

$$(\text{Individual?} = \text{Sí} \wedge \text{Asistente} = \text{Pedro}) \vee (\text{Individual?} = \text{No}) \qquad (5)$$

Otra apreciación de interés es que el método realiza una búsqueda de *lo más general a lo más específico*, añadiendo nuevas particiones (subárboles) a lo largo del proceso hasta que todos las ramas discriminan los distintos valores de clase. Este proceso se realiza mediante un método de *generación y prueba* frente a los métodos ya estudiados *dirigidos por los ejemplos*. Entre estos últimos se pueden citar el procedimiento de eliminación de candidatos (sec. 0.1) o el A^q (sec. **??**), en los que un ejemplo se utiliza como modelo para modificar las hipótesis elegidas. Por el contrarío, en ID3 se consideran primero las distintas formas que podrían tomar la hipótesis y luego se elige entre éstas en función del rendimiento sobre los ejemplos presentados. En este caso, el rendimiento se concreta en una menor entropía sobre las particiones resultantes.

Algoritmo

El algoritmo ID3 realiza un proceso recursivo sobre todas las ramas del árbol generado, tal y como se refleja en la figura 2. Inicialmente, se le llama con el conjunto de ejemplos, el de atributos, y el nodo raíz del árbol de decisión que estará vacío. Como ya se ha comentado, el proceso se realiza hasta que todos los ejemplos de la rama en cuestión pertenecen a una única clase. Sin embargo, esto no siempre es posible. En primer lugar, puede ocurrir que se hayan agotado todos los atributos y, sin embargo, sigan existiendo ejemplos con distintos valores de clase. De otra parte, también puede suceder que, una vez elegido un atributo para un nodo de decisión, no exista ningún ejemplo para una de las ramas generadas por dicho atributo. En esos casos, se etiqueta el nodo hoja con la clase mayoritaria. En los otros casos, se selecciona un atributo de acuerdo a la heurística definida anteriormente. Por cada valor del atributo escogido se crea un nodo sucesor de N etiquetado con el valor correspondiente del atributo, a través de la función `crear-nodo`. Ese nodo se añade al conjunto de nodos sucesores del nodo raíz N y, finalmente, se llama recursivamente a ID3. La llamada recursiva se hace con el conjunto

de ejemplos que tienen ese valor concreto para el atributo escogido A_i, el conjunto de atributos menos el que acabamos de utilizar, y el nodo que acabamos de crear.

Función ID3 (E, A, N): N

E: conjunto de ejemplos
A: conjunto de atributos con sus posibles valores
N: nodo raíz del árbol de decisión

Si $(A := \emptyset)$ ó todos los ejemplos de E pertenecen a la misma clase
Entonces `clase-nodo`(N)=`clase-mayoritaria`(E)
Si no A_i=`mejor-atributo`(A)
    pregunta(N)=A_i
    Para cada valor v de A_i
        H=`crear-nodo`(A_i, v)
        hijos(N)=hijos(N)+H
        $E_i = \{e \in E \mid \texttt{valor}(e, A_i) = v\}$
        ID3$(E_i, A - \{A_i\}, H)$
Devolver N

Figura 2: Procedimiento ID3.

Ejemplo ilustrativo

Supóngase que se dispone de datos sobre los clientes de una tienda en Internet dados por la tabla 0.1. Se dispone de información relativa a los atributos (para abreviar se ha utilizado en algunos una codificación basada en tres valores, 0, 1 y 2): sitio de acceso (se ha codificado como 0, 1 y 2, que pueden ser diferentes proveedores de Internet, por ejemplo), primera cantidad gastada (se podría haber discretizado como 0 menos de 10 euros, 1 entre 10 y 100 y 2 más de 100), zona de la vivienda (por ejemplo podría ser 0 internacional, 1 nacional y 2 local) y última compra (libro o disco). Para cada ejemplo, se especifica la clase a la que pertenece (bueno o malo). Esta

clase puede ser también el resultado de un cómputo, como, por ejemplo, más de 100 euros gastados en total (bueno) o menos de 100 (malo).

Tabla 1: Ejemplos de entrada para ID3.

Ejemplo	Sitio de acceso A_1	1ª cantidad gastada A_2	Vivienda (zona) A_3	Última compra A_4	Clase
1	1	0	2	Libro	Bueno
2	1	0	1	Disco	Malo
3	1	2	0	Libro	Bueno
4	0	2	1	Libro	Bueno
5	1	1	1	Libro	Malo
6	2	2	1	Libro	Malo

Dada esta tabla, ID3 determinaría qué atributo escoger de acuerdo a la heurística relativa a la ganancia de información. Así, por ejemplo, los valores de los diferentes atributos serían (se han calculado con mayor detalle para A_1):

$$I(A_1) = \sum_{j=1}^{nv(A_1)} \frac{n_{ij}}{n} I_{ij} = \sum_{j=1}^{3} \frac{n_{ij}}{6} I_{ij} =$$
$$\frac{n_{10}}{6} I_{10} + \frac{n_{11}}{6} I_{11} + \frac{n_{12}}{6} I_{12} = \frac{1}{6} I_{10} + \frac{4}{6} I_{11} + \frac{1}{6} I_{12}$$
$$I_{10} = -\sum_{k=1}^{2} \frac{n_{10k}}{n_{10}} \log_2 \frac{n_{10k}}{n_{10}} = -\frac{1}{1} \log_2 \frac{1}{1} - \frac{0}{1} \log_2 \frac{0}{1} = 0$$
$$I_{11} = -\sum_{k=1}^{2} \frac{n_{11k}}{n_{11}} \log_2 \frac{n_{11k}}{n_{11}} = -\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} = 1$$
$$I_{12} = -\sum_{k=1}^{2} \frac{n_{12k}}{n_{12}} \log_2 \frac{n_{12k}}{n_{12}} = -\frac{0}{1} \log_2 \frac{0}{1} - \frac{1}{1} \log_2 \frac{1}{1} = 0$$

$$I(A_1) = \frac{1}{6} I_{10} + \frac{4}{6} I_{11} + \frac{1}{6} I_{12} = \frac{1}{6} 0 + \frac{4}{6} 1 + \frac{1}{6} 0 = 0,66$$

$$I(A_2) = \frac{2}{6} I_{20} + \frac{1}{6} I_{21} + \frac{3}{6} I_{22} = \frac{2}{6} 1 + \frac{1}{6} 0 + \frac{3}{6} (-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3}) = 0,79$$

$$I(A_3) = \frac{1}{6} I_{30} + \frac{4}{6} I_{31} + \frac{1}{6} I_{32} = \frac{1}{6} 0 + \frac{4}{6} (-\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4}) + \frac{1}{6} 0 = 0,54$$

$$I(A_4) = \frac{1}{6} I_{4Disco} + \frac{5}{6} I_{4Libro} = \frac{1}{6} 0 + \frac{5}{6} (-\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5}) = 0,81$$

Con esto, ID3 seleccionaría al atributo A_3, generando un árbol de decisión mostrado en la figura 3. Las llamadas recursivas para los valores 0 y 2

generarían sendos nodos hoja, mientras que el valor 1 obligaría a encontrar un atributo para dividir los ejemplos con $A_3 = 1$, dado que hay mezcla de clases en los ejemplos.

