Capítulo 7

Metodología STAR
Generación progresiva de reglas

1. Introducción

Bajo el nombre de STAR se recogen un conjunto de técnicas de aprendizaje inductivo incremental basadas en la utilización de expresiones lógicas en forma normal disyuntiva. Las ideas aquí expuestas se deben principalmente a Michalski y sus colaboradores.

Como modelo de representación del conocimiento, los algoritmos de la familia STAR emplean expresiones lógicas en forma normal disyuntiva. Este modelo de representación es más expresivo que el empleado por los algoritmos de la familia TDIDT, que emplean árboles de decisión o clasificación.

En los algoritmos de tipo STAR, una estrella [*star*] denota los posibles descriptores de los ejemplos de entrenamiento. Utilicemos un sencillo ejemplo para comprenderlo mejor:

Color	Dureza	Raya	Mineral
Gris	Elevada	Blanca	Diamante
Incoloro	Elevada	Blanca	Diamante
Gris	Baja	Blanca	Yeso
Blanco	Baja	Blanca	Yeso

De esta pequeña tabla se derivan con facilidad las siguientes reglas:

$(Dureza=Elevada) \Rightarrow (Mineral=Diamante)$ $(Color=Incoloro) \Rightarrow (Mineral=Diamante)$ $(Color=Gris) \wedge (Dureza=Elevada) \Rightarrow (Mineral=Diamante)$
$(Dureza=Baja) \Rightarrow (Mineral=Yeso)$ $(Color=Blanco) \Rightarrow (Mineral=Yeso)$ $(Color=Gris) \wedge (Dureza=Baja) \Rightarrow (Mineral=Yeso)$

La lista de reglas para cada una de las posibles clasificaciones del mineral se denomina estrella [*star*], que no es más que un conjunto de expresiones lógicas que describen a todos los ejemplos de una clase concreta.

Las reglas mostradas en la tabla anterior recogen toda la información contenida en los casos de entrenamiento. Sin embargo, no todas esas reglas son necesarias para describir el conjunto de entrenamiento. El siguiente conjunto de reglas sería suficiente:

$(Dureza=Elevada) \Rightarrow (Mineral=Diamante)$
$(Color=Blanco) \Rightarrow (Mineral=Yeso)$ $(Color=Gris) \wedge (Dureza=Baja) \Rightarrow (Mineral=Yeso)$

El conjunto reducido de descriptores recogido en la tabla anterior se denomina estrella reducida [*reduced star*]. Una estrella reducida no es más que una estrella en la cual se han eliminado todos los elementos redundantes en la descripción del conjunto de entrenamiento.

Todos los algoritmos de la familia STAR pueden encontrarse en algún momento con algún ejemplo concreto que satisfaga varias de las reglas obtenidas o que incluso no satisfaga ninguna. En el primer caso, el ejemplo se asocia a la clase más común de aquéllas recogidas en las reglas satisfechas. En el segundo, el ejemplo se suele asignar por defecto a la clase más común en el conjunto de entrenamiento (para minimizar la probabilidad de error en la clasificación).

Algo más de nomenclatura

Una variable asociada a un valor o a una disyunción de valores se denomina *selector* y se denota ***(Variable∈{conjunto de valores})***. Un selector cubre a un ejemplo si la expresión que denota es cierta para dicho ejemplo.

Un complejo [*complex*] es una conjunción de selectores. Un complejo vacío (conjunción de cero selectores) cubre a todos los ejemplos posibles.

Finalmente, los complejos se agrupan en *covers* utilizando conjunciones (*AND*) o disyunciones (*OR*) para representar la parte *IF* de las reglas. Un *cover* vacío (disyunción de cero complejos) no cubre a ningún ejemplo. Cada cover se asocia a una etiqueta que representa la clase más general de ejemplos de entrenamiento que cubre.

2. Algoritmos STAR

En esta sección se expondrán los algoritmos más significativos de la familia STAR, comenzando por los propuestos por Michalski. Suponiendo la existencia de alguna técnica que nos permita generar una estrella reducida, Michalski propuso seguir los pasos que se enumeran a continuación para conseguir de forma automática la descripción de un concepto (que puede ser una clase en un problema típico de clasificación) a partir de un conjunto de ejemplos positivos y un conjunto de ejemplos negativos:

1. Elegir aleatoriamente un caso del conjunto de ejemplos positivos (semilla).

2. Generar una estrella reducida (lista de complejos con un máximo de *m* elementos) para la semilla y el conjunto de ejemplos negativos. Emplear reglas de generalización y heurísticas.

