Programación Lógica Inductiva

Sara Olías Zapico

10 de junio de 2021

Índice

1. Introducción

Hoy en día la clasificación de datos en clases y subclases representa uno de los desafíos computacionales más importantes debido a la gran aplicabilidad de los mismos. Los algoritmos de aprendizaje que proporcionan mejores resultados en este ámbito son aquellos que inducen a árboles de decisión o reglas de clasificación, sin embargo, su lenguaje de representación es esencialmente proposicional.

Estos algoritmos presentan problemas cuando tratamos con grandes conjuntos de datos o cuando nos encontramos con demasiadas relaciones recursivas. Por ejemplo, si consideramos un árbol genealógico muy grande (con muchas relaciones entre sus nodos y hojas), las conexiones directas pueden ser realmente fáciles de mostrar y aprender para estos algoritmos, pero las conexiones complejas darían lugar a declaraciones largas y salidas confusas. Esto ocurre porque los algoritmos descritos hablan de un solo objeto a la vez y no podemos relacionar propiedades de dos o mas objetos a menos que definamos una nueva propiedad que exprese esa relación.

La **Programación Lógica Inductiva** o **IPL** muestra fortalezas adicionales cuando se trata de mostrar relaciones recursivas de datos. El término IPL, como lo define Stephen Muggleton, describe una intersección entre el aprendizaje inductivo y la programación lógica e incluye técnicas clásicas de aprendizaje automático y el poder de representación y formalismo de la lógica de primer orden.

Además de contar con una representación mas clara y compacta y con la capacidad de realcionar propiedades de más de un objeto a la vez, una de las ventajas más importantes que presenta la IPL es la capacidad de introducir nuevos predicados automáticamente durante el proceso de aprendizaje, simplificando la representación de los conceptos aprendidos.

2. Nociones de lógica y notación utilizada

La lógica proposional, es un sistema formal cuyos elementos más simples representan proposiciones o enunciados, y cuyas constantes lógicas representan operaciones sobre proposiciones, capaces de formar otras proposiciones de mayor complejidad. Pero en la Lógica proposicional las proposiciones no se analizan, sino que se toman como un todo, en bloque. Las proposiciones son los elementos últimos sobre los cuales opera esta rama de la Lógica. Las proposiciones $\ll$*Las moscas son insectos*$\gg$ y $\ll$*La Tierra es un planeta*$\gg$ son proposiciones simples. En cambio, $\ll$*Las moscas son insectos y la Tierra es un planeta*$\gg$ y $\ll$*Si las moscas son insectos, entonces la Tierra es un planeta*$\gg$ son proposiciones complejas.

Necesitamos por tanto, conocer las siguientes nociones:

- Una **variable** se representa con una cadena de letras o dígitos empezando con una letra mayúscula.
- Un s**ímbolo funcional** se representa como una letra minúscula seguida de una cadena de letras y números.
- Un **término** es una constante, una variable o la aplicación de un símbolo funcional a un número determinado de términos.
- Un **átomo** o **forma atómica** es la aplicación de un predicado a un número de términos.
- Un **literal** es un átomo o su negación.
- Dos literales son **compatibles** si se llaman igual, tienen el mismo signo y el mismo número de argumentos.
- La **negación** se denota como $\neg$.
- Una **cláusula conjuntiva** es una conjunción de un conjunto finito de literales.
- Una **cláusula disyuntiva** es una disjunción de un conjunto finito de literales.
- Una **cláusula de Horn** es una cláusula con a lo más un literal positivo $H \leftarrow (L_1 \wedge \ldots \wedge L_n)$. La literal positiva (H) se llama la cabeza y las literales negativas, el cuerpo.
- Un conjunto de cláusulas de Horn es un **programa lógico**.
- Dado un lenguaje de primer orden L, una sustitución θ es una función del conjunto V de las variables en el conjunto $TERM$ de los terminos $(\theta : V \rightarrow TERM)$. Se denota por SUST al conjunto de las sustituciones.