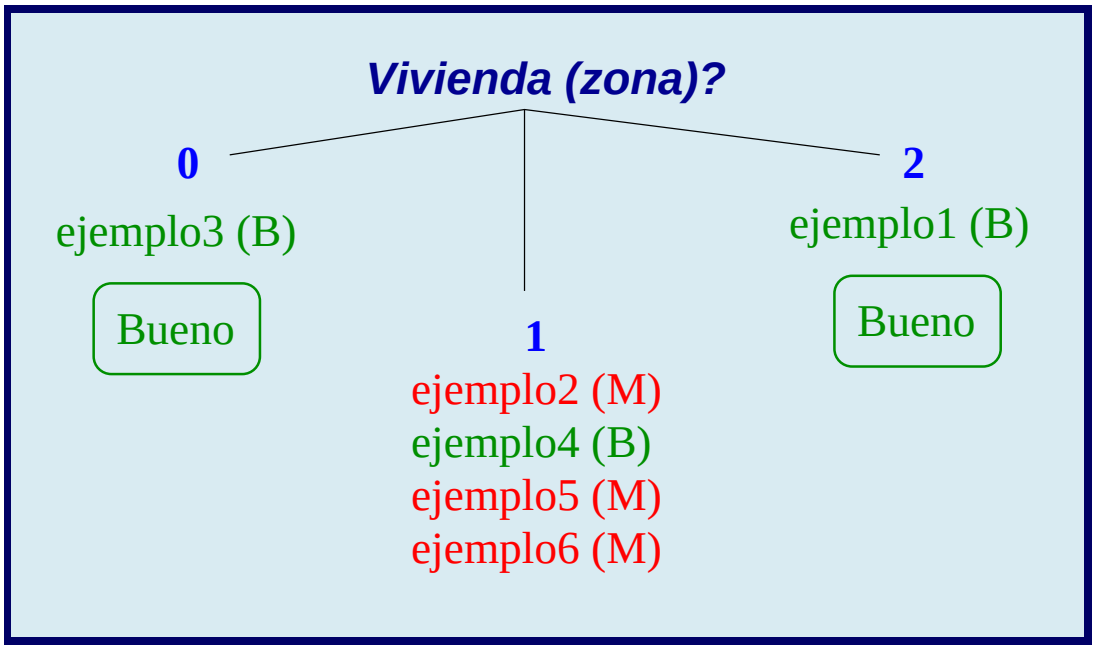

Figura 3: Árbol parcialmente construido por ID3.

Por tanto, se vuelven a calcular las ganancias de información con los atributos restantes (al quitar el A_3) y los ejemplos que tenían $A_3 = 1$. Es decir, la nueva tabla de ejemplos sería la mostrada en la tabla 0.1.

Los resultados del cálculo serían:

$$I(A_1) = \tfrac{1}{4}I_{10} + \tfrac{2}{4}I_{11} + \tfrac{1}{4}I_{12} = \tfrac{1}{4}0 + \tfrac{2}{4}0 + \tfrac{1}{4}0 = 0$$

$$I(A_2) = \tfrac{1}{4}I_{20} + \tfrac{1}{4}I_{21} + \tfrac{2}{4}I_{22} = \tfrac{1}{4}0 + \tfrac{1}{4}0 + \tfrac{2}{4}1 = 0,5$$

$$I(A_4) = \tfrac{1}{4}I_{4Disco} + \tfrac{3}{4}I_{4Libro} = \tfrac{1}{4}0 + \tfrac{3}{4}(-\tfrac{1}{3}\log_2\tfrac{1}{3} - \tfrac{2}{3}\log_2\tfrac{2}{3}) = 0,23$$

Por lo que el árbol de decisión final generado sería el mostrado en la figura 4.

Tabla 2: Ejemplos de entrada para la llamada recursiva a ID3.

Ejemplo	Sitio de acceso A_1	1ª cantidad gastada A_2	Última compra A_4	Clase
2	1	0	Disco	Malo
4	0	2	Libro	Bueno
5	1	1	Libro	Malo
6	2	2	Libro	Malo

La heurística del ID3 es de búsqueda y consiste en preferir árboles que tengan atributos con mayor información más cerca de la raíz, y árboles más cortos. Esto sigue la hipótesis de la *navaja de Occam* (preferir siempre las hipótesis más sencillas que describan los datos). También, por la heurística que emplea, favorece atributos con muchos valores.

Entradas/salidas

- Representación de las entradas: igual que en el resto de los algoritmos inductivos, los ejemplos vienen en función de los valores de los atributos. En la versión simple del ID3, los atributos no pueden tener valores contínuos (en los reales).
- Tratamiento de las entradas: de un sólo paso. El planteamiento es no incremental, por lo que todos los ejemplos se suponen que están disponibles al principio (se comentará una formulación incremental más adelante).
- Preprocesamiento de las entradas: no
- Fuente de las entradas: externa
- Representación de las salidas: es un árbol de decisión (más adelante, se describirá cómo transformarlo a un conjunto de reglas)

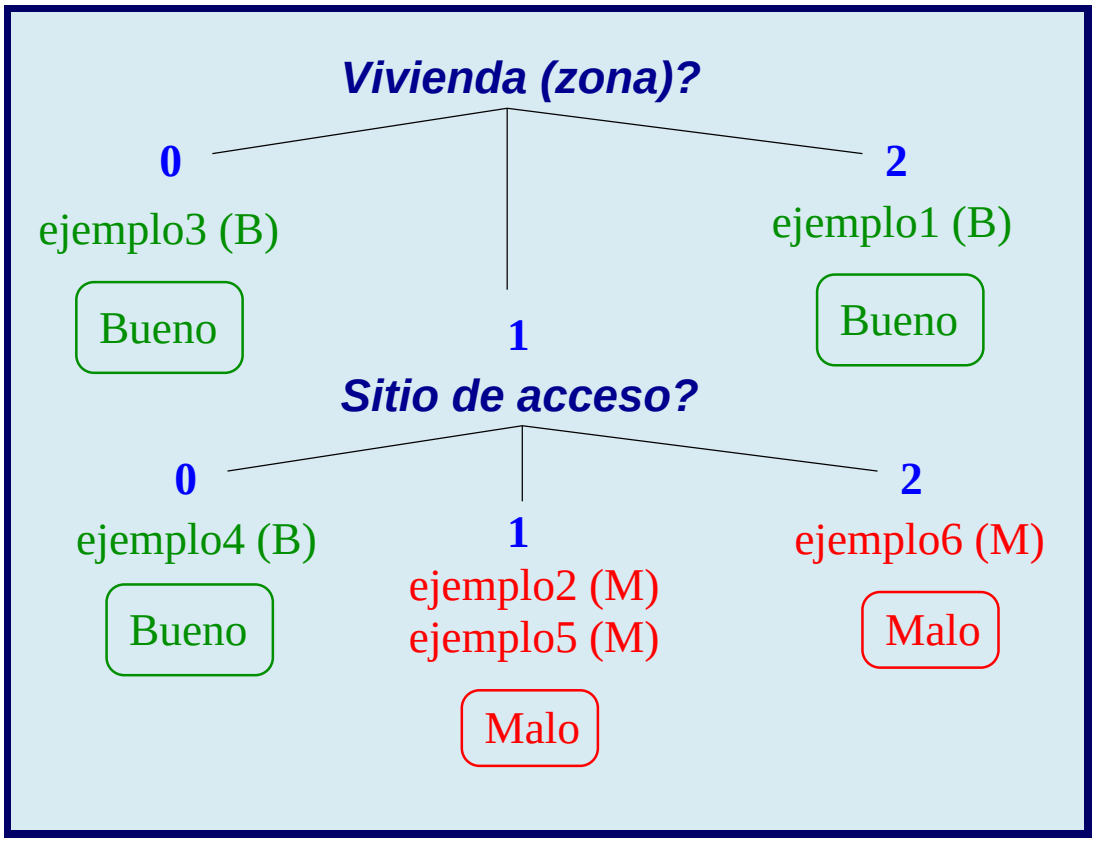

Figura 4: Árbol final resultado de ID3.

Ruido

Este método es capaz de tratar ejemplos con ruido debido a que no es necesario que todos los ejemplos pertenezcan a la misma clase para generar un nodo hoja. Esto permite aplicar el método a problemas en los que pueden aparecer esporádicamente ejemplos mal clasificados, atributos erróneos o valores ausentes de atributo o de clase. Posteriormente, se describirán mejoras sobre este algoritmo que realizan poda del árbol de decisión en función de una serie de criterios relacionados con el ruido.

- En entradas: como se ha comentado, se permite el ruido.

- En estructura: la salida de ID3 es tolerante a fallos, debido a que si se elimina algún nodo del árbol, todavía se podría seguir clasificando las instancias con las ramas restantes.