3. Encontrar la descripción más adecuada de la estrella reducida obtenida (de acuerdo a algún criterio de preferencia).

4. Si la descripción cubre todos los ejemplos positivos, ir a 6.

5. Reducir el conjunto de ejemplos positivos a aquéllos no cubiertos por la descripción y repetir el proceso completo desde el paso 1.

6. La disyunción de todas las descripciones generadas es una descripción consistente y completa del concepto. Como paso final, reformúlense las reglas para obtener expresiones más simples.

Ejemplo

Para mostrar el funcionamiento de los algoritmos de Michalski utilizaremos de nuevo el ejemplo de clasificación de minerales empleado en la introducción. Fijaremos en 3 el máximo número de complejos de una estrella (*m=3*), un parámetro usado en la búsqueda dirigida [*beam search*] que se realiza para la obtención de la estrella reducida.

☞ *Mineral = Diamante*

Elijamos el primer ejemplo positivo de diamante: *(Gris,Elevada,Blanca)⇒(Diamante)*. Obtenemos la estrella reducida que cubra el ejemplo positivo y no los ejemplos negativos:

(Dureza=Elevada)
(Dureza=Elevada) AND (Color=Gris)
(Raya=Blanca) AND (Dureza=Elevada) AND (Color=Gris)

De las descripciones obtenidas escogemos la mejor: *(Dureza=Elevada)*. Como esta descripción cubre a todos los ejemplos positivos de diamante, hemos obtenido:

(Dureza=Elevada) ⇒ (Mineral=Diamante)

☞ *Mineral = Yeso*

Escojamos el último caso como ejemplo de yeso: *(Blanco,Baja,Blanca)⇒(Yeso)*. Una estrella reducida que cubre el ejemplo positivo pero no los negativos es:

(Color=Blanco)
(Dureza=Baja)
(Raya=Blanca) AND (Color=Blanco)

Escogemos *(Color=Blanco)* como mejor descripción. Sin embargo, este descriptor no cubre todos los ejemplos positivos de yeso, por lo que eliminamos el ejemplo empleado y repetimos el proceso. Sólo nos queda un ejemplo positivo de yeso: *(Gris,Baja,Blanca)⇒(Yeso)*. La estrella reducida correspondiente es:

(Dureza=Baja) AND (Color=Gris)
(Dureza=Baja)
(Raya=Blanca) AND (Dureza=Baja)

Si el descriptor de mayor preferencia es *(Dureza=Baja) AND (Color=Gris)*, entonces hemos cubierto el conjunto de ejemplos positivos de yeso con las reglas:

(Color=Blanco) ⇒ (Mineral=Yeso)
(Dureza=Baja) AND (Color=Gris) ⇒ (Mineral=Yeso).

☞ Reglas de clasificación:

(Dureza=Elevada) ⇒ (Mineral=Diamante)
(Color=Blanco) ⇒ (Mineral=Yeso)
(Dureza=Baja) AND (Color=Gris) ⇒ (Mineral=Yeso).

Como se muestra en el ejemplo, la generación progresiva de reglas realizada por los algoritmos STAR requiere la utilización de una técnica que nos permita obtener la estrella reducida y el uso de un criterio de preferencia con el que podamos seleccionar el descriptor más prometedor de los incluidos en una estrella reducida.

Criterio de preferencia

Como criterio de preferencia se utiliza un función heurística LEF [*Lexicographical Evaluation Function*] que incluye una serie de criterios elementales como la simplicidad de la descripción o la maximización del número de ejemplos positivos cubiertos. La función LEF es una serie ordenada de pares criterio-tolerancia.

A la hora de seleccionar la descripción más prometedora de un conjunto dado, se evalúa el criterio C_i para cada una de las descripciones obtenidas. Se escoge la mejor descripción según este criterio y todas aquellas cuya evaluación queda dentro del margen establecido por la tolerancia T_i. El proceso se repite hasta que nos quedamos con una sola descripción o se acaba la serie de pares criterio-tolerancia.

Construcción de la estrella reducida

La mayor parte de los programas que utilizan la metodología STAR dividen el conjunto de casos de entrenamiento en ejemplos positivos y negativos, seleccionan un ejemplo positivo al azar (la semilla) y forman una estrella reducida que cubra ese ejemplo.