- Definimos el dominio de una sustitucion θ como el conjunto

$$Dom(\theta) = \{x \in V : \theta(x) \neq x\}$$

- El codominio de una sustitucion θ como

$$Cod(\theta) = \{\theta(x) : x \in Dom(\theta)\}$$

- El rango de una sustitucion θ como

$$Rang(\theta) = \cup_{x \in Dom(\theta)} V(\theta(x))$$

- Decimos que el término t es una **instancia** del término s si existe una sustitución σ tal que $s(\sigma) = t$.
- Se dice que la sustitución Θ es un **unificador** de t_1 y t_2 si $(t_1)\Theta = (t_2)\Theta$
- Un **modelo** de un programa lógico es una interpretación para la cual las cláusulas asumen un valor de verdad.
- Decimos que F_1 implica semanticamente a F_2, o F_2 es consecuencia lógica de F_1, si y solo si todo modelo de F_1 es un modelo de F_2.

Definición 2.1. *Sean t_1 y t_2 dos términos. Se dice que θ es un **unificador de máxima generalidad**(umg) de t_1 y t_2 si:*

- $\theta \in \mathcal{U}(t_1, t_2)$
- *Para toda $\theta' \in \mathcal{U}(t_1, t_2), \theta' \leq \theta$, o dicho de otro modo, existe una sustitución θ'' al que $\theta\theta'' = \theta'$*

Por último, debemos hablar de la **resolución**, una regla de inferencia utilizada sobre fórmulas descritas en forma de cláusulas disyuntivas que se utiliza para probar, por medio de refutación, implicaciones sintácticas. Para hacer resolución tenemos que comparar si dos literales complementarios unifican. Este algoritmo de unificación construye el mgu de un conjunto de expresiones.

Ejemplo 2.1. Veamos un ejemplo en el que se aplica resolución dos veces:

1	$femenino(ana)$	*Hiptesis*
2	$hija(X,Y) :- femenino(X), padre(Y,X)$	*Hiptesis*
3	$padre(juan, ana)$	*Hiptesis*
4	$hija(ana, Y) :- padre(ana, X)$	Resolvente de 1 y 2
5	$hija(ana, juan)$	Resolvente de 3 y 4

3. Programación lógica inductiva.

La programación lógica es un paradigma de programación basado en la lógica de primer orden. En un programa lógico generalmente se definen un hechos y reglas, lo que se suele llamar base de conocimiento, y a partir de ellos, se pueden obtener respuestas. Vamos a verlo con un ejemplo sencillo en pseudo-prolog en el que definimos la nacionalidad de varias personas y las relaciones de pertenencia entre paises y continentes.

```
% Hechos:
es_español("Manolo").
es_italiano("Marco").
es_colombiano("Marcelo").

% Reglas:
es_europeo(A) :- es_español(A).
es_europeo(A) :- es_italiano(A).
es_americano(A) :- es_colombiano(A).
es_terricola(A) :- es_europeo(A).
es_terricola(A) :- es_americano(A).
son_del_mismo_continente(A,B) :- es_europeo(A), es_europeo(B).
son_del_mismo_continente(A,B) :- es_americano(A), es_americano(B).
```

Las reglas indican que, si se cumple la parte derecha, entonces se cumple la parte izquierda. Por ejemplo, la primera regla se leería «si A es español, entonces A es europeo». La coma en la parte derecha funciona como un operador lógico entre las claúsulas que aparecen. La última regla sería «Si A es americano y B es americano, entonces son del mismo continente A y B».

A partir de la base de conocimiento que hemos definido, podemos establecer objetivos y el sistema intentará satisfacerlos e indicarnos si son ciertos o falsos:

```
?- son_del_mismo_continente("Manolo", "Marco").
yes
```

También podemos establecer objetivos abiertos y el sistema nos dirá qué valores hacen que se cumplan:

```
?- es_europeo(A).
A = Manolo
A = Marco
```

Además de este tipo de proposiciones basadas en hechos y relaciones, la mayoría de los lenguajes de programación lógica soportan también predicados basados en restricciones.