Complejidad

- Espacio: el espacio ocupado por ID3 es el ocupado por el árbol de decisión. En el peor de los casos, habrá un nodo hoja por cada ejemplo, con un número de nodos intermedios igual al tamaño del espacio de instancias. Esto se produciría, por ejemplo, cuando se le pasara como entrada todos los posibles ejemplos, cada uno con una clase diferente. Como intentaría encontrar una forma de separar todos los ejemplos, esto le llevaría a considerar todos las posibles combinaciones de valores de atributos (tamaño de espacio de instancias). Evidentemente, esto no es un caso real. En el mejor de los casos (por ejemplo, todos los ejemplos pertenecen a la misma clase), habrá un único nodo.
- Tiempo: crece linealmente con el número de ejemplos de entrenamiento y exponencialmente con el número de atributos.

Fiabilidad

Si los ejemplos no tienen ruido, ID3 encuentra el árbol de decisión que describe correctamente a todos los ejemplos. Si hay ruido, entonces depende de lo significativos que sean los ejemplos como en el resto de algoritmos inductivos.

Control de la tarea aprendida

- Crítica/utilidad/valoración: ID3 no realiza una crítica de lo aprendido.
- Utilizacion de lo aprendido: la forma de utilizar lo aprendido consiste en llamar a un procedimiento `clasificar` con el ejemplo que se desee clasificar y el árbol de decisión. La función `clasificar` irá bajando

por el árbol de decisión de acuerdo a las respuestas a las preguntas realizadas en los nodos intermedios. Así, si en el caso del árbol de decisión de la figura 4 se quisiera clasificar una instancia en la que el valor de A_3 fuera 1 y el valor del A_1 fuera 1, entonces, se clasificaría como de la clase Malo.

Dependencia del conocimiento del dominio

No utiliza más conocimiento de dominio que la definición de los atributos y posibles valores de cada atributo.

Variantes

La principal mejora sobre el algoritmo la realizó el propio Quinlan y la escribió en forma de libro con código incluido [Quinlan, 1993]. Entre las mejoras que este algoritmo presentaba sobre ID3 se pueden encontrar las siguientes:

- Tratamiento de los valores contínuos de atributos: cuando se desea saber qué atributo es mejor y se tiene uno (o varios) que son contínuos, por cada uno de los contínuos se crean artificialmente nuevos atributos de la siguiente manera. Por cada atributo contínuo, se ordenan los valores del atributo en los ejemplos de la tabla, y se especifica la clase a la que pertenecen los ejemplos, eliminando los demás atributos. Por ejemplo, la figura 5 muestra una tabla de ejemplos (ahora representados en las columnas) ordenados por el valor del atributo `peso`, así como la clase a la que pertenecen esos ejemplos (sí o no).

peso	30	36	44	60	72	78
dodó	no	no	sí	sí	sí	no

Figura 5: Tabla ordenada de ejemplos por el valor del atributo `peso`.

Como se puede ver, hay dos puntos de corte en el valor de la clase entre los valores 36 y 44, y entre 72 y 78. A continuación, se calculan

los valores medios de cada punto de corte: 40 y 75. Entonces, hay dos formas de crear atributos artificiales:

- dos atributos binarios (valor verdad o falso) nuevos: A_1=(`peso` $<$ 40) y A_2=(`peso` $<$ 75). El primer atributo tendrá valor verdad si la instancia a clasificar tiene el valor de `peso` menor que 45, y lo mismo ocurrirá con el segundo atributo respecto al corte 75.
- si hay k puntos de corte, se puede generar un atributo con $k+1$ valores. En este caso, serían los valores: (`peso` $<$ 40), (40 $<$ `peso` $<$ 75) y (75 $<$ `peso`). Las instancias a clasificar se irían por un valor u otro dependiendo del valor del atributo `peso`.

Normalmente, se escoge la primera opción por lo que se añadirían esos dos atributos al conjunto de posibles atributos a considerar, se evaluarían mediante la misma heurística (ganancia) y si alguno sale vencedor, se crearía un nodo del árbol con esa pregunta.

- Atributos con muchos valores: ID3 prefiere atributos con mayor número de valores (dado que tienden a separar más los ejemplos). Por tanto, una estrategia consiste en desfavorecerles utilizando la medida Razón de ganancia (*GainRatio*, GR):

$$GR(A_i) = \frac{G(A_i)}{-\sum_{j=1}^{nv(A_i)} \frac{n_{ij}}{n} \log_2 \frac{n_{ij}}{n}}$$

  Un problema que tiene esta fórmula es que cuando n_{ij} tiende a n, el denominador se hace 0.

- Conversión del árbol de decisión a un conjunto de reglas. Cualquier árbol de decisión se puede convertir a un conjunto de reglas. El algoritmo `convertir-árbol` toma como entrada un árbol de decisión y genera como salida un conjunto de reglas. Por cada rama del árbol de decisión, se recogen las preguntas sobre el valor de un atributo A_i y el valor v correspondiente (reflejado en el sucesor que se ha seguido por esa rama). La conjunción de estas preguntas $A_i = v$ formará la parte izquierda de la regla que representa a la rama. La etiqueta (clase) del nodo hoja de la rama será la parte derecha de la regla (clasificación).

  Por ejemplo, en el caso del ejemplo anterior, el conjunto de reglas equivalente será el mostrado en la figura 6.

Si Vivienda (zona)=0
Entonces Bueno

Si Vivienda (zona)=1 y Sitio de acceso=0
Entonces Bueno

Si Vivienda (zona)=1 y Sitio de acceso=1
Entonces Malo

Si Vivienda (zona)=1 y Sitio de acceso=2
Entonces Malo

Si Vivienda (zona)=2
Entonces Bueno

Figura 6: Conjunto de reglas equivalente al árbol de decisión de la figura 4.

- Poda. Debido a que puede haber ruido en los ejemplos, se han diseñado varios mecanismos de poda, que se pueden subdividir en pre-poda y post-poda. La pre-poda permite no terminar de separar los ejemplos según su clase en una determinada rama del árbol si se produce una determinada condición o, lo que es lo mismo, no seguir construyendo un árbol de decisión a partir de ese nodo. La primera opción consiste en utilizar el estimador estadístico χ^2 [Quinlan, 1986]. En el caso de que en un nodo no sea significativa (de acuerdo al estimador) la diferencia que existe entre el número de ejemplos que pertenecen a las distintas clases, no se sigue separando los ejemplos; es decir, no se sigue generando un subárbol por debajo del nodo correspondiente. Así, si por ejemplo, en un nodo hay 1000 ejemplos de la clase C_1 y 5 ejemplos de la clase C_2, el estimador devolverá que no es significativa la diferencia. El problema que tiene esta solución es que suele ser muy conservadora y puede hacer que se pare de construir el árbol antes de lo que conviene.

  La segunda solución de pre-poda consiste en generar, cada vez que se expande un nodo del árbol de decisión, la curva de error del árbol ge-

nerado hasta ese momento con el conjunto de test. Si después de haber expandido algún nodo, el error que se produce con el árbol resultante es mayor que el error con el árbol sin expandir ese nodo, entonces el nodo no se expande (se poda).

El segundo tipo de poda es la post-poda que se realiza una vez se ha generado todo el árbol de decisión. También existen varias formas de realizarla de las que se comentarán dos: utilizando el árbol o utilizando las reglas. En el primer caso, se genera el árbol y, a continuación, se analiza recursivamente, y desde las hojas, qué preguntas (nodos) se pueden eliminar sin que se incremente el error de clasificación con el conjunto de *test*. Es decir, supóngase que se tiene un error calculado con todo el árbol de e_a. Ahora, supóngase que se elimina el nodo intermedio situado más a la izquierda del árbol de decisión cuyos sucesores son todos nodos hoja. Esto genera un nuevo árbol de decisión a' y se calcula el error que se comete con este árbol y el conjunto de test. Si $e_a \geq e'_a$, vale la pena eliminar el nodo y todos sus sucesores (que serán nodos hoja). Se continua recursivamente realizando este análisis hasta que no se puede eliminar ningún nodo más sin incrementar el error.

La segunda técnica de post-poda se realiza sobre las reglas generadas a partir del árbol de decisión. Consiste en eliminar progresivamente condiciones de reglas o reglas enteras de forma que no se incremente el error de clasificación. El algoritmo se muestra en la figura 7.