La estrella reducida es un conjunto de descripciones alternativas que cubren a la semilla (y posiblemente a otros casos positivos) sin cubrir a ningún ejemplo negativo.

Para obtener este conjunto de selectores (la estrella reducida) se han de realizar transformaciones de generalización y de especialización. Por ejemplo, si tenemos una descripción $P(X)$ y un ejemplo negativo que verifica $P(X) \wedge Q(X)$, se especializa la descripción para no cubrir el caso negativo: $P(X) \setminus Q(X)$.

Como resulta evidente, se necesita algún tipo de heurística que nos guíe a la hora de realizar transformaciones de generalización y de especialización para evitar la explosión combinatoria que supondría la exploración de todas las descripciones posibles de un concepto (es decir, todos los selectores posibles).

Generalmente se opta por realizar una búsqueda dirigida [*beam search*], una variante de la búsqueda primero el mejor en la que sólo se siguen explorando los *m* caminos más prometedores (como si se tratase de n ascensiones de colinas en paralelo). Esta estrategia de búsqueda heurística también se ha utilizado con éxito en sistemas para el reconocimiento del habla como SPHINX (un sistema de reconocimiento continuo independiente del orador).

2.1 INDUCE (Michalski 1983)

En el algoritmo INDUCE, se extrae de la semilla un conjunto de selectores S ordenados según la función LEF. Los selectores de este conjunto se especializan añadiendo otros selectores de menor preferencia para formar nuevos selectores que se añaden al conjunto (cuyo tamaño siempre está limitado por el parámetro *m*).

Se comprueba la consistencia (no cubrir ejemplos negativos) y completitud (cubrir todos los ejemplos positivos) de los selectores del conjunto. Aquellos selectores que satisfagan ambas propiedades se pasan a una lista R de soluciones y aquéllos que sean consistentes pero incompletos se colocan en otra lista C. Los selectores de S se especializan mientras que los de C han de generalizarse. El proceso se detiene cuando se consigue un número determinado de elementos en la lista de soluciones.

Esta técnica implica considerar todos los ejemplos positivos en cada paso del algoritmo. Aunque se realice una enumeración extensiva de combinaciones de selectores, el parámetro *m* de la búsqueda dirigida nos permite limitar el espacio de búsqueda.

Ejemplo

Utilicemos el mismo ejemplo que antes para obtener la estrella reducida del diamante. Inicialmente, la lista ordenada de selectores es:

❶ *(Raya=Blanca)*
❷ *(Color=Gris)*
❸ *(Dureza=Elevada)*

Se especializan los selectores anteriores y se ordenan de acuerdo con la función LEF:

❹	*(Raya=Blanca) AND (Color=Gris)*	Inconsistente e incompleto
❺	*(Raya=Blanca) AND (Dureza=Elevada)*	Consistente y completo
❻	*(Color=Gris) AND (Dureza=Elevada)*	Consistente pero incompleto

El selector ❺ pasa a la lista de soluciones y el ❻ a la lista de selectores consistentes incompletos. El selector que queda ha de especializarse aún más:

❼ *(Raya=Blanca) AND (Color=Gris) AND (Dureza=Elevada)*

Al ser consistente pero incompleto, lo pasamos a la lista C. Los selectores de esta lista (❻ y ❼) se pueden generalizar para intentar obtener nuevos selectores consistentes y completos.

Finalmente, la lista de soluciones incluye los selectores ❸ y ❺:

(Dureza=Elevada)
(Raya=Blanca) AND (Dureza=Elevada)

2.2 AQ

La serie de programas AQ utiliza una técnica diferente para la generación de la estrella reducida a la empleada en INDUCE. En ellos se escoge un único ejemplo positivo o semilla en cada ocasión. El algoritmo general se aplica a cada una de las clases definidas y es el siguiente:

Mientras no se cubran todos los ejemplos positivos

Seleccionar uno de ellos aleatoriamente (semilla)

Generar una estrella que cubra la semilla y no cubra ejemplos negativos

Seleccionar el mejor complejo de la estrella generada de acuerdo a algún criterio

Añadir el complejo escogido a lo que ya teníamos

Se puede comenzar sin ninguna información o con algo ya aprendido anteriormente. Los pasos necesarios para generar la estrella que cubra la semilla pero no los ejemplos negativos se recogen a continuación:

Mientras la estrella parcial cubra ejemplos negativos

Seleccionar un ejemplo negativo cubierto por la estrella parcial

Generar la estrella que cubra a la semilla pero no al ejemplo negativo (conjunto de selectores que cubren la semilla pero no el ejemplo negativo)

Especializar los complejos de la estrella parcial para excluir el ejemplo negativo (intersección de la estrella parcial con la estrella generada en el paso anterior)

Si el número de complejos de la estrella excede un umbral predefinido [maxstar], podarla eliminando sus peores complejos

---

Ejemplo

Usemos una vez más el ejemplo del diamante. Escogemos el primer caso positivo de la tabla: *(Gris,Elevada,Blanca)⇒(Diamante)*.

Obtengamos la estrella parcial que cubra al caso pero no a los ejemplos negativos. Para ello, escogemos el ejemplo negativo *(Blanco,Baja,Blanca)⇒(Yeso)*. La estrella que cubre a la semilla y excluye a este ejemplo es:

(Dureza ∈ {Elevada})
(Color ∈ {Gris, Incoloro})

Como esta estrella incluye al otro ejemplo negativo, repetimos el proceso. La estrella que cubre a la semilla y excluye al ejemplo *(Gris,Baja,Blanca)⇒(Yeso)* es la siguiente:

(Dureza ∈ {Elevada})

Se realiza la intersección de la estrella parcial que teníamos con la obtenida:

(Dureza ∈ {Elevada}) AND (Dureza ∈ {Elevada}) = (Dureza ∈ {Elevada})
(Color ∈ {Gris, Incoloro}) AND (Dureza ∈ {Elevada})

Hemos obtenido la estrella parcial que cubre a la semilla y excluye a los ejemplos negativos. Seleccionamos el mejor complejo de esta estrella y lo añadimos a la solución:

(Dureza ∈ {Elevada})

Como esta simple estrella cubre todos los ejemplos positivos, hemos concluido la generación de la estrella que sirve para distinguir los diamantes del yeso.

Tal como se muestra en el ejemplo anterior, AQ es un algoritmo iterativo en el cual se genera una regla de decisión en cada iteración. Para cada clase concreta se construye una disyunción de complejos [*cover*]. En cada paso del algoritmo se añade un complejo a la descripción de la clase analizada y se eliminan del conjunto de entrenamiento los ejemplos cubiertos por el nuevo complejo. Se prosigue este proceso hasta que no queden ejemplos por cubrir de la clase analizada y no queden más clases por describir.

Los programas AQ buscan la descripción más general que permita identificar la pertenencia a una clase de forma incremental. Los selectores obtenidos suelen empezar siendo muy generales y resultan más y más específicos conforme se van incluyendo ejemplos positivos.

Las heurísticas empleadas por los algoritmos AQ dependen de la implementación concreta aunque lo más normal es seleccionar en cada momento el complejo más simple que maximiza el número de ejemplos positivos cubiertos (sin que incluya ejemplos negativos, como es obvio). Dado que cuanto más simple sea un complejo más ejemplos cubre, lo que podría parecer un problema multiobjetivo (- complejos + ejemplos) no lo es.

La estrategia usada para reducir la estrella parcial durante la generación de un complejo es, normalmente, maximizar la suma de ejemplos positivos cubiertos y ejemplos negativos excluidos. En caso de empate, se opta siempre por el complejo más simple, es decir, el complejo con menos selectores.

La semilla se escoge aleatoriamente del conjunto de entrenamiento. Los ejemplos negativos se suelen escoger empezando por los más cercanos a la semilla. Si se encuentra una contradicción (la semilla tiene los mismos valores que un ejemplo negativo para todos sus atributos) se descarta directamente el ejemplo negativo, ya que el complejo no puede especializarse para excluirlo sin dejar de incluir la semilla.

AQ11 ya superó a los expertos humanos a comienzos de los años 80. AQ15 es capaz de generar nuevos atributos no presentes en los datos de entrada. AQ15-GA utiliza algoritmos genéticos para generar subconjuntos de atributos que permitan simplificar las reglas obtenidas. AQ17-DCI [*Data-Driven constructive Induction*] escoge nuevos atributos utilizando una función de calidad. AQ17-FCLS [*Flexible Concept Learning*] utiliza representaciones simbólicas y numéricas para elaborar la descripción de un concepto. AQ17-HCI [*Hypothesis-driven Constructive Induction*] genera nuevos atributos analizando las hipótesis obtenidas en cada iteración. DLG (Webb 1991) comienza con un clasificador específico que va generalizando conforme va examinando ejemplos positivos (al revés que AQ).

