Programación Automática

MÁSTER EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

Programación Lógica Inductiva y Aprendizaje Relacional

Fernando Fernández y Raquel Fuentetaja

Contenidos

Aprendizaje proposicional vs. relacional

- Aproximaciones proposicionales: buscan patrones en una única tabla de datos
- Aproximaciones relacionales: buscan patrones en múltiples tablas (relaciones) de una base de datos relacional
- Muchos algoritmos de aprendizaje proposicional tienen una versión relacional

Ejemplo

Tabla clientes					
id	sexo	edad	ingresos	total-gastado	buen-cliente
c1	hombre	30	214000	18800	Si
c2	mujer	19	139000	151000	Si
c3	hombre	55	50000	12400	No
c4	mujer	48	26000	8600	No
...	...	...	...	...	...

Tabla casado-con	
Persona1	Persona2
c1	c2
c2	c1
c3	c4
c4	c3
...	...

Ejemplo

- Regla proposicional

  IF ingresos $>$ 10800 THEN buen-cliente = si

- Regla relacional

  buen-cliente(C1, Edad1, Ingresos1, Total1) $\leftarrow$
  casado-con(C1,C2) $\wedge$
  cliente(C2,Age2,Ingresos2,Total2,B) $\wedge$
  Ingresos2 $>=$ 108000.

- El lenguaje relacional es más expresivo: lógica de primer orden

De relacional a proposicional

- Generar una única tabla a partir de varias (join y agregación): pérdida de informácion
- Ejemplo

  cliente(Id, Nombre, Edad, GastaMucho)
  compra(IdCliente, Idcompra, Fecha, Cantidad, FormaPago)
- Cada cliente puede realizar varias compras
- Posible agregación

  cliente1(IdCliente, Edad, NúmeroCompras, CantidadTotal, GastaMucho)
  - Pérdida de información
  - Pérdida de expresividad

    cliente(Idcliente, Nombre, Edad, si) $\leftarrow$
    Edad $>$ 30 $\wedge$ compra(IdCliente, IdCompra,F,C,P) $\wedge$
    F = tarjetacredito $\wedge$ C = 100.

Programación Lógica Inductiva (ILP)

- Aprende automáticamente reglas expresadas en lógica de primer orden
- Es básicamente inferir programas PROLOG a partir de ejemplos
- Uno de los sistemas ILP más conocidos es FOIL

Terminología

- Términos
  - Constantes: a,b,c, ...
  - Variables: X, Y, Z ...
  - Funciones sobre términos: age(juan)
- Literal: predicado o su negación sobre términos: padre(juan, X), ¬padre(juan, ana)
- Claúsula: disyunción de literales (variables cuantificadas universalmente)
- Claúsula de Horn: claúsula que contiene como mucho un literal positivo

$$H \vee \neg L_1 \vee \neg L_2 \vee \neg L_3$$

$$H \leftarrow L_1 \wedge L_2 \wedge L_3$$

$$\text{IF } L_1 \wedge L_2 \wedge L_3 \text{ THEN } H$$

FOIL [Quinlan, 90]

- Aprende reglas compuestas por claúsulas de Horn, pero
  - No se permiten símbolos de función
  - Sin embargo se permiten literales negados en el cuerpo de las reglas

FOIL: tarea

- Dados
  - E: ejemplos positivos del predicado a aprender
  - N: ejemplos negativos del predicado a aprender
  - B: otro conocimiento que se puede utilizar para expresar la definición del predicado a aprender (*Background Knowledge*)
- Encontrar una definición del predicado a aprender que sea
  - **Completa**: todos los ejemplos positivos son ciertos, $B \wedge H \models E$
  - **Consistente**: ningún ejemplo negativo es cierto $B \wedge H \not\models N$

FOIL: algoritmo

FOIL (P : predicado meta, B : background knowledge, E : ejemplos)

$Pos \leftarrow$ *Ejemplos* para los que *Predicado_meta* es cierto
$Neg \leftarrow$ *Ejemplos* para los que *Predicado_meta* es falso
$Regla \leftarrow \{\}$

while Pos **do**
    /* Aprender una nueva claúsula C */
    $C \leftarrow P$. (predicado meta sin precondiciones)
    NegativosCubiertos $\leftarrow$ *Neg*

    while *NegativosCubiertos* **do**
        /* Añadir un nuevo literal para especializar C */
        Candidatos $\leftarrow$ `Generar_Candidatos`(B)
        MejorCandidato $\leftarrow \underset{L \in Candidatos}{\arg\max}$ `Ganancia_Foil`(L, C)
        Añadir *MejorCandidato* a las precondiciones de C
        Eliminar de *NegativosCubiertos* los que no satisfacen las precondiciones de C