Definición 3.1. *Un programa lógico P se dice* ***completo*** *(con respecto a un conjunto de ejemplos positivos ε^+) sii para todos los ejemplos $e \in \varepsilon^+, P \vdash e$.*

Definición 3.2. *Un programa lógico P se dice* ***consistente*** *(con respecto a un conjunto de ejemplos negativos ε^-) sii para todos los ejemplos $e \in \varepsilon^-, P \vdash e$.*

La idea principal en ILP es aprender una hipótesis que cubra los ejemplos positivos y no los negativos. Por ello se parte del objetivo que queremos resolver y se van seleccionando reglas hasta intentar encontrar una solución. Cuando nos enfrentamos a la resolución de un problema mediante ILP, partimos de:

- Un conjunto de ejemplos positivos ε^+
- Un conjunto de ejemplos negativos ε^-
- Un programa lógico consistente T (normalmente se refiere a conocimiento del dominio o conocimiento a priori), tal que $T \nvdash e^+$ para al menos un $e^+ \in \varepsilon^+$

Y nuestro objetivo es encontrar un programa lógico H tal que H y T sea completo y consistente, o lo que es lo mismo $T \cup H \vdash \varepsilon^+$ y $T \cup H \nvdash \varepsilon^-$.

Desde un punto de vista semántico la definición de IPL es:

- Satisfacibilidad previa: $T \wedge \varepsilon^- \nvdash \square$
- Satisfacibilidad posterior: $T \wedge H \wedge \varepsilon^- \nvdash \square$
- Necesidad previa: $T \nvdash \varepsilon^+$
- Suficiencia posterior: $T \wedge H \nvdash \varepsilon^+$

Para realizar una búsqueda eficiente de hipótesis, es necesario estructurar el espacio de hipótesis y para ello se suele utilizar un modelo de especificación, que nos diga si una hipótesis es más general o más específica que otra. Esto agiliza la búsqueda ya que nos permite "cortar ramas": como las especificaciones o generalizaciones de hipótesis suelen heredar alguna propiedad (incapacidad de cubrir un ejemplo verdadero, dar por valido un ejemplo conocido como falso, ...) nos hace más fácil eliminar estas hipótesis del espacio de búsqueda sin necesidad de probarlas una por una.

Esta estructuración, se puede hacer utilizando subumptiom. Necesitamos por lo tanto introducir los siguientes términos:

Definición 3.3. *Se define la relación de* ***subsunción*** *como la relación $\leq$ definida en $TERM$ del siguiente modo:*

$$t \leq s \Leftrightarrow \textit{Existe una sustitución } \sigma \textit{ tal que } (s)\sigma = t$$

Además, esta relación es un preorden en $TERM$, es decir, es reflexiva y transitiva en $TERM$.

Definición 3.4. *Definimos en TERM la relación $<$ del siguiente modo:*

$$t < s \Leftrightarrow (s \leq t) \wedge (t \not\leq s)$$

Llamaremos a esta relación ***subsunción estricta****.*

Definición 3.5. *Decimos que una cláusula C es más general que C' si C $\theta-$subsume a C'. También se dice en este caso que C' es una especificación de C.*

Para la estruccturación del espacio de hipótesis, es muy útil utilizar la $\theta-$subsumpción ya que este es decidible entre cláusulas, es fácil de calcular y crea un *lattice*. Esto es importante porque permite buscar en ese lattice por hipótesis. La busqueda puede hacerse:

(i) De específico a general, buscando cláusulas que subsuman a la hipótesis actual.

(ii) De general a específico, buscando clausulas subsumidas por la hipótesis actual.

(iii) En ambos sentidos.

En las siguientes secciones veremos algunos ejemplos de estos métodos.