Otras variantes del ID3 han tratado cuestiones como:

- Tratamiento de atributos con costes variables: hasta ahora se ha supuesto que conocer el valor de cada atributo es una constante. Sin embargo, en algunos dominios como el médico [Núñez, 1991] o la robótica [Tan and Schlimmer, 1989], conocer el valor de un atributo (por ejemplo, valor de la presión sanguínea) puede no costar lo mismo que conocer el valor de otro atributo (como, por ejemplo, el resultado de un scanner). En esos casos, se puede utilizar la siguiente función como heurística, en lugar de la ganancia de información:

$$\frac{\textit{Ganancia de información}}{\textit{unidad de coste}^k}$$

  penalizando aquellos atributos con mayor coste.

Función POST-PODA (E, N): R

E: conjunto de ejemplos de test
N: nodo raíz del árbol de decisión
R: conjunto de reglas

$\mathcal{R}$=`convertir-árbol`(N)
error = `error-clasificación`$(\mathcal{R},E)$
Para cada regla $R_i \in \mathcal{R}$
    Para cada condición $p_j \equiv (A_k = v) \in$*condiciones*(R_i)
        $\mathcal{R}'=\mathcal{R}$, eliminando la condición p_j de R_i
        nuevo-error = `error-clasificación`$(\mathcal{R}',E)$
        Si nuevo-error $\leq$ error
        Entonces error = nuevo-error
                $\mathcal{R} = \mathcal{R}'$
    Si *condiciones*(R_i)=
    Entonces $R = R - \{R_i\}$
Devolver R

Figura 7: Procedimiento de post-poda del C4.5.

- Versiones incrementales. El ID3 es un algoritmo en un sólo paso, dado que estudia todos los ejemplos al mismo tiempo. Más adelante, se han planteado versiones incrementales del algoritmo como son ID4 [Schlimmer and Granger, 1986] e ID5 [Utgoff, 1989]. En el caso del ID4, la reconstrucción del árbol no se hace por cada clasificación errónea, sino por degradación de la consistencia. Mantienen estadísticas de la distribución de los valores de los atributos clasificados debajo de cada nodo y si el valor de información de un atributo baja con respecto al de otro, el subárbol se rehace. El algoritmo no asume perfecta consistencia con los datos. Desde el punto de vista de búsqueda tiene un operador de especialización (crecer un árbol) y un operador de generalización (borrar un subárbol). Esto realmente supone un retroceso simulado. En el caso del ID5 en lugar de borrar o crecer subárboles, los reorganiza

si determina que se ha degradado el subárbol. Generalmente, necesita muchos menos ejemplos de entrenamiento.

Conclusiones

Desde el punto de vista de la búsqueda, el ID3 busca hipótesis en un espacio de búsqueda completo. En cada momento, mantiene una hipótesis única, disyunción de conjunciones, frente al espacio de versiones que mantiene un conjunto. Esto le hace perder oportunidades, como, por ejemplo, cómo seleccionar el siguiente ejemplo, pero le hace ser más eficiente. Al utilizar escalada no realiza retroceso por lo que se pueden alcanzar mínimos locales. Sin embargo, si no se tiene ruido y se tienen todos los atributos relevantes, se llega al mínimo. Por otro lado, la heurística se basa en la estadística, lo que lo hace robusto al ruido: si un ejemplo es incorrecto, la estadística suavizará el efecto. Por todas estas características, ha sido uno de las técnicas más utilizadas en las aplicaciones de análisis de datos (*data mining*) o de aprendizaje automático aplicado a tareas tan diversas como predicción de enfermedades, control de robots, o caracterización de clientes en bancos o entidades de seguros.

0.2. Arboles de regresión. M5

Motivación

Los algoritmos que generan clasificadores que se han visto en los capítulos previos suponen que las clases con discretas. Muchas veces lo que se pretende predecir son valores contínuos, por lo que las clases son contínuas. Esto hace que se tengan que modificar los algoritmos convenientemente. Clásicamente, para resolver este problema se ha utilizado la regresión, normalmente lineal. Si llamamos A_{ij} al valor del atributo j para el ejemplo i si es contínuo o la posición que ocupa entre los valores del atributo si el atributo es discreto, se genera un modelo "a priori" definido por la ecuación:

$$c_i = \alpha_0 + \sum_{j=1}^{a} \alpha_j \times A_{ij} \tag{6}$$

donde c_i es la clase para el ejemplo i, α_j son los coeficientes del modelo, y a es el número de atributos del dominio. El objetivo de estos métodos es encontrar los valores de los α_j de forma que se minimice el error (por ejemplo, definido como el error cuadrático medio) de las clasificaciones de los ejemplos de entrenamiento.

Los principales problemas que tienen es que los atributos deben ser numéricos, y que los modelos obtenidos también son numéricos (ecuaciones de rectas en a dimensiones). Breiman y otros [**?**] propusieron en los 80 el sistema CART, parecido al ID3, que generaba en los nodos hoja valores numéricos en lugar de clases discretas. A este tipo de estructuras se las denominó *árboles de regresión*. Más tarde, Quinlan (el autor del ID3) propuso una solución basada en CART con algunas mejoras y es el sistema M5 que se describirá en esta sección. Al igual que CART, M5 genera árboles de decisión similares a los producidos por ID3. Quinlan los denominó *árboles de modelo* (*model trees* en inglés).

Objetivos

El principal objetivo de este método es encontrar un árbol de regresión a partir de instancias de entrenamiento que sea capaz de predecir correctamente el valor numérico de la clase de una instancia futura.

Entradas y salidas

Las entradas son equivalentes a las de los demás sistemas inductivos (clasificadores), salvo que el conjunto de clases, al ser una clase contínua, no es una entrada. Como en los demás sistemas, aparece asociada a cada uno de los ejemplos.

- **Entradas:**
  - A, conjunto de atributos
  - V, conjunto de valores posibles de los atributos
  - E, conjunto de instancias de entrenamiento descritas en términos de A, V y un valor numérico C_i que representa la clase de cada ejemplo $i \in E$

- **Salidas:** un árbol de regresión que separa a los ejemplos de acuerdo a las clases a las que pertenecen.

El algoritmo está preparado para utilizar valores numéricos de atributos, así como la utilización de las técnicas estándar para tratamiento de valores desconocidos tratadas anteriormente en este libro.

Tarea

En términos de búsqueda, M5 realiza tres búsquedas. La primera genera un árbol de regresión a partir de los datos de entrada de forma análoga a ID3 y calcula un modelo lineal (como los modelos clásicos de regresión lineal descritos sucintamente en la introducción) para cada nodo del árbol generado. La segunda búsqueda intenta simplificar el árbol de regresión encontrado en el paso anterior (primera forma de post-poda), eliminando de los modelos lineales de los nodos aquellos atributos que no aumentan el error. La tercera búsqueda tiene por objeto reducir el árbol, manteniendo el mismo poder predictivo (segunda post-poda). La primera búsqueda se puede definir como:

- Conjunto de estados: cada estado es un árbol de regresión, en el que los nodos intermedios son preguntas sobre valores de los atributos y todos los nodos también tienen un modelo asociado de regresión lineal (ecuaciones de hiperplanos con un subconjunto de los a atributos) que modelizan a los ejemplos que caen en esos nodos.

- Conjunto de operadores: el único operador es "introducir la pregunta por un atributo en un nodo". Esto provoca la expansión de un nodo intermedio del árbol de regresión en construcción en varios sucesores, uno por cada posible valor del atributo. Este operador no se puede aplicar cuando en un nodo hay muy pocos ejemplos (menos ejemplos que un umbral ω), o cuando se parecen mucho (su desviación típica es menor que un umbral τ).

- Estado inicial: árbol de regresión vacío.

- Meta: árbol de regresión que separa los ejemplos de entrenamiento dependiendo de su clase, y tiene asociado en cada nodo un modelo lineal que calcula el valor de la clase dependiendo del valor de los atributos considerados.