2.3 CN2

El algoritmo CN2 [Clark & Niblett] trata de combinar la eficiencia y el manejo de información con ruido que permite la familia de algoritmos TDIDT con la flexibilidad de la familia AQ en su estrategia de búsqueda de reglas IF-THEN.

Este algoritmo produce un conjunto ordenado de reglas IF-THEN (denominado "lista de decisión" [*decision list*]) empleando técnicas heurísticas basadas en una estimación del ruido presente en los datos para acotar el espacio de búsqueda. De esta forma se consigue un conjunto de reglas que, si bien puede no clasificar bien todos los datos del conjunto de entrenamiento, sí suele comportarse bien a la hora de clasificar nuevos datos.

El proceso descendente de construcción de árboles de decisión de los algoritmos TDIDT le permiten manejar con facilidad información con ruido. Generalmente se realiza una post-poda del árbol de decisión una vez que éste ha sido completamente generado. No obstante, el sobreaprendizaje también podría evitarse deteniendo el crecimiento del árbol cuando la ramificación de un nodo terminal (esto es, una hoja) no aporte una ganancia de información significante (vg: utilizando medidas de entropía).

Por su parte, los algoritmos AQ realiza una búsqueda dirigida [*beam search*] del mejor complejo [*complex*] consistente con el conjunto de entrenamiento. Esta estrategia podría considerarse como varias búsquedas en paralelo de tipo ascensión de colinas [*hill-climbing*].

En CN2 se mantiene la estrategia de búsqueda dirigida, aunque se elimina la dependencia de AQ respecto a los casos concretos de entrenamiento. Además, el espacio de búsqueda se amplía para permitir la inclusión de reglas que no se ajusten perfectamente al conjunto de entrenamiento, lo que se consigue examinando todas las especializaciones posibles de cada complejo (del mismo modo que los algoritmos TDIDT analizan todas las posibilidades de ramificación en cada nodo del árbol de decisión). Al realizar una búsqueda descendente en el espacio de posibles complejos (análoga a la construcción de árboles de decisión), también se pueden aplicar técnicas de poda características de los algoritmos TDIDT para detener el proceso de especialización de los complejos cuando no se obtienen mejoras estadísticamente significativas.

Se ha de destacar que CN2, a diferencia de los algoritmos AQ, produce una lista ordenada de reglas IF-THEN. Esto dificulta la comprensibilidad del conjunto de reglas obtenido (la interpretación de una regla aislada depende de las que la preceden en la lista) aunque elimina la necesidad de utilizar un mecanismo adicional de resolución de conflictos (necesario en los algoritmos de la familia AQ). De cualquier modo, CN2 puede también conseguir un conjunto desordenado de reglas IF-THEN como AQ si se cambia la función de evaluación.

Las reglas obtenidas por CN2 son de la forma `IF <complejo> THEN <clase>`, donde `<complejo>` es una conjunción de tests sobre los atributos. La última regla de la lista ordenada es una regla que asigna por defecto la clase más común del conjunto de entrenamiento a cualquier ejemplo que llegue hasta ella. Rivest acuñó el término "lista de decisión" [*decision list*] para referirse a esta representación ordenada de reglas.

A la hora de realizar la clasificación, simplemente hay que recorrer ordenadamente la lista de decisión hasta encontrar una regla cuya condición satisfaga el ejemplo. Si no se satisface ninguna regla, el ejemplo se asigna a la clase más común en el conjunto de entrenamiento.

CN2 es un algoritmo iterativo. En cada iteración se busca un complejo que cubra un gran número de ejemplos de una clase determinada C y sólo algunos de otras clases, de forma que éste sirva para después hacer una predicción fiable de la clase de los ejemplos a los que cubre. Mediante una función de evaluación se selecciona el complejo más prometedor y se añade la regla *IF <complex> THEN C* al final de la lista de decisión. El proceso de búsqueda se repite hasta que no consigamos más complejos lo suficientemente satisfactorios.