3.1. Clasificación de los sistemas ILP

Una vez analizado el fundamento de la ILP, se definen diferentes criterios para clasificar sus sistemas

- **Según el número de conceptos:** se refiere a la cantidad de conocimientos (predicados) que se aprenden en el proceso:
  - *Sistemas Simples:* Pueden aprender un único concepto. Estos sistemas son más populares, sin embargo, menos potentes.
  - *Sistemas Múltiples:* Pueden aprender múltiples conceptos. Estos sistemas son más potentes, lo cual implica que son sistemas más eficientes y duros.
- **Según el número de ejemplos:**
  - *Sistemas de Aprendizaje de a un paso o no Incrementales*: Para ello se necesitan todos los ejemplos disponibles y estos son dados al comienzo y no cambia en el proceso.
  - *Sistemas de Aprendizaje secuencial o Incrementales*: Los ejemplos son suministrados al sistema por el usuario uno a uno, de forma prudente y paulatina.
- **Según se necesite o no un supervisor:**
  - *Sistemas Interactivos*: Cuando el supervisor valida la corrección del aprendizaje,lo que le permite podar grandes partes del espacio de hipótesis.
  - *Sistemas no Interactivos*: El caso contrario. La mayoría de los sistemas son no interactivos.

- **Según la técnica empleada:**
  - *Sistemas Botton-Up*: Estos sistemas están basados en la generalización de un conjunto de ejemplos positivos.
  - *Sistemas Top-Down*: Realizan una búsqueda de general a específico en grafos de refinamiento.
- **Según la revisión de teoría:** Son todos aquellos sistemas ILP que permiten que el conocimiento base B sea corregido durante el proceso de aprendizaje. Se trata que dichas correcciones sean mínimas.

4. Métodos de específico a general (Bottom-Up)

La generalización se entiende como una tarea que va de lo específico hacia lo general (Bottom-Up): el punto de partida son los ejemplos y el objetivo final es la descripción aprendida en forma de reglas que cubren esos ejemplos y quizás otros. En esta sección vamos a presentar las técnicas más utilizadas para conseguir esta generalización.

4.1. Generalización menos general

Una forma de ir buscando hipótesis es generalizando las cláusulas gradualmente. Fué *Plotking* uno de los primeros en introducir el sistema de la *generalización menos general (lgg)*, que se define como sigue:

Definición 4.1. *La cláusula C es la Generalización Menos General de otra cláusula D bajo una $\theta-$subsumpción si $C \leq D$, es decir, la cláusula C es más general que la cláusula D, y para cualquier otra cláusula E tal que $E \leq D$, se cumple que $E \leq C$.*

En otras palabras, la generalizacion menos general de dos cláusulas C y D es la generalización más especifica de las cláusulas C y D dentro del lattice generado por $\theta-$subsumption.

El algoritmo para evaluar el *lgg* entre dos términos se puede observar en la siguiente tabla:

Algoritmo:

Si L_1 y L_2 son dos términos o literales compatibles:

1. Sea $P_1 = L_1$ y $P_2 = L_2$

2. Encuentra dos términos, t_1 y t_2, en el mismo lugar en P_1 y P_2, tal que $t_1 \neq t_2$ y: o los dos tienen un nombre de función diferente o por lo menos uno de ellos es una variable.

3. Si no existe ese par, entonces acaba. $P_1 = P_2 = lgg(L_1, L_2)$

4. Si existe, escoge una variable X distinta de cualquier variable que ocurra en P_1 o P_2, y donde t_1 y t_2 aparezcan en el mismo lugar en P_1 y P_2, reemplazalos con X.

5. Vuelve al paso 2.

Ejemplo 4.1. lgg([a,b,c],[a,d,e]) = [a,X,Y]

Ejemplo 4.2. lgg (f(a,a), f(b,b)) = f (lgg(f(a,a)), lgg(f(b,b))) = lgg(V,V)

Ejemplo 4.3. lgg(padre(ana,maria), padre(ana,tomas)) = padre (ana,X)

Ejemplo 4.4. Dadas las siguientes cláusulas:

$$C_1 = hija(ana, juan) \leftarrow padre(juan, ana), femenino(ana), pequea(ana)$$

$$C_1 = hija(sara, pedro) \leftarrow padre(pedro, sara), femenino(sara), grande(sara)$$

La $lgg(C_1, C_2) = hija(X, Y) \leftarrow padre(Y, X), femenino(X)$.