    $Regla \leftarrow Regla \cup C$
    $Pos \leftarrow Pos$ - Positivos cubiertos por C
return *Regla*

Generación de candidatos

- $Q(V_1, \ldots, V_r)$
  - Q predicado en el *background knowkledge*
  - V_i variables tal que al menos una debe aparecer previamente en la claúsula
- $V_j = V_k$ y $V_j = c$ para variables que aparecen en la claúsula
- Negación de las anteriores

Ganancia FOIL

Mide el número de instanciaciones de la claúsula que cubren ejemplos positivos y negativos antes y después de añadir un literal

$$\texttt{Ganancia\_Foil}(L, C) = t \left(\log_2 \frac{p^C}{p^C + n^C} - \log_2 \frac{p^{CL}}{p^{CL} + n^{CL}} \right)$$

- p^C y p^{CL}: número de instanciaciones de C que dan lugar a ejemplos positivos antes y después de añadir L respectivamente
- n^C y p^{CL}: número de instanciaciones de C que dan lugar a ejemplos negativos antes y después de añadir L respectivamente
- t: número de instanciaciones de C que dan lugar a ejemplos positivos y se mantienen después de añadir L

Ejemplo

- Predicado a aprender: miembro(A,B)
- Ejemplos

(1, [1])⊕	(2, [2])⊕	(3, [3])⊕	(1, [1, 2])⊕	(2, [1, 2])⊕
(2, [2, 3])⊕	(3, [2, 3])⊕	(1, [1, 2, 3])⊕	(2, [1, 2, 3]⊕	(3, [1, 2, 3])⊕
(1, [ ])⊖	(1, [2])⊖	(1, [3])⊖	(1, [2, 3])⊖	(2, [ ])⊖
(2, [1])⊖	(2, [3])⊖	(3, [ ])⊖	(3, [1]⊖	(3, [2])⊖
(3, [1, 2])⊖				

- Background knowledge: **componentes(H,R,L)**

([1], 1, [ ]) ([2], 2, [ ]) ([3], 3, [ ]) ([1, 2], 1, [2]) ([2, 3], 2, [3]) ([1, 2, 3], 1, [2, 3])

- Los ejemplos negativos se pueden determinar asumiendo *mundo cerrado*

Ejemplo

- Claúsula inicial miembro(A,B):-
- Supongamos que se añade componentes(B,A,C)
  miembro(A,B):-componentes(B,A,C).
  - Instanciaciones que satisfacen la claúsula
    (1, [1], [ ])⊕ (2, [2], [ ])⊕ (3, [3], [ ])⊕ (1, [1, 2], [2])⊕ (2, [2, 3], [3])⊕
    (1, [1, 2, 3], [2, 3])⊕

Ejemplo

- Claúsula inicial miembro(A,B):-
- Supongamos que se añade componentes(B,A,C)
  miembro(A,B):-componentes(B,A,C).
  - Instanciaciones que satisfacen la claúsula
    $(1, [1], [\,])\oplus$ $(2, [2], [\,])\oplus$ $(3, [3], [\,])\oplus$ $(1, [1, 2], [2])\oplus$ $(2, [2, 3], [3])\oplus$
    $(1, [1, 2, 3], [2, 3])\oplus$
  - $p^C = 10$, $n^C = 11$
  - $p^{CL} = 6$, $n^{CL} = 0$
  - $t = 6$
  - $Ganancia\_Foil(L, C) = 6{,}42$

Ejemplo

- Claúsula inicial miembro(A,B):-
- Supongamos que se añade componentes(B,A,C)
  miembro(A,B):-componentes(B,A,C).
  - Instanciaciones que satisfacen la claúsula
    $(1, [1], [\,])\oplus$ $(2, [2], [\,])\oplus$ $(3, [3], [\,])\oplus$ $(1, [1, 2], [2])\oplus$ $(2, [2, 3], [3])\oplus$
    $(1, [1, 2, 3], [2, 3])\oplus$
  - $p^C = 10$, $n^C = 11$
  - $p^{CL} = 6$, $n^{CL} = 0$
  - $t = 6$
  - $Ganancia\_Foil(L, C) = 6{,}42$

Ejemplo

- Nueva claúsula
- Comienza con los ejemplos positivos restantes y todos los negativos

(2, [1, 2])⊕	(3, [2, 3])⊕	(2, [1, 2, 3])⊕	(3, [1, 2, 3])⊕	
(1, [ ])⊖	(1, [2])⊖	(1, [3])⊖	(1, [2, 3])⊖	(2, [ ])⊖
(2, [1])⊖	(2, [3])⊖	(3, [ ])⊖	(3, [1]⊖	(3, [2])⊖
(3, [1, 2])⊖				

- Si se añade componentes(B,C,D)

miembro(A,B):-componentes(B,C,D).