La búsqueda de complejos adecuados se realiza manteniendo un conjunto finito o estrella de los mejores complejos encontrados hasta el momento (lo que poda el espacio de búsqueda considerablemente). Sólo se examinan las especializaciones de los complejos incluidos en la estrella (búsqueda dirigida del espacio de búsqueda). En principio, las especializaciones posibles consisten en añadir un selector a una conjunción o eliminar un término en una disyunción. Tras evaluar todas las especializaciones posibles de los complejos incluidos en la estrella, nos quedamos sólo con los mejores de ellos (eliminando todos los demás).

Para tratar con valores desconocidos, CN2 utiliza en método más simple posible: los valores desconocidos se reemplazan con los más comunes en el conjunto de entrenamiento.

Finalmente, CN2 requiere el uso de dos heurísticas durante el proceso de aprendizaje, que nos permitan caracterizar los complejos que se van obteniendo:

❖ *Complejos de calidad* (*capacidad de predicción*)

En primer lugar se ha de evaluar la calidad de los complejos para determinar cuáles han de ser incluidos en la estrella de mejores complejos encontrados hasta el momento. Se ha de encontrar el conjunto E de ejemplos cubiertos por el complejo (aquéllos que satisfacen todos sus selectores) y la distribución de probabilidad P de ejemplos de E para cada clase. La versión original de CN2 utiliza una medida de entropía que ha de minimizarse:

$$H = -\sum_i p_i \log p_i$$

Esta función se escogió en vez de un simple porcentaje de clasificación **max(P)** porque la medida de entropía considera peor $P=(0.7,0.1,0.1,0.1)$ que $P=(0.7,0.3,0,0)$, lo que es deseable al ser más fácil especializar la segunda distribución para identificar una clase concreta. Nótese que las distribuciones resultantes de la eliminación de los ejemplos de la clase más común son $P=(0,0.33,0.33,0.33)$ y $P=(0,1,0,0)$, respectivamente.

NOTA: Si sustituimos la medida de entropía por la diferencia entre el número de ejemplos positivos y el número de ejemplos negativos cubiertos por el complejo, podemos obtener un conjunto desordenado de reglas tal como en AQ.

Una versión mejorada del algoritmo CN2 sustituye la entropía por una estimación del error. El problema de la entropía como medida de calidad (y también del porcentaje de clasificación, tal como puede usarse, por ejemplo, en AQ15) es que tiende a escoger reglas muy específicas que cubren sólo a unos pocos ejemplos de entrenamiento, ya que es más fácil encontrar reglas muy precisas en conjuntos de ejemplos muy específicos. En el caso extremo, cada regla cubrirá a un solo ejemplo del conjunto de entrenamiento (¡¡con un 100% de precisión y 0 de entropía!!). Esto, además, es poco deseable: las reglas respaldadas sólo por algunos ejemplos son poco fiables (y menos aún cuando existe ruido en los datos de entrada). Por lo tanto, la entropía no refleja suficientemente bien la capacidad de predicción de una regla.

En el siguiente apartado se verá que CN2 sólo considera complejos estadísticamente significativos al generar reglas, pero esta restricción adicional (que elimina casos extremos como los descritos en el párrafo anterior) no soluciona el problema. CN2 aún tenderá a escoger las reglas más específicas frente a reglas más generales y fiables pero aparentemente menos precisas.

Por suerte, existe una medida que permite estimar la precisión esperada directamente, la estimación laplaciana del error esperado:

$$LA = \frac{n_c + 1}{n_{tot} + k}$$

donde k es el número de clases del problema, n_c es el número de ejemplos de la clase más común c cubiertos por la regla y n_{tot} el número total de ejemplos cubiertos por la regla.

Por último, se ha de comprobar que la precisión estimada de la regla seleccionada sea mejor que la regla por defecto (resultante de asignar directamente a todos los ejemplos la clase más común entre ellos).