Proposición 4.1. Sean dos cláusulas C_1 y C_2 la longitud del lgg de las cláusulas es, a lo más $|C_1| \times |C_2|$.

Proposición 4.2. El lgg entre literales es único (renombrando variables), pero entre cláusulas no necesariamente. Esto mismo se extiende para un conjunto de cláusulas.

Con respecto a los átomos, lgg es el dual de mgu. Es decir, dados dos términos f_1 y f_2, el lgg es su límite inferior más grande y el mgu su limite superior más bajo.

Por ejemplo, si tenemos los literales $L_1 = foo(a, f(a), g(X, b), Z)$ y $L_2 = foo(Y, f(Y), g(c, b), Z)$ podemos encontrar el lgg y mgu entre ellas; donde el lgg va a ser el literal más específico que subsume los dos literales y el mgu el literal más general subsumido por L_1 y L_2:

$$lgg(L_1, L_2) = foo(V, f(V), g(X, b), Z)$$

$$mgu(L_1, L_2) = foo(a, f(a), g(c, b), Z)$$

4.2. Generalización menos general relativa

En general, interesa encontrar generalizaciones de un conjunto de ejemplos en relación a cierta teoría o conocimiento del dominio. A continuación se introduce una definición de este tipo de generalización, dada por *Buntine*

Definición 4.2. *La generalización menos general relativa (rlgg) de dos cláusulas C_1 y C_2, es la generalización menos general(lgg)0 de ambas, relativas a cierto conocimiento base B. En otras palabras, si el conocimiento base B consiste en hechos, y K denota la conjunción de todos esos hechos ($K = \{h_1 \wedge \ldots \wedge h_n\}$), el rlgg de dos átomos A_1 y A_2 (ejemplos positivos), relativo al conocimiento de base K, es el definido como:*

$$rlgg(A_1, A_2) = lgg((K \rightarrow A_1), (K \rightarrow A_2))$$

Ejemplo 4.5. Dados dos ejemplos positivos $e_1 = hija(maria, ana)$ y $e_2 = hija(ivana, tomas)$ y el conocimiento de base B:

padre(ana,maria) padre(tomas,ivana) padre(tomas, ignacio) padre(ana,tomas)

femenino(maria) femenito(ivana) femenino(ana)

El *rlgg* será entonces: $rlgg(e_1, e_2) = lgg((K \rightarrow e_1), (K \rightarrow e_2))$ donde

$$K = \{padre(ana, maria), padre(tomas, ivana), padre(tomas, ignacio), padre(ana, tomas) \\ femenino(maria), femenito(ivana), femenino(ana)\}$$

Aplicándose las reglas de *lgg* se obtiene el siguiente resultado:

$$rlgg(e_1, e_2) = lgg((K \rightarrow e_1), (K \rightarrow e_2)) = femenino(X), padre(Y, X) \rightarrow hija(X, Y)$$

Rlgg, sin embargo, puede tener algunas conclusiones no intuitivas. Para mejorar esto, Buntine introdujo la noción de subsumción generalizada, el cual es un caso especial de rlgg, restringido a las cláusulas definitivas. La idea es que C_1 es mas general que C_2 con respecto a K, si cada vez que C_2 se puede usar (junto con K) para explicar algún ejemplo, C_1 también se puede usar. Esto lo podemos expresar más formalmente como sigue:

Definición 4.3. *Una cláusula $C_1 \equiv C_{1cabeza} \leftarrow C_{1cuerpo}$, subsume a otra cláusula $C_2 \equiv C_{2cabeza} \leftarrow C_{2cuerpo}$ con respecto a K si existe una sustitución mínima σ tal que $C_{1cabeza}\sigma = C_{2cuerpo}$ y para cualquier sustitución θ con constantes nuevas para C_2, se cumple que: $K \cup C_{2cuerpo}\theta \Vdash \exists(C_{1cuerpo}\sigma\theta)$.*