- Heurística: elegir aquel atributo que minimice la variación interna de los valores de la clase dentro de cada subconjunto. En concreto, se elige aquel atributo que maximice la reducción del error, de acuerdo a la siguiente ecuación:

$$\Delta\text{error} = \sigma_E - \sum_i \frac{\mid E_i \mid}{\mid E \mid} \times \sigma_{E_i}$$

donde

- Δerror es la diferencia de error entre dos árboles de regresión,
- E es el conjunto de ejemplos en el nodo a dividir,
- E_i son los ejemplos con valor i del atributo a considerar, y
- σ_X es la desviación típica de los valores de la clase para los ejemplos en el conjunto X

Pero, dado que el valor σ_E es constante para cada atributo, sólo es necesario elegir aquel atributo que minimice la segunda componente según la ecuación 7:

$$\sum_i \frac{\mid E_i \mid}{\mid E \mid} \times \sigma_{E_i} \tag{7}$$

El último paso de esta búsqueda consiste en crear un modelo lineal para cada nodo del árbol de regresión creado. La segunda búsqueda se realiza utilizando la técnica de escalada y se describe de la siguiente forma:

- Conjunto de estados: cada estado es un árbol de regresión.
- Conjunto de operadores: el operador es "eliminar un atributo de un modelo lineal de un nodo". Esto provoca la simplificación de los modelos lineales de los nodos de un árbol de regresión.
- Estado inicial: árbol de regresión resultante de la primera búsqueda.
- Meta: árbol de regresión simplificado.

- Heurística: para decidir si quitar o no un atributo se debe calcular el error de clasificación esperado antes y después. La fórmula de la estimación del error en posteriores instancias, calcula la media del error residual $e(E, m)$ producido al clasificar con el modelo creado m cada instancia del conjunto E:

$$e(E, m) = \frac{1}{n} \sum_{i \in E} \|c_i - c(m, i)\| \tag{8}$$

  donde

  - $n =\mid E \mid$ es el número de instancias de entrenamiento,
  - c_i es la clase asociada a la instancia i, y
  - $c(m, i)$ es la clasificación que establece el modelo m de la instancia i

  Como esto se subestima el error en posteriores instancias, por lo que se multiplica por:

$$\frac{n + \nu}{n - \nu}$$

  donde ν es el número de atributos en el modelo m, de forma que cuantos más atributos tenga el modelo, más se incrementa el error, por lo que se prefieren modelos con menos atributos. Esto consigue incrementar el error en modelos construidos con muchos parámetros y pocas instancias. La búsqueda se realiza por escalada incluso llegando a eliminar las referencias a todos los atributos en el modelo lineal, dejando sólo la constante α_0.

La tercera búsqueda se describe de la siguiente forma:

- Conjunto de estados: cada estado es un árbol de regresión.
- Conjunto de operadores: el operador es "eliminar un subárbol por debajo de un nodo". Esto provoca la poda de un árbol de regresión.
- Estado inicial: árbol de regresión resultante de la segunda búsqueda.

- Meta: árbol de regresión que separa los ejemplos de entrenamiento dependiendo de su clase, y tiene asociado en cada nodo un modelo lineal que calcula el valor de la clase dependiendo del valor de los atributos considerados. Este árbol es el que se espera que tenga menor error de clasificación.

- Heurística: dado que cada nodo interno del árbol tiene un modelo simplificado lineal y un modelo descrito por su subárbol, se pueden realizar dos clasificaciones en cada nodo del árbol de regresión, y, por tanto, calcular un error de clasificación con las instancias de test: una dada por el modelo lineal y otra por la utilización del subárbol. La heurística dice que habrá que utilizar el operador de eliminar un subárbol en un nodo si el error de clasificación que se produce con el subárbol es mayor que el producido por el modelo lineal del nodo.

Realmente, para ser eficientes, se pueden unir las dos últimas búsquedas en una sóla de forma que se recorra el árbol de regresión una sóla vez, simplificando los modelos lineales de cada nodo al mismo tiempo que se simplifican los nodos en sí.

Descripción

Como se ha planteado anteriormente mediante las búsquedas, el M5 calcula primero un árbol de regresión que minimice en los nodos hojas la variación de los valores de las clases de los ejemplos que caen en ellos. Posteriormente, genera un modelo lineal para cada nodo del árbol. En el siguiente paso, simplifica los modelos lineales de cada nodo, borrando aquellos atributos del modelo lineal que no reduzcan el error de clasificación al eliminarlos. Por último, simplifica el árbol de regresión al borrar subárboles por debajo de los nodos intermedios, cuyo error de clasificación sea mayor que el producido por la clasificación dada por el modelo lineal correspondiente a esos nodos intermedios.

Algoritmo

Una visión general del procedimiento M5 aparece en la tabla 8. Este procedimiento incluye las tres búsquedas descritas, representadas por los algoritmos `construir-árbol-regresión`, `simplificar-modelos-lineales` y

`simplificar-árbol-regresión`. Llama en sucesión a estas funciones después de crear un nodo del árbol de regresión a través de la función `crea-nodo-árbol-regresió`

Procedimiento M5 (E, A)

E: conjunto de ejemplos
A: conjunto de atributos con sus posibles valores

R :=`crea-nodo-árbol-regresión`
R :=`construir-árbol-regresión`(E, A, R)
R :=`simplificar-modelos-lineales`(E, R)
R :=`simplificar-árbol-regresión`(E, R)
Devolver R

Figura 8: Procedimiento M5.

A continuación, se pasa a describir en más detalle cada uno de los procedimientos correspondientes a cada una de las búsquedas. En la tabla 9 aparece la descripción del algoritmo que construye en primera instancia el árbol de regresión. La función `mejor-atributo` elige el atributo a que maximiza la reducción del error esperado según la ecuación 7. A continuación, se estudian cada uno de los sucesores de ese nodo del árbol de regresión, que corresponden a los diferentes valores del atributo a. Cada nodo guarda información respecto al valor concreto i del atributo que trata (valor) y al atributo que utiliza para clasificar por debajo (test). Si la desviación típica en alguno de los sucesores es menor que una determinada cantidad τ o el número de ejemplos en ese sucesor es menor que una determinada cantidad ω, entonces no se sigue generando un subárbol por debajo de ese nodo y se convierte en un nodo hoja. Además, se calcula el modelo lineal correspondiente a los ejemplos que hayan caido en ese nodo.

Si la reducción del error no es mayor que τ, se llama recursivamente al algoritmo para que genere un subárbol por debajo de aquellos sucesores del nodo R que no se hayan considerado nodos hoja. Al finalizar, se calcula un modelo lineal para el nodo R. En atributos-usados se guarda el conjunto de atributos que se han utilizado en nodos del subárbol del nodo R. Esto

es necesario para calcular los modelos lineales de cada nodo, ya que estos modelos solo van a utilizar los atributos utilizados en el subárbol del nodo correspondiente. Esto se hace para establecer una comparación correcta en el procedimiento `simplificar-árbol-regresión` entre el error del modelo lineal de cada nodo y el error de clasificación utilizando el subárbol por debajo de ese nodo.

Función CONSTRUIR-ÁRBOL-REGRESIÓN (E, A, R): R

E: conjunto de ejemplos
A: conjunto de atributos con sus posibles valores
R: árbol de regresión

a :=`mejor-atributo`(E, A)
test(R):=a
Por cada valor i del atributo a
    n:=`crea-nodo-árbol-regresión`
    valor(n):=i
    E_i:=`ejemplos-con-valor`(E, a, i)
    número-ejemplos(n):=$\mid E_i \mid$
    Si número-ejemplos(n)< ω ó $\sigma_{E_i} < \tau$
    Entonces atributos-usados(n):=$\emptyset$
        modelo-lineal(n):=`calcular-modelo-lineal`(E_i, A)
    Si no `construir-árbol-regresión`$(E_i, A - \{a\}, n)$
    hijos(R):=hijos$(R) \cup \{n\}$
    atributos-usados(R):=atributos-usados$(R) \cup$atributos-usados(n)
modelo-lineal(R):=`calcular-modelo-lineal(`E`,atributos-usados(`R`))`
Devolver R

Figura 9: Procedimiento CONSTRUIR-ÁRBOL-REGRESIÓN.