❖ *Complejos significativos (fiabilidad de la predicción)*

Una segunda heurística resulta imprescindible para poder evaluar si un complejo es significativo o no. Es decir, se pretende reflejar la correlación existente entre los valores de los atributos y las distintas clases del problema. CN2 compara la distribución observada de ejemplos que satisfacen un complejo con la distribución esperada si el complejo escogiese ejemplos aleatoriamente. CN2 intenta distinguir las diferencias debidas al azar de las resultantes de la correlación entre atributos y clases utilizando

$$LR = 2\sum_i f_i \log \frac{f_i}{e_i}$$

donde F es la distribución observada de los ejemplos que satisfacen un complejo y E la distribución esperada (los ejemplos distribuidos entre las distintas clases con la misma probabilidad que en el conjunto completo de entrenamiento). La medida *LR* [*likelihood ratio*] mide en cierto modo la distancia entre las dos distribuciones. Cuanto menor sea *LR*, más probable es la intervención del azar.

Curiosamente, el comportamiento de *LR* varía según utilicemos la entropía *H* o la estimación laplaciana del error *LA* como medida de la calidad de las reglas obtenidas. Utilizando la entropía, cuando incrementamos el umbral de *LR* se obtienen reglas más generales en el algoritmo inductivo de aprendizaje. Sin embargo, al utilizar *LA*, el umbral de *LR* sólo sirve de criterio de parada para el algoritmo de búsqueda (no afecta al conjunto de reglas escogidas, ya que *LA* tiende por sí mismo a escoger las reglas más generales posibles).

CN2

Entrada *E: Conjunto de ejemplos de entrenamiento.*

Salida *L: Lista de decisión, inicialmente vacía*

Algoritmo

Mientras se encuentre un buen complejo estadísticamente significativo

*Eliminar de E los ejemplos cubiertos por el complejo seleccionado **

Añadir al final de L la regla "IF complejo THEN C", donde C es la clase más común en los ejemplos cubiertos por el complejo.

Búsqueda dirigida de complejos [reglas]

Mientras la estrella actual no esté vacía

Especializar todos los complejos de la estrella actual

Recordar el mejor complejo encontrado de los estadísticamente significativos

Reducir la estrella a su tamaño máximo preestablecido

** NOTA: Para poder conseguir un conjunto de reglas en el que no exista un orden implícito, se han de eliminar de E sólo aquéllos ejemplos cubiertos por el complejo seleccionado que correspondan a la clase C. En ese caso, el proceso completo ha de repetirse independientemente para cada una de las clases.*

3. Resultados y limitaciones

El principal cuello de botella a la hora de construir sistemas expertos se encuentra en la fase de adquisición del conocimiento. Por este motivo, los sistemas que aprenden a partir de ejemplos (vg. las familias de algoritmos TDIDT y STAR) han tenido bastante éxito en este ámbito.

Utilicemos como muestra la conocida tabla de jugar o no al golf usada por Quinlan como ejemplo para la construcción de árboles de decisión. Los algoritmos greedy de construcción de árboles de decisión de la familia TDIDT (por ejemplo, el ID3) obtendrían las siguientes reglas:

$$OUTLOOK = overcast \Rightarrow PLAY = yes$$

$$OUTLOOK = rain \wedge WINDY = false \Rightarrow PLAY = yes$$
$$OUTLOOK = rain \wedge WINDY = true \Rightarrow PLAY = no$$

$$OUTLOOK = sunny \wedge HUMIDITY = normal \Rightarrow PLAY = yes$$
$$OUTLOOK = sunny \wedge HUMIDITY = high \Rightarrow PLAY = no$$

Si empleamos un algoritmo de tipo STAR, conseguimos un resultado bastante parecido:

$$OUTLOOK = sunny \wedge TEMPERATURE = hot \Rightarrow PLAY = no$$
$$OUTLOOK = rain \wedge WINDY = true \Rightarrow PLAY = no$$
$$OUTLOOK = sunny \wedge HUMIDITY = high \Rightarrow PLAY = no$$

$$OUTLOOK = overcast \Rightarrow PLAY = yes$$
$$OUTLOOK = rain \wedge WINDY = false \Rightarrow PLAY = yes$$
$$OUTLOOK = sunny \wedge HUMIDITY = normal \Rightarrow PLAY = yes$$

En este caso, los resultados obtenidos por una y otra técnica son prácticamente equivalentes. La única diferencia se encuentra en la primera regla generada por el algoritmo STAR, que es redundante (la tercera abarca todos los casos cubiertos por ella).