  - Instanciaciones que satisfacen la claúsula

(2, [1, 2], 1, [2])⊕	(3, [2, 3], 2, [3])⊕	(2, [1, 2, 3], 1, [2, 3])⊕	(3, [1, 2, 3], 1, [2, 3])⊕
(1, [2], 2, [ ])⊖	(1, [3], 3, [ ])⊖	(1, [2, 3], 2, [3])⊖	(2, [1], 1, [ ])⊖
(2, [3], 3, [ ])⊖	(3, [1], 1, [ ])⊖	(3, [2], 2, [ ])⊖	(3, [1, 2], 1, [2])⊖

Ejemplo

- Nueva claúsula
- Comienza con los ejemplos positivos restantes y todos los negativos

(2, [1, 2])⊕	(3, [2, 3])⊕	(2, [1, 2, 3])⊕	(3, [1, 2, 3])⊕	
(1, [ ])⊖	(1, [2])⊖	(1, [3])⊖	(1, [2, 3])⊖	(2, [ ])⊖
(2, [1])⊖	(2, [3])⊖	(3, [ ])⊖	(3, [1]⊖	(3, [2])⊖
(3, [1, 2])⊖				

- Si se añade componentes(B,C,D)

  miembro(A,B):-componentes(B,C,D).

  - Instanciaciones que satisfacen la claúsula

(2, [1, 2], 1, [2])⊕	(3, [2, 3], 2, [3])⊕	(2, [1, 2, 3], 1, [2, 3])⊕	(3, [1, 2, 3], 1, [2, 3])⊕
(1, [2], 2, [ ])⊖	(1, [3], 3, [ ])⊖	(1, [2, 3], 2, [3])⊖	(2, [1], 1, [ ])⊖
(2, [3], 3, [ ])⊖	(3, [1], 1, [ ])⊖	(3, [2], 2, [ ])⊖	(3, [1, 2], 1, [2])⊖

Ejemplo

- Si se añade member(A,D)

  miembro(A,B):-componentes(B,C,D), member(A,D).

  - Instanciaciones que satisfacen la claúsula

    (2, [1, 2], 1, [2])⊕ (3, [2, 3], 2, [3])⊕ (2, [1, 2, 3], 1, [2, 3])⊕ (3, [1, 2, 3], 1, [2, 3])⊕

Ejemplo

- Si se añade member(A,D)

  miembro(A,B):-componentes(B,C,D), member(A,D).

  - Instanciaciones que satisfacen la claúsula

    $(2, [1, 2], 1, [2])\oplus$ $(3, [2, 3], 2, [3])\oplus$ $(2, [1, 2, 3], 1, [2, 3])\oplus$ $(3, [1, 2, 3], 1, [2, 3])\oplus$

Ejemplo: regla aprendida

```
miembro(A,B):-componentes(B,A,C).
miembro(A,B):-componentes(B,C,D), member(A,D).
```

Contenidos

Árboles de Decisión y Regresión: S-CART

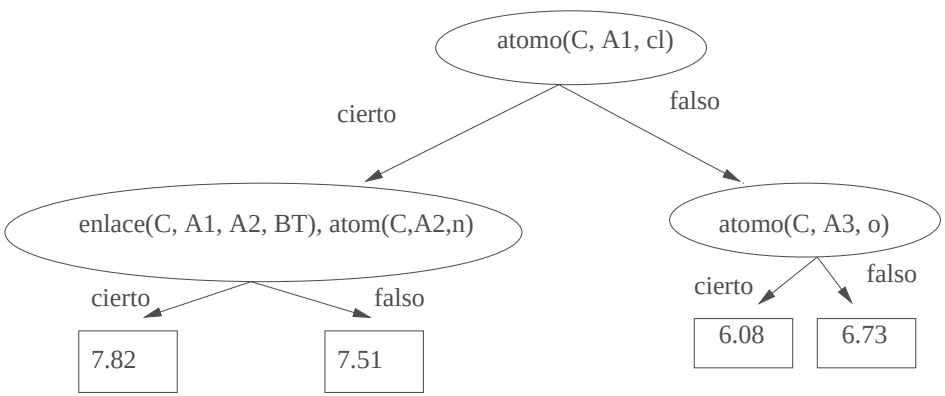
- S-CART: Structural Classification and Regression Trees
- Derivado de CART: árboles de decisión con representación atributo-valor
- Permite:
  - Clasificación: predicción de clases discretas
  - Regresión: predicción de valores continuos.
- Considera propiedades estructurales de los ejemplos en los nodos de decisión