En la tabla 10 se muestra el procedimiento `simplificar-modelos-lineales` cuya función es la de eliminar de los modelos lineales de cada nodo aquellos atributos que no incrementen el error de predicción del modelo lineal. Primero, simplifica los modelos lineales de los nodos sucesores de un determinado

nodo. Después, realiza una búsqueda en escalada, de forma que en cada iteración elimina aquel atributo tal que, si se elimina, el modelo lineal restante minimiza el error de clasificación, siempre y cuando este error sea menor que el del modelo de la anterior iteración.

Por último, el procedimiento `simplificar-árbol-regresión` aparece descrito en la tabla 11. Su función es eliminar el subárbol de regresión por debajo de cada nodo del árbol de regresión siempre y cuando el error de clasificación que produce ese subárbol sea mayor o igual que el producido por el del modelo lineal asociado al nodo.

Ejemplo ilustrativo

Supóngase un banco que desea construir un sistema que prediga el saldo que cada uno de sus clientes tendrán al año siguiente en función de los datos que conoce sobre sus clientes. Para limitar el problema, se van a utilizar los siguientes atributos y valores:

- Sueldo anual en miles de euros, A_1: alto, bajo
- Saldo actual en miles de euros, A_2: alto, medio, bajo
- Saldo año pasado en miles de euros, A_3: alto, medio, bajo
- Casado, A_4: sí, no

Obviamente, la clase es numérica (valor contínuo). Los ejemplos E son los mostrados en la tabla 0.2.

El algoritmo comienza construyendo un árbol de regresión correspondiente a estos ejemplos. Para ello, en cada paso tiene que elegir aquel atributo que maximiza la reducción esperada del error, o lo que es lo mismo, minimiza el error que produce, indicado por la ecuación 7. Para calcular este valor es necesario calcular la desviación típica para cada subconjunto E_i en el que el atributo A_1 divide al conjunto E, σ_{E_i}. Esto se hace calculando primero las medias μ_{E_i}.

$$\mu_{E_{A_1=alto}} = \frac{2300 + 3800 + 600 + 2800}{4} = 2375$$

$$\mu_{E_{A_1=bajo}} = \frac{2500 + 90 + 2000 + 1500}{4} = 1522$$

Función SIMPLIFICAR-MODELOS-LINEALES (E, R): R

E: conjunto de ejemplos
R: árbol de regresión

```
Por cada nodo hijo n ∈ hijos(R)
    n :=simplificar-modelos-lineales(n)
error-reducido:=verdadero
A':=atributos-usados(R)
M:=modelo-lineal(R)
mejor-modelo:=M
min-error:=error-modelo(M, E)
Hasta que error-reducido=falso ó A' = ∅
    Por cada atributo a ∈ A'
        M':=calcular-modelo-lineal(E, A' − {a})
        error:=error-modelo(M', E)
        Si error<=min-error
        Entonces min-error:=error
                 mejor-modelo:=M'
                 atributo-eliminado:=a
    Si M =mejor-modelo
    Entonces error-reducido:=falso
    Si no M :=mejor-modelo
          error-modelo-lineal(R):=min-error
          A' := A' − {atributo-eliminado}
modelo-lineal(R):=M
Devolver R
```

Figura 10: Procedimiento SIMPLIFICAR-MODELOS-LINEALES.

Función SIMPLIFICAR-ÁRBOL-REGRESIÓN (E, R): R

E: conjunto de ejemplos
R: árbol de regresión

Si hijos$(R) \neq \emptyset$
Entonces Por cada nodo hijo $n \in$ hijos(R)
`simplificar-árbol-regresión`(n)
Si error-modelo-lineal(R)<=error-subárbol(R, E)
Entonces hijos(R):=$\emptyset$
Devolver R

Figura 11: Procedimiento SIMPLIFICAR-ÁRBOL-REGRESIÓN.

$$\sigma_{E_{A_1=alto}} = \sqrt{(2300-2375)^2 + (3800-2375)^2 + (600-2375)^2 + (2800-2375)^2} = 2317$$

$$\sigma_{E_{A_1=bajo}} = \sqrt{(2500-1522)^2 + (90-1522)^2 + (2000-1522)^2 + (1500-1522)^2} = 1799$$

El valor del error es, por tanto:

$$\text{error}_{A_1} = \frac{4}{8} \times \sigma_{E_{A_1=alto}} + \frac{4}{8} \times \sigma_{E_{A_1=bajo}} = 0{,}5 \times 2317 + 0{,}5 \times 1799 = 2058$$

Así se calcularían también las variaciones del error producidas por la clasificación utilizando los atributos A_2 a A_4. Los valores obtenidos serían:

$$\text{error}_{A_2} = 689$$

$$\text{error}_{A_3} = 1401$$

$$\text{error}_{A_4} = 2097$$

Según esto, el mejor atributo es el atributo A_2. Por tanto, se crea un nodo raíz en el que se distingue por este atributo y que tiene tres sucesores etiquetados con los valores del atributo: alto, medio y bajo, dando como resultado el árbol mostrado en la figura 12.

Tabla 3: Ejemplos de entrada para el M5.

Ejemplo	Sueldo anual	Sueldo actual	Saldo pasado	Casado	Clase
1	alto	medio	medio	Sí	2300
2	bajo	medio	alto	Sí	2500
3	alto	alto	alto	No	3800
4	alto	bajo	medio	Sí	600
5	bajo	bajo	bajo	No	90
6	bajo	medio	bajo	No	2000
7	alto	alto	medio	Sí	2800
8	bajo	bajo	medio	Sí	1500

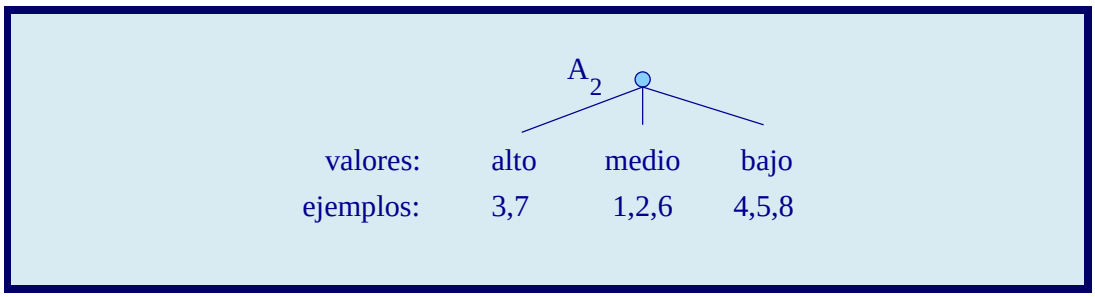

Figura 12: Árbol de regresión formado al elegir el atributo A_2.

Suponiendo que la mínima desviación típica esperada para separar un nodo es $\tau = 500$,[4] y que el mínimo número de nodos para seguir subdividiendo es $\omega = 2$ habría que estudiar qué sucesores hace falta seguir subdividiendo. Normalmente, un valor de $\omega = 2$ es muy bajo porque puede generar nodos hoja con un sólo ejemplo. Esto causa que al calcular la función del error de la ecuación 8, y ser $n = \nu = 1$, el error se haga infinito. Por tanto, se suelen seleccionar valores mínimos mayores que 3.

Al calcular la desviación típica para cada sucesor del nodo raíz se tiene:

$$\sigma_{A_2=alto} = 707$$

[4]Experimentalmente, hay algunos autores que encontraron que valores del 5 % del valor de la varianza en el nodo superior eran buenos.

$$\sigma_{A_2=medio} = 356$$

$$\sigma_{A_2=bajo} = 1010$$

por lo que habrá que seguir subdividiendo sólo los valores *alto* y *bajo*. Si se considera el valor *alto*,

$$\text{error}_{A_1} = 707$$

$$\text{error}_{A_3} = 0$$

$$\text{error}_{A_4} = 0$$

En este caso, la división por cualquiera de los dos atributos A_3 ó A_4 llevaría a un error de 0. Se puede optar por una selección aleatoria. Si, por ejemplo, se selecciona el A_4, daría lugar al árbol de regresión mostrado en la figura 13.

