2.4 PROGRAMACIÓN LÓGICA INDUCTIVA

Francisco José Ribadas Pena, Santiago Fernández Lanza

Modelos de Razonamiento y Aprendizaje
5º Informática
ribadas@uvigo.es, sflanza@uvigo.es

18 de febrero de 2013

2.4.1 INTRODUCCIÓN

Aprendizaje de programas lógicos a partir de ejemplos.

- Se aprenden reglas de lógica de primer orden
- Mayor poder expresivo que árboles decisión

OBJETIVO: Aprender las reglas que definan un predicado P a partir de un conocimiento de partida Q y un conjunto de ejemplos E.

Dados:

1. Q**:** Conocimiento base del dominio (en forma de cláusulas Prolog)
2. E**:** Conj. de ejemplos positivos y negativos del predicado P

Se pretende construir un conjunto de clausulas Prolog que definan el predicado objetivo P de forma que "cubran" todos los ejemplos positivos y ninguno negativo.

Debe verificarse:

1. Para cada ejemplo positivo $p_i \in E$: $P \cup Q \models p_i$
2. Para cada ejemplo negativo $n_i \in E$: $P \cup Q \not\models n_i$

El conocimiento de base Q puede darse en forma extensional (lista de hechos) o intensional (reglas Prolog).

Ejemplos:

Se aprendren las reglas para *primo(X,Y)*, a partir de conocimiento (lista de hechos) sobre las relaciones *progenitor(X,Y)* y *hermano(X,Y)*

```
primo(X,Y) :- hermano(A, B), progenitor(A,X), progenitor(B,Y)
```

Se aprendren las reglas para *ancestro(X,Y)*, a partir de conocimiento sobre las relación *progenitor(X,Y)*

```
ancestro(X,Y) :- progenitor(X,Y)
ancestro(X,Y) :- progenitor(Z,Y), ancestro(X,Z)
```

$\Rightarrow$ **es posible recursividad**

TÉCNICAS Dos técnicas para aprender programas lógicos a partir de ejemplos

1. Invirtiendo la regla de inferencia de **resolución**
   - Si es posible deducir los ejemplos E a partir del conocimiento base Q y la hipótesis que buscamos P (o una aproximación P')

$$P' \cup Q \models E$$

   - Se podrá demostrar E usando resolución (resolución es completa) a partir de P' y Q.
   - **IDEA:** Generar una nueva hipótesis para P a partir de la demostración de uno de los ejemplos de entrenamiento
   - Una vez realizada la demostración de un ejemplo:
     - Aplicar regla de resolución "al revés" (de abajo a arriba)
     - Generalizar la demostración
       - Mediante sustitución de algunas constantes por variables
   - Necesidad de un proceso de búsqueda que encuentre la hipótesis "más conveniente" P' entre todas las que se puedan generar a partir de la demostración de cada ejemplo.
   - Esta aproximación se conoce como aprendizaje basado en las demostraciones *(Explanation Based Learning) (EBL)*
     - Forma parte de las técnicas de aprendizaje deductivo
2. Aplicando algoritmos de aprendizaje de reglas
   - Aproximación similar al aprendizaje de árboles de decisión
   - Son algoritmos "descendentes"
     - Parten de reglas generales
     - Van especializándolas para generar reglas más específicas

2.4.2 Algoritmo FOIL

Uno de los algoritmos más simples de PLI. (Quilan, 90)

Extensión directa de algoritmos de aprendizaje de reglas por cobertura de ejemplos al caso de reglas Prolog.

Implementa un algoritmo descendente, que genera el conjunto de reglas que mejor "cubren" los ejemplos.