Árboles de Decisión y Regresión con Representación Atributo-Valor

- Ventajas de los árboles de decisión:
  - Baja complejidad computacional
  - Buena aceptación por los usuarios
  - Manejo efectivo del ruido y la incertidumbre
  - Base teórica bien entendida
- Desventajas de los árboles con representación proposicional:
  - Vectores de atributos de tamaño fijo: ejemplos de entrenamiento como una matriz de valores, de la que no se puede obtener información estructural
  - Problema de la replicación y la fragmentación
  - Inestables ante el conjunto de entrenamiento

Árboles de Decisión y Regresión con Representación Relacional

- Árbol de decisión que predice la biodegradabilidad de un componente a partir de su estructura.
- La cantidad a predecir es el logaritmo de la mitad del tiempo de biodegradación del componente en agua.
- En la figura, C es un componente, A_i es un átomo, y BT es un tipo de enlace



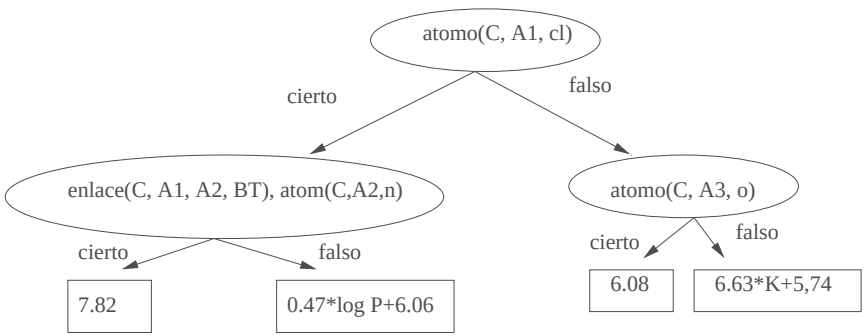

```
actividad(C, A) :- átomo(C, A1, cl),
                   enlace(C, A1,A2, BT),
                   átomo(C, A2, n),
                   A is 7.82, !.
actividad(C, A) :- átomo(C, A1, cl),
                   A is 7.51, !.
actividad(C,A) :- átomo(C, A3, o),
                   A is 6.08, !.
actividad(C,A) :- A is 6.73.
```

Árbol de Regresión

Representación Prolog

Árboles de Decisión y Regresión con Representación Relacional

- Un árbol de clasificación o decisión es como el anterior, pero en sus hojas, en vez de valores, tiene clases.
- Árbol con modelos de regresión:



Árbol de Regresión

actividad(C, A) :- átomo(C, A1, cl),
enlace(C, A1,A2, BT),
átomo(C, A2, n),
A is 7.82, !.
actividad(C, A) :- átomo(C, A1, cl),
A is $0{,}47 * \log P + 6{,}06$, !.
actividad(C,A) :- átomo(C, A3, o),
A is 6.08, !.
actividad(C,A) :- A is $0{,}63 * K + 7{,}74$.

Representación Prolog

S-CART: Structural Classification and Regression Trees

- S-CART: algoritmo que aprende una teoría para la predicción de clases discretas o valones numéricos a partir de un conjunto de ejemplos y conocimiento del dominio relacional.
- Pasos del algoritmo:
  - Construir el árbol
  - Podar el árbol
  - Añadir modelos de regresión lineal

Método Divide y Vencerás para Árboles Proposicionales

DivideYVencerás(*ejemplos***)**

if `CondiciónFin(`*ejemplos*`)` **then**
    nuevaHoja = `CrearNuevaHoja(`*ejemplos*`)`
    return *nuevaHoja*
else
    mejorAtributo = `EncontrarMejorAtributo(`*ejemplos*`)`
    particiones = `PartirEjemplos(`*mejorAtributo*`,` *ejemplos*`)`
    subarboles = `[]`

    for each *particion* $\in$ *particiones* **do**
        $subarbol_i$ = `DivideYVencerás(`$particion_i$`)`
        subarboles = [$subarbol_i$ | *subarboles*]

    return [*mejorAtributo* | *subarboles*]