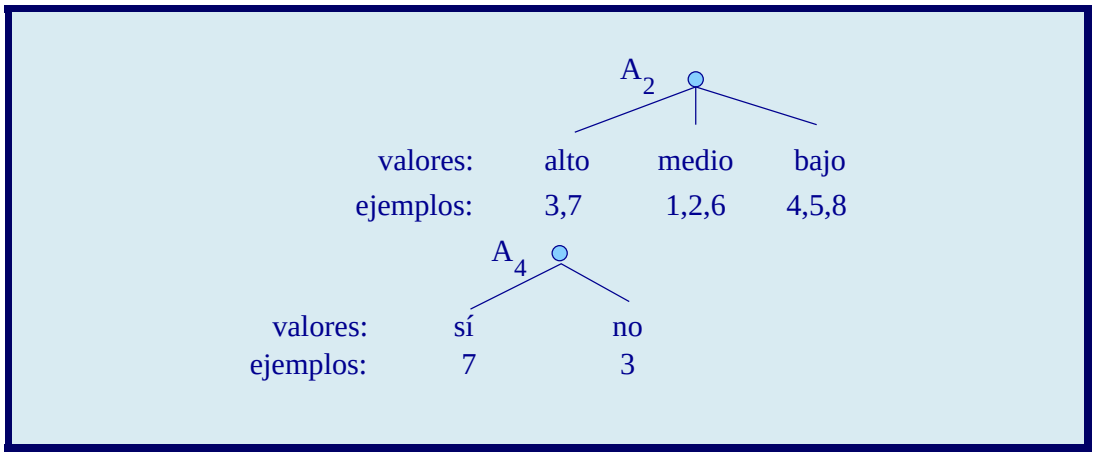

Figura 13: Árbol de regresión formado al elegir el atributo A_4.

En el caso del valor *bajo*,

$$\text{error}_{A_1} = 665$$

$$\text{error}_{A_3} = 424$$

$$\text{error}_{A_4} = 424$$

En este caso, la división sería por cualquiera de los dos atributos A_3 ó A_4. Si, por ejemplo, se selecciona otra vez el A_4, daría lugar al árbol de regresión mostrado en la figura 14.

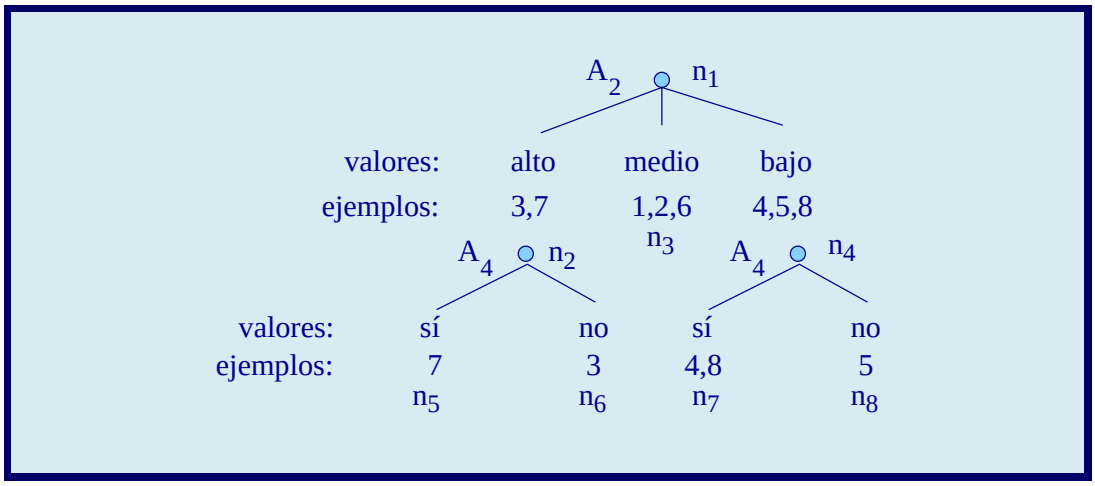

Figura 14: Árbol de regresión formado al elegir el atributo A_4.

Al calcular la desviación típica para cada sucesor se tiene:

$$\sigma_{A_4=\text{SÍ}} = 636$$

$$\sigma_{A_4=\text{no}} = 0$$

y, por tanto, no habría que seguir subdividiendo ninguno por ser menores que el umbral $\tau = 650$. Ahora habría que generar un modelo de regresión lineal para cada nodo hoja del árbol, utilizando la ecuación 6 del principio de este tema. Supóngase que el modelo lineal para cada hoja es el siguiente:

$$\begin{aligned} &n_3\text{: } c = 1800 - 200 \times A_3 - 100 \times A_4 \\ &n_5\text{: } c = 2800 \\ &n_6\text{: } c = 3800 \\ &n_7\text{: } c = 900 \times A_2 - 300 \\ &n_8\text{: } c = 90 \end{aligned} \tag{9}$$

A continuación, habría que calcular los modelos lineales en el resto de nodos. Para ello, se utilizan los atributos que aparezcan en los subárboles por debajo de ellos. Así, en los nodos n_2 y n_3 se utilizaría el A_4, y en el nodo raíz, n_1, el A_2 y el A_4. Así, podrían quedar los modelos lineales como:

$$
\begin{aligned}
&n_1\text{: } c = \\
&n_2\text{: } c = 1800 + 1000 \times A_4 \\
&n_3\text{: } c = 1800 - 200 \times A_3 - 100 \times A_4 \\
&n_4\text{: } c = \\
&n_5\text{: } c = 2800 \\
&n_6\text{: } c = 3800 \\
&n_7\text{: } c = 900 \times A_2 - 300 \\
&n_8\text{: } c = 90
\end{aligned}
\tag{10}
$$

A partir de este momento, se comienza la segunda búsqueda (función `simplificar-modelos-lineales`), que elimina en los modelos lineales construidos aquellos atributos tales que, al eliminarlos, no se incremente el error residual estimado de clasificación (ecuación 8). Por ello, lo primero que hay que calcular es dicho error para el modelo (árbol de regresión construido con los pasos anteriores). En este caso, el árbol de regresión clasificaría correctamente todos los ejemplos de entrenamiento, con lo que el error sería 0 y no se podría mejorar aunque se eliminara algún atributo de cualquier modelo lineal.

El último paso del algoritmo consiste en simplificar (podar) el árbol seleccionando en cada nodo intermedio si interesa quedarse con el subárbol por debajo de él o con el modelo lineal del nodo. Para ello, se calcula el error producido por el árbol total al considerar dicho árbol y el error eliminando dicho árbol y utilizando en ese nodo el modelo lineal. Así, por ejemplo, en el nodo n_2 si se elimina el subárbol por debajo de él, el error que se comete también es 0, por lo que conviene eliminar el subárbol, quedando el árbol que aparece en la figura 15.

En el caso del nodo n_4, sin embargo, el error producido es mayor que 0, por lo que no se puede eliminar dicho subárbol, quedando definitivamente el árbol de regresión de la figura 15.

Bias

En el caso del M5, al igual que en el caso de muchos de los sistemas de aprendizaje descritos en este libro, sus *bias* son de búsqueda. Éstas vienen determinadas por las heurísticas utilizadas a la hora de elegir qué atributo es el mejor por medio de la ecuación 7, y el cómputo del error producido por un modelo (lineal o basado en un árbol). En el caso del M5 también se produce

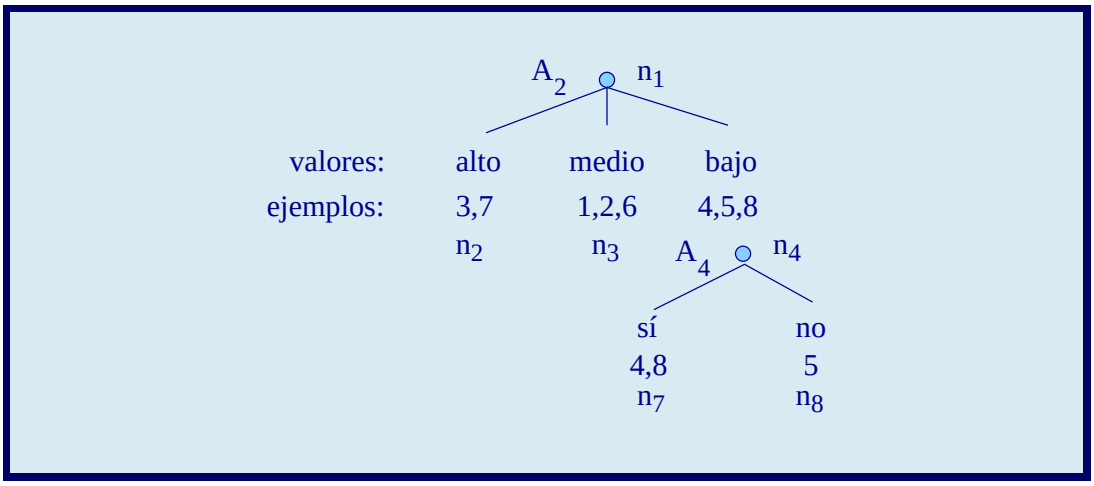

Figura 15: Árbol de regresión después de eliminar el subárbol por debajo del nodo n_2.

un *bias* de lenguaje, al representar modelos lineales solamente. Podría haber escogido para cada nodo la utilización de modelos no lineales, como, por ejemplo, redes de neuronas, descritas en el capítulo **??** de este libro.