- **IDEA:** Ir generando nuevas reglas, cada vez más específicas, hasta que se cubran todos los ejemplos positivos
  1. Comenzar con la regla más general para el conjunto de ejemplos actual
  2. Añadir restricciones a esa regla para que se eliminen ejemplos negativos
  3. Se para de especializar la regla si $\begin{cases} \text{no quedan ejemplos negativos} \\ \text{no se puede especializar más} \end{cases}$
  4. Se eliminan los ejemplos positivos que cubre la nueva regla y se repite el proceso (si quedan ejemplos positivos)

Objetivo: cubrir todos los ejemplos positivos con el menor número de reglas posibles, sin cubrir ninguno negativo

Simplificaciones en FOIL

- Se define un predicado meta $P(x_1, ..., x_n)$, con n argumentos
- No admite funciones en los predicados (evita problemas de no terminación)
- Conocimiento de base Q (predicados auxiliares) en forma extensional (conjuntos de hechos)

Funcionamiento

1. En cada iteración comienza siempre con el predicado meta P más general, inicializado a:

$$P(X1, ..., X_n) :- \qquad \text{(verdadero)}$$

2. Genera un conjunto de reglas Prolog que lo especializan, hasta que cubre todos los ejemplos positivos
   - Se añaden nuevos predicados en la parte derecha de la regla.
   - En la parte derecha sólo se consideran predicados de la forma:

$$\left.\begin{array}{l} Q_i(v_1, ...V_j) \\ NOT(Q_i(v_1, ...V_j)) \end{array}\right\} \begin{array}{l} \text{Siendo } Q_i \text{ un predicado de base o el propio } P \text{ (recursividad).} \\ \\ \text{Donde al menos una variable } V_s \text{ ya existía en la regla que se especializa} \end{array}$$

$$\left.\begin{array}{l} X_k = X_l \\ NOT(X_k = X_l) \end{array}\right\} \begin{array}{l} \text{Donde } X_k \text{ y } X_l \text{ son variables presentes en la regla que se especializa} \end{array}$$

Algoritmo

ENTRADA: predicado objetivo P
ejemplos (positivos y negativos)
predicados de apoyo Q
SALIDA: lista de reglas para P

Postitivos := ejemplos positivos
MIENTRAS haya *Positivos*
    /* Aprender una nueva regla*/
    nuevaRegla := ReglaMasGeneral()
    Negativos := ejemplos negativos
    MIENTRAS haya *Negativos*
        /* Especializar nuevaRegla */
        restricciones := GenerarRestricciones(*nuevaRegla*)
        mejorRestricción := MejorHeurística(*nuevaRegla*,*restricciones*)
        añadir *mejorRestricción* al lado derecho de *nuevaRegla*
        eliminar de *Negativos* los ejemplos que *nuevaRegla* rechaza
    FIN MIENTRAS
    añadir *nuevaRegla* a la lista de reglas
    eliminar de *Positivos* los ejemplos que cubre *nuevaRegla*
FIN MIENTRAS
devolver lista de reglas

Funciones

- ReglaMasGeneral(): inicializa la regla como se comentó anteriormente
- GenerarRestriciones(regla): aplica las reglas de especialización anteriores
  Generar todos los posibles predicados que se pueden incluir en la parte derecha de la regla

- MejorHeurística(regla, restricciones): evalua una a una las reglas resultantes despues de añadir a la regla uno de los predicados de restricción.
  - Se pueden definir muchas heurísticas
  - La más sencilla: (α es la restricción que se añade a la regla)

    $$h(regla, \alpha) = \frac{\text{n}^{\text{o}} \text{ ejemplos positivos cubiertos al añadir la restricción } \alpha}{\text{n}^{\text{o}} \text{ total de ejemplos (+ y -) que cubre la regla con } \alpha}$$

    Se elije la restricción α que maximize esa heurística
  - FOIL se basa en la ganancia de información:

    $$h(regla, \alpha) = t \left(log_2 \frac{p_1}{p_1 + n_1} - log_2 \frac{p_0}{p_0 + n_0} \right)$$

    donde:
    - n_0, p_0: n° ejemplos - y + que cubre la regla original
    - n_1, p_1: n° ejemplos - y + que cubre la regla al añadirle α
    - t : n° ejemplos positivos cubiertos por la regla original también cubiertos por la nueva regla