Método Divide y Vencerás para Árboles Relacionales

DivideYVencerás(*testsAnteriores*, *ejemplos***)**

if `CondiciónFin(`*ejemplos*`)` **then**
    nuevaHoja = `CrearNuevaHoja(`*ejemplos*`)`
    return *nuevaHoja*

else
    mejorTest = `EncontrarMejorTest(`*testsAnteriores*`,` *ejemplos*`)`
    (*particion*$_1$, *particion*$_2$) = `PartirEjemplos(`*mejorTest*`,` *ejemplos*`,` *testsAnteriores*`)`
    subarbol$_1$ = `DivideYConquistarás(`*testsAnteriores* $\wedge$ *mejorTest*`,` *particion*$_1$`)`
    subarbol$_2$ = `DivideYConquistarás(`*testsAnteriores*`,` *particion*$_2$`)`
    return [*mejorTest*, *subarbol*$_1$, *subarbol*$_2$]

Diferencias entre árboles Relacionales y Proposicionales

- Los tests no son simples comprobaciones de atributos, sino literales o conjunciones de literales que contienen variables.
- Dos tests en una rama deben contener variables en común, por lo que, en cada punto, el test que se puede hacer dependen de los tests anteriores
- Sólo se propagan los test positivos
- Sólo decisiones binarias

Cálculo de Literales o Conjunción de Literales en cada Nodo

- *ejemplos*: conjunto de ejemplos en un nodo
- *testsAnteriores*: conjunción de todos los tests anteriores positivos en el camino desde la raíz hasta el nodo actual.
- Asumimos que tenemos un conjunto de posibles literales o refinamiento para el nodo actual, calculados con la función *refs*(*testsAnteriores*)
- Para cada refinamiento $ref_i \in refs(testsAnteriores)$ se evalúa teniendo en cuenta como particiona el conjunto de ejemplos:
  - Clasificación: función de la frecuencia relativa de cada clase
  - Regresión: función del error cuadrático medio

Sesgo Declarativo del Lenguaje

- La función *refs*(*testsAnteriores*) se genera teniendo en cuenta sesgos declarativos del lenguaje
- El diseñador define cómo se pueden insertar literales mediante esquemas
- Ejemplo:

  esquema ((enlace(V,W,X,Y), átomo(V, X, Z))
  [V: químico:'+', W:id_átomo:'+', X:id_átomo:'+',
  Y:tipo_enlace:'-', Z:elemento:=]).

- En la primera lista aparecen los predicados que se pueden añadir (sólos o como conjunción de varios de ellos)
- Para cada variable se define el tipo y el modo
- Modos:
  - '+': La variable debe ser unificada con una variable ya existente
  - '-': La variable puede no estar unificada
  - '=': Se debe incluir una constante

Contenidos

Introducción

Los métodos basados en medidas de distancia asumen que es posible calcular, para cada par de objetos en el dominio, su distancia mutua (o similitud)

Aproximaciones

- Aprendizaje predictivo
  - Almacenar todos los ejemplos disponibles
  - Dado un nuevo ejemplo no clasificado, predecir el valor asociado buscando el vecino más cercano al objeto consulta, es decir, el objeto ejemplo que tiene la menor distancia al objeto consulta.
- Agrupación o clustering:
  - Agrupar un conjunto de instancias, $\mathcal{I}$, en diferentes subconjuntos, grupos o *clusters*, de forma que los objetos que pertenecen a un mismo grupo maximicen la función de similitud entre ellos a la vez que minimizan su similitud respecto a las instancias de los otros grupos
  - Estrategias heurísticas: k-medias y métodos aglomerativos

Datos relacionales: Medida de Distancia para predicados en lógica de primer orden

Ejemplo: cestas de navidad

- Vocabulario:
  - P(a_1: nombre, a_2: número, a_3: discreto)
    %(identificador, precio, modo entrega)
  - queso(a_1: nombre, a_2: discreto, a_3: número, a_4: discreto)
    % (identificador, tipo de queso, peso, origen)
  - vino(a_1: nombre, a_2: nombre, a_3: número, a_4:número)
    % (identificador, tipo de vino, año, tamaño)
  - bodega(a_1: nombre, a_2: discreto, a_3: discreto, a_4: discreto)
    %(identificador, popularidad, tamaño, origen)
- Tipos: nombre, discreto, número, lista, término
- El tipo nombre se utiliza para identificadores (claves de una base de datos relacional)