Por otro lado, los algoritmos STAR de Michalski carecen de un modelo para guiar la poda de las descripciones obtenidas y la terminación del proceso de búsqueda, por lo que el método en sí es incompleto, además de ser bastante ineficiente (para una tabla de votaciones del Congreso norteamericano de 435 tuplas y 16 atributos el clasificador AQ tarda hasta 10 veces más tiempo en construirse que el basado en árboles de decisión, mientras que para una tabla de enfermedades de la soja de 683 tuplas y 36 atributos consume ya unas 60 veces más tiempo de CPU). En cuanto a su eficiencia, el algoritmo CN2 es similar a los de la familia AQ.

Dependencia del orden de presentación

Al igual que sucede con los espacios de versiones de Mitchell, los resultados obtenidos con los algoritmos STAR de Michalski dependen en gran medida del orden en que se consideren los ejemplos (y, como es lógico, de la función heurística LEF empleada). Esto no sucede en la construcción de árboles de decisión y también se evita en el algoritmo CN2 (donde no se utiliza una semilla en el proceso de construcción de las estrellas reducidas).

No obstante, hay que resaltar que los algoritmos STAR de Michalski están pensados para realizar un aprendizaje incremental (algo no permitido en la construcción de árboles de decisión) y las técnicas de aprendizaje incremental suelen ser dependientes del orden de llegada de los casos de entrenamiento.

Manejo de información con ruido

Los algoritmos de aprendizaje básicos presuponen que no existe ruido en los datos de entrada e intentar alcanzar una descripción conceptual perfecta de los datos de entrada. Esto suele ser contraproducente en problemas reales, donde se necesitan métodos capaces de manejar información con ruido y mecanismos que eviten el sobreaprendizaje [*overfitting*].

Algoritmos como el ID3 (de la familia TDIDT) pueden modificarse fácilmente para evitar el sobreaprendizaje, ya que se trata de métodos descendentes que parten de conceptos generales que se van especificando conforme descendemos en el árbol de decisión. Las técnicas de poda (como en ASSISTANT o C4.5) han demostrado ser muy útiles en este sentido.

Por otro lado, los algoritmos tipo AQ son más difíciles de modificar debido a que los resultados que obtienen dependen del orden de presentación de los ejemplos de entrenamiento. Existen implementaciones (AQ11 y AQ15) que permiten manejar información con ruido utilizando técnicas de pre- y post-procesamiento sin modificar el algoritmo AQ en sí. De hecho, el objetivo del algoritmo CN2 es modificar el algoritmo AQ de forma que la dependencia del orden de presentación sea eliminada.

Manejo de atributos continuos

Además, por si todo lo ya comentado fuese poco, los algoritmos de tipo STAR no pueden manejar directamente valores continuos. Sólo pueden tratar valores discretos. Como sucede en muchas otras técnicas de la IA simbólica, los atributos continuos han de ser discretizados previamente utilizando técnicas auxiliares (vg: clustering).

4. Bibliografía

Peter Clark & Tim Niblett
"The CN2 Induction Algorithm"
Machine Learning Journal, 3 (4), pp. 261-283, Netherlands: Kluwer, 1989

En este artículo se expone el algoritmo CN2 y se compara con ID3 y AQ. Los autores desarrollaron el algoritmo CN2 con el objetivo de conseguir un método eficiente de aprendizaje inductivo para la obtención de reglas de producción en presencia de ruido, conocimiento incompleto, etc.

Peter Clark & Robin Boswell
"Rule Induction with CN2: Some Recent Improvements"
Machine Learning, EWSL-91 Proceedings, pp. 151-163, Berlin: Springer-Verlag, 1991

Una versión revisada del algoritmo CN2 en la cual se propone una función de evaluación alternativa basada en una estimación laplaciana del error y la generación de una lista no ordenada de reglas. En esta ocasión, CN2 es comparado con el algoritmo C4.5 de Quinlan.

Elaine Rich & Kevin Knight
"Inteligencia Artificial"
McGraw-Hill Interamericana de España, 1994 [2ª edición]

Uno de los libros de IA más populares. Le dedica un capítulo entero a técnicas de aprendizaje. Aunque no habla de la metodología STAR de Michalski, sí trata los espacios de versiones de Mitchell (que siguen ideas similares).

Sabrina Sestito & Tharam S. Dillon
"Automated Knowledge Acquisition"
Prentice-Hall Series in Computer Science and Engineering, Australia, 1994

Este libro de ML [Machine Learning] le dedica uno de sus capítulos a la metodología STAR ideada por Michalski y sus colaboradores, el tercero para ser más concretos.