Entradas/salidas

- Representación de las entradas: los ejemplos vienen en función de los valores de los atributos
- Tratamiento de las entradas: en un sólo paso
- Preprocesamiento de las entradas: no
- Fuente de las entradas: externa
- Representación de las salidas: es un árbol de regresión, que tiene en cada nodo intermedio una pregunta sobre el valor de un atributo y en cada nodo hoja un modelo lineal para calcular la clase numérica

Ruido

- En entradas: acepta ejemplos con ruido, que queda suavizado por el análisis de los errores. Los modelos lineales son aproximaciones al valor de la clase, con lo que suavizan el efecto del ruido en la clasificación de los ejemplos.

- En estructura: la salida del M5 es tolerante a fallos, debido a que dispone de dos tipos de modelos en cada nodo intermedio (modelo lineal y modelo basado en árbol), por lo que si se elimina algún nodo o algún modelo de algún nodo, se podría seguir clasificando.

Complejidad y fiabilidad

En relación con la complejidad:

- Espacio: el espacio ocupado por la salida del M5 es el ocupado por el árbol de regresión. En el peor de los casos, en cada rama se utilizarían todos los atributos, aunque en la práctica es inviable que esto suceda si se eligen convenientemente los parámetros τ y ω. Si los atributos en media tuvieran i valores, el número de nodos del árbol resultante sería:

$$\text{nodos} = \sum_{j=0}^{|A|} i^j$$

  En el mejor de los casos, el árbol de regresión quedaría reducido al nodo raíz con un modelo lineal que representaría a los ejemplos de entrenamiento.

- Tiempo: el tiempo por cada ejemplo depende del número de atributos, $a =\mid A \mid$, y del número de ejemplos n. Así, en el primer proceso en el caso peor tendría que generar un árbol de regresión con todos los atributos en cada rama y generar un modelo lineal por cada nodo del árbol. Después tiene que recorrer todos los modelos lineales para simplificarlos (lineal en el número de nodos del árbol, lo que es exponencial respecto al número de atributos), y volver a recorrer todos los nodos para eliminar o no el subárbol por debajo de cada nodo (lineal en el número de nodos).

El sistema de clasificación descrito por el árbol de regresión será fiable, al igual que los demás métodos inductivos, en tanto los ejemplos de entrenamiento sean representativos de los futuros y se elijan convenientemente los valores de ν y ω.

Control de la tarea aprendida

- Crítica/utilidad/valoración: la forma de realizar una valoración interna de lo aprendido por M5 es a través del cálculo del error, que se puede realizar con un conjunto externo de instancias.

- Utilizacion de lo aprendido: la forma de utilizar lo aprendido consiste en llamar a un procedimiento `clasificar` con el ejemplo que se desee clasificar e y el árbol de regresión R devuelto por el proceso de entrenamiento por la función `m5`. La función `clasificar`, aparece descrita por el algoritmo de la tabla 16. Si el nodo en el que se va a clasificar el ejemplo es hoja, se calcula la clase con el modelo lineal asociado a ese nodo. Si no, primero se encuentra el nodo sucesor que tiene el mismo valor del atributo por el que se subdivide el nodo que el ejemplo a clasificar e (la función `valor atributo` devuelve verdadero cuando encuentra el nodo). Una vez encontrado, se llama recursivamente a la función `clasificar`, que devolverá la clase.

  Este es el valor que debería devolver el proceso normal de clasificación. Sin embargo, se puede mejorar el poder de clasificación si se suaviza esa clasificación teniendo en cuenta las clasificaciones basadas en los modelos lineales de cada nodo intermedio. Eso es lo que logra la fórmula matemática que aparece en el algoritmo que tiene en cuenta la clasificación basada en el subárbol de regresión c_a y la clasificación basada en el modelo lineal del nodo c_m. Esa fórmula utiliza un parámetro k que es una entrada al algoritmo.[5] Este proceso de suavizado se ha encontrado más útil cuando los nodos hoja se han construido con pocos ejemplos y las clasificaciones dadas por esos modelos en los nodos hoja no estaban muy de acuerdo con las clasificaciones de los modelos lineales más arriba en el árbol de regresión.

[5]Se han encontrado experimentalmente como buenos valores los valores cercanos a $k = 15$, pese a que no se ha estudiado en profundidad la sensibilidad a ese parámetro.

Función CLASIFICAR (e, R, k): clase

---

e: ejemplo a clasificar
R: árbol de regresión
k: parámetro

Si hijos(R) = $\emptyset$
Entonces Devolver `clase-modelo-lineal(modelo-lineal(`R`),`e`)`
Si no a:=test(R)
    Por cada nodo hijo $n \in$ hijos(R)
        Si `valor-atributo`(e, a,valor(n))
        Entonces c_a:=`clasificar`(n)
                `sucesor`:=n
    ne:=número-ejemplos(sucesor)
    c_m:=`clase-modelo-lineal(modelo-lineal(sucesor),`e`)`
    Devolver $\frac{ne \times c_a + k \times c_m}{ne + k}$

Figura 16: Procedimiento CLASIFICAR.

Dependencia del conocimiento del dominio

No utiliza más conocimiento de dominio que la definición de los atributos y posibles valores de cada atributo.

Variantes

Como ya se mencionó en la introducción, M5 está basado en CART. Existen varias diferencias entre los dos sistemas entre las que cabe citar: CART utiliza la heurística de elegir aquel atributo que maximiza la reducción esperada en varianza o en desviación absoluta, en lugar de reducir el error esperado; y los nodos hoja en el caso de CART son valores concretos, mientras que en el caso del M5 son modelos lineales que aproximan la clase por un hiperplano.

Algunos autores han hecho estudios con variaciones del algoritmo, como, por ejemplo, el que aparece en el libro [**?**].

Otros sistemas generan también clasificadores numéricos: redes de neuronas (capítulo **??**), BACON [**?**] que realiza búsquedas en el espacio de las fórmulas matemáticas que mejor se ajusten a un conjunto de observaciones dadas, o MARS [**?**] que utiliza un esquema similar al M5 con "splines" para suavizar la predicción.

Conclusiones

En este capítulo se ha tratado un método que permite construir modelos de clasificación cuando las clases a predecir son numéricas. Está basado en la ccombinación de la construcción de árboles de decisión y la regresión lineal multivariante clásica. Las dos principales ventajas que ofrece sobre otras técnicas, es que maneja clases numéricas y que la salida es más entendible que otras técnicas numéricas (como las redes de neuronas), pero, aún así, los modelos de regresión lineal no son tan entendibles como las reglas o árboles generados por ID3. Aún así, constituyen una herramienta muy útil para proporcionar salidas con clases contínuas entendibles.

Bibliografía

[Kononenko and Simec, 1995] I. Kononenko and E. Simec. Induction of decision trees using relieff, 1995.

[Núñez, 1991] Marlon Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, 6:231–250, 1991.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[Schlimmer and Granger, 1986] Jeffery C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[Tan and Schlimmer, 1989] M. Tan and J.C. Schlimmer. Cost-sensitive concept learning of sensor use in approach recognition. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY (USA), 1989.

[Utgoff, 1989] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, November 1989.