Ejemplo

Una instancia se compone de

1. La instancia en sí:

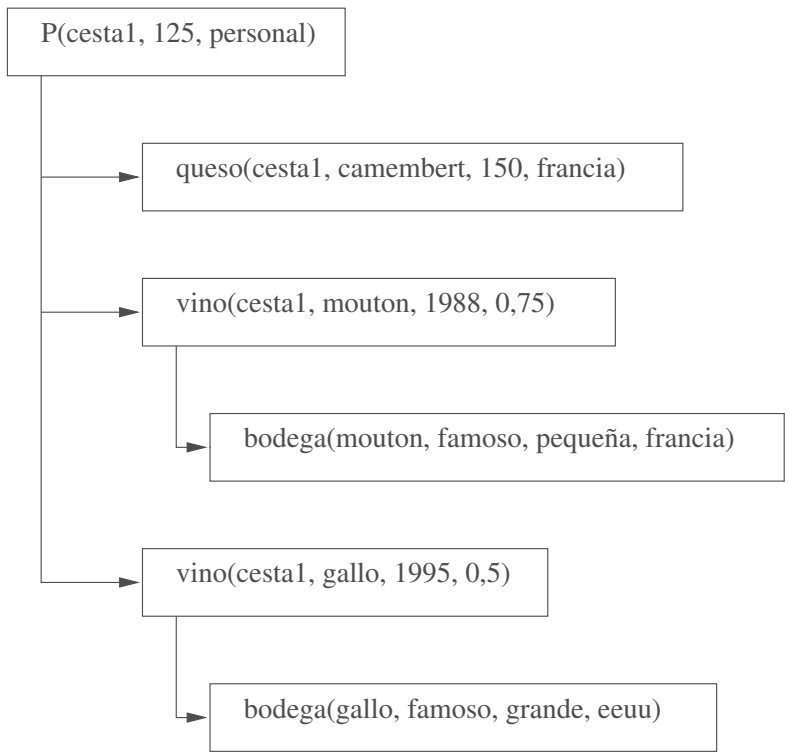
$$I = P(cesta1, 125, personal)$$

2. Y el conocimiento de fondo ó background knowledge:

$queso(cesta1, camembert, 150, francia)$
$vino(cesta1, mouton, 1988, 0{,}75)$
$vino(cesta1, galla, 1995, 0{,}5)$
$bodega(gallo, famosa, grande, eeuu)$
$bodega(mouton, famosa, pequeña, francia)$

Definición de Casos de Instancias

- El caso de una instancia $\mathcal{I}$, dado un conocimiento de fondo $\mathcal{B}$ y un límite de profundidad (entero $d \geq 0$) es el conjunto más grande $\{\mathcal{I}\} \cup \mathcal{B}'$, tal que $\mathcal{B}' \subseteq \mathcal{B}$, y para cada literal atómico de fondo $B \in \mathcal{B}'$ se cumple que hay un $d' < d$ y $B_0, B_1, \ldots, B_{d'}$ que satisfacen:
  - $B_0 = I$, $B_{d'} = B$, y $B_i \in \mathcal{B}'$ para $i = 1, \ldots, d' - 1$.
  - B_i y B_{i+1} contienen al menos un identificador común



Definición de la Distancia entre dos Instancias

- Pasos en el cálculo de la distancia entre dos instancias I_1 e I_2:
  - Calcular sus correspondientes casos de instancia
  - Calcular recursivamente la distancia entre dos instancias:

$$dist(\mathcal{I}_i, \mathcal{I}_j, \mathcal{B}, d) = \frac{1}{n} \sum_{k=1}^{n} dist(t_k(\mathcal{I}_i), t_k(\mathcal{I}_j))$$

- Donde:
  - Distancia entre números: distancia normalizada entre 0 y 1 (o distancia equivalente)
  - Distancia entre valores discretos: 1 si son iguales, 0 si son distintos (o distancia equivalente)
  - Distancia entre términos y listas: número de pasos que necesito para convertir el primer elemento en el segundo, o viceversa, dada una serie de operadores como insertar, borrar, etc.

Definición de la Distancia entre dos Instancias

Y donde la distancia entre nombres se calcula de la siguiente forma:

- Si la recursión ha alcanzado la profundidad máxima, compararlos como si fueran tipos discretos
- En otro caso:
  - Coleccionar todos los hechos del caso que incluyan el nombre de los dos objetos, generando dos conjuntos de predicados
  - Cada conjunto anterior es dividido en predicados
  - Para cada predicado en los dos conjuntos, elegimos aquel conjunto con menos elementos. Calculamos recursivamente la distancia mínima entre esos predicados con los predicados del conjunto mayor, devolviendo ese valor.

Ejemplo de Cálculo de la Distancia

- Instancias:
  - I_1=P(cesta1,125,personal)
  - I_2 =P(cesta25,195,mail)
- Conocimiento de fondo:
  - queso(cesta1,camembert,150,francia)
  - queso(cesta25,roquefort,200,francia)
  - queso(cesta25,ricotta,100,italia)
  - vino(cesta1,mouton, 1988, 0.75)
  - vino(cesta1,galla, 1995, 0.5)
  - vino(cesta25, mouton, 1995, 0.75)
  - bodega(gallo, famosa, grande, eeuu)
  - bodega(mouton, famosa, pequeña, francia)
- $d = 2$

Ejemplo de Cálculo de la Distancia

$$dist(l_1, l_2, \mathcal{B}, 2) =$$
$$\frac{1}{3}(dist(cesta1, cesta25) + dist(125, 195) + dist(personal, correo))$$

$$dist(125, 195) = \frac{|195 - 125|}{500}$$

$$dist(personal, correo) = 1$$

$$dist(cesta1, cesta25) = ?$$

Ejemplo de Cálculo de la Distancia

Generar los conjuntos de predicados que continen *cesta*1 y *cesta*2:

- $L_{cesta1,queso} = \{queso(cesta1, camembert, 150, francia)\}$
- $L_{cesta1,vino} = \{vino(cesta1, mouton, 1988, 0{,}75), vino(cesta1, galla, 1995, 0{,}5)\}$
- $L_{cesta25,queso} = \{queso(cesta25, roquefort, 200, francia), queso(cesta25, ricotta, 100, italia)\}$
- $L_{cesta25,vino} = \{vino(cesta25, mouton, 1995, 0{,}75)\}$

$$D_{queso} = \frac{\min_{l \in L_{cesta25,queso}} dist(queso(cesta1, camembert, 150, francia), l)}{|L_{cesta25,queso}|} =$$
$$= \frac{min((1 + \frac{|150-200|}{300} + 0)/3, 1 + \frac{|150-100|}{300} + 1/3)}{2} = 0{,}1944$$

$$dist(mouton, gallo) = \frac{min(\frac{dist(famoso,famoso)+dist(pequena,grande)+dist(francia,eeuu)}{3})}{1} = 0{,}666$$

$$D_{vino} = \ldots = 0{,}0117$$

$$dist(cesta1, cesta25) = \frac{D_{queso}+D_{vino}}{2} = 0{,}1031$$

Algoritmos

- KNN Relacional: equivalente a KNN proposicional, pero con la nueva medida de distancia
- K-medias relacional: también equivalente excepto en el cálculo del centroide $\rightarrow$ nos quedamos con la instancia que minimice su distancia media al resto de las instancias del clúster

Contenidos

Aprendizaje de Conjuntos Frecuentes: APRIORI

- Eficiencia: cualquier subconjunto de un conjunto frecuente es también un conjunto frecuente
- Algoritmo iterativo
  - Generar conjuntos con $k = 1$ ítems
  - Los conjuntos frecuentes de tamaño 1 son aquellos con

    $$soporte > soporte\_minimo$$

  - Repetir hasta que no se encuentran más
    1. $k = k + 1$
    2. Generar conjuntos de tamaño k a partir de los conjuntos frecuentes de tamaño $k - 1$
    3. Mantener sólo aquellos con $soporte > soporte\_minimo$
- Ejemplo: carro de la compra

WARMR [Dehaspe Toivonen, 99]

- Versión relacional del algoritmo APRIORI
- Permite encontrar patrones frecuentes en datos relacionales
- Estos patrones frecuentes se expresan como conjunciones de átomos en lógica de predicados

$$persona(X) \wedge tiene\_mascota(X, Y)$$

- Es necesario definir una clave que representa el predicado principal sobre el que se extraen los patrones frecuentes

$$warmode\_key(persona(-)).$$

Definición del Problema

- Dada una base de datos relacional D y una clave *key*, se trata de encontrar el conjunto patrones frecuentes que contienen la clave
- Un patrón es frecuente si su frecuencia es mayor o igual que una frecuencia mínima (umbral) introducida por el usuario.
- La frecuencia de un patrón P es el porcentaje de ejemplos para los que el patrón es cierto y se calcula como el porcentaje de sustituciones de las variables del predicado clave que hacen el patrón cierto

$$frecuencia(P, D, key) = \frac{|\{\theta \in answerset(? - key, D) \mid P\theta \text{ es cierto en D}\}|}{|\{\theta \in answerset(? - key, D)\}|}$$

Ejemplo: cálculo de la frecuencia

- Datos D:
  - $persona(ana)$, $persona(juan)$, $persona(pedro)$, $persona(isabel)$
  - $tiene\_mascota(isabel, gato)$, $tiene\_mascota(pedro, perro)$, $tiene\_mascota(ana, gato)$
- Clave: predicado *persona*
- Patrón:

$$P : persona(X) \wedge tiene\_mascota(X, gato)$$

$$frecuencia(P, D, persona) = \frac{2}{4}$$

Funcionamiento

- Análogamente a la búsqueda de conjuntos frecuentes, un patrón frecuente más complejo (más específico) sólo se puede generar a partir de uno más simple (más general). Sin embargo.
  - En APRIORI todos los subconjuntos de conjuntos frecuentes son también conjuntos frecuentes
  - En WARMR no todos los sub-patrones de un patrón frecuente son necesariamente patrones frecuentes, por lo que es necesario mantener una lista de patrones *infrecuentes*
- En el nivel 1, WARMR comienza con un patrón que sólo contiene la clave y va generando candidatos para el siguiente nivel $l + 1$ refinando (añadiendo literales) a los patrones frecuentes del nivel anterior l

Algoritmo WARMR

WARMR(D**,** *key***,** *minfrec***):** Q
Entradas: base de datos D, clave *key*, umbral de frecuencia *minfrec*
Salidas: Todos los patrones Q cuya frecuencia $>=$ *minfrec*

1. $nivel = 1$
2. Inicializar conjunto de candidatos $Q_1 = \{key\}$
3. Inicializar conjunto de patrones frecuentes e infrecuentes $F = \emptyset$, $I = \emptyset$
4. **while** Q_d no esté vacío
5. Calcular la frecuencia de los patrones en Q_d
6. Incluir aquellos con frecuencia $<$ *minfrec* en I
7. Actualizar $F = F \cup Q_d$
8. Calcular nuevos candidatos:
$Q_{d+1} = \text{WARMRgen}(I, F, Q_d)$
9. $d = d + 1$
10. **return** F

Generación de Candidatos

WARMRgen(I, F, Q_d)

1. Inicializar $Q_{d+1} = \emptyset$
2. for each $Q_j \in Q_d$ **do**
    for each refinamiento Q'_j de Q_j **do**
        Añadir Q'_j a Q_{d+1} a menos que:
        (a) Q'_j sea más específico que algún patrón en I
        (a) Q'_j sea equivalente a algún patrón en $Q_{d+1} \cup F$
3. return Q_{d+1}

Sesgo Declarativo

- Permite expresar restricciones sobre cómo se realiza el refinamiento para la generación de candidatos
- Define qué predicados se pueden utilizar y cómo deben ser sus parámetros

$$
\begin{aligned}
&\textit{warmode\_key}(\textit{persona}(-))\\
&\textit{warmode\_key}(\textit{padre}(+,-))\\
&\textit{warmode\_key}(\textit{tiene\_mascota}(+,\textit{gato}))\\
&\textit{warmode\_key}(\textit{tiene\_mascota}(+,\textit{perro}))\\
&\textit{warmode\_key}(\textit{tiene\_mascota}(+,\textit{loro}))
\end{aligned}
$$

Bibliografía

- Machine Learning. Tom Mitchell. Capítulo 10
- Relational Data Mining. Saso Dzeroski y Nada Lavrak (editores). Springer 2001. Capítulos 6 y 9
- Clausal Discovery. Luc de Taedt, Luc Dehaspe. Machine Learning 26, 99-146. 1997
- Induction of logic programs: FOIL and related systems. New Generation Computing 13. 1995
- Multi-relational data mining: an introduction. Saso Dzeroski. ACM SIGKDD Explorations Newsletter. Volume 5 Issue 1. 2003.
- Discovery of frequent DATALOG patters. Luc de Dehaspe y Hannu Toivonen. Data Mining and Knowledge Discovery, 3. 1999