Ejemplo 4.6. Supongamos que queremos aprender la definición faldero y tenemos las siguientes cláusulas:

$$C_1 = faldero(fido) \leftarrow consentido(fido), pequeo(fido), perro(fido)$$
$$C_2 = faldero(morris) \leftarrow consentido(morris), gato(morris)$$

Entonces, $lgg(C_1, C_2) = faldero(X) \leftarrow consentido(X)$, lo cual no tiene por qué ser verdad. En cambio, si tenemos el siquiente conocimiento del dominio:

$$mascota(X) \leftarrow perro(X)$$
$$mascota(X) \leftarrow gato(X)$$
$$pequeo(X) \leftarrow gato(X)$$

Podemos añadir al cuerpo de C_1 y C_2 lo que se deduce del cuerpo de cada cláusula con el conocimiento del dominio. Esto es:

$$C_1' = faldero(fido) \leftarrow consentido(fido), pequeo(fido), perro(fido), mascota(fido)$$
$$C_2' = faldero(morris) \leftarrow consentido(morris), gato(morris), mascota(morris), pequeo(morris)$$

Entonces: $rlgg_T(C_1, C{-}_2) = lgg(C_1', C_2') =$

$$rlgg_T(C_1, C{-}_2) = lgg(C_1', C_2') =$$
$$faldero(X) \leftarrow consentido(X), mascota(X), pequeo(X)$$

que se acerca más a una definición plausible que toma en cuenta nuestro conocimiento del dominio.

4.3. Resolución inversa

En lógica de predicados de primer orden, la resolución requiere de sustituciones de variables por valores de atributos. Dicha resolución utiliza un operador de generalización basado en invertir la sustitución. La idea básica de la Resolución Inversa (*IR*) (introducida por *Muggleton*) como una técnica de generalización de ILP, es invertir la regla de resolución en inferencia deductiva, es decir, un método general que sirva para automatizar la deducción. Esta regla es consistente y completa para la deducción en primer orden.

Para esto necesitamos definir una sustitución inversa θ^{-1} que mapea términos a variables. Veamos esto en un ejemplo:

Ejemplo 4.7. Si $C_1 = hija(X, Y) \leftarrow femenino(X), padre(Y, X)$, la sustitución $\theta = \{X/ana, Y/juan\}$ nos da

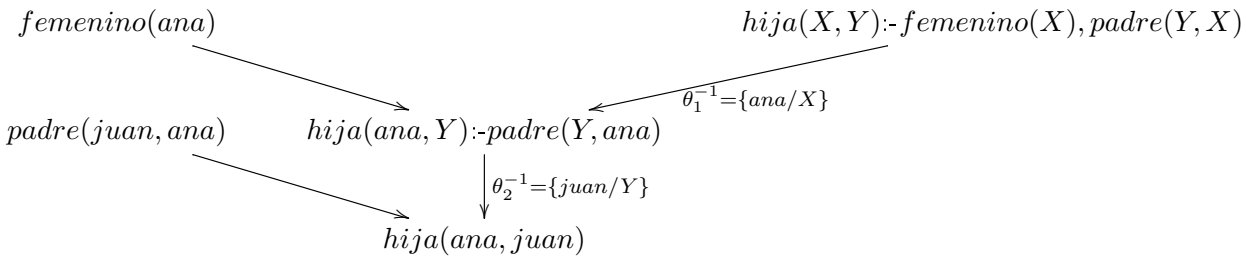
$$C_1' = C_1\theta = hija(ana, juan) \leftarrow femenino(ana), padre(juan, ana)$$

y la sustitución inversa $\theta^{-1} = \{ana/X, juan/Y\}$ nos da

$$C_1\theta^{-1} = C_1 = hija(X, Y) \leftarrow femenino(X), padre(Y, X)$$

De forma similar, si conocemos $hija(ana, juan)$ y $padre(juan, ana)$, podríamos aplicar un paso inverso de resolución para obtener $hija(ana, Y) \leftarrow padre(Y, ana)$, con una sustitución inversa $\theta_2^{-1} = \{juan/Y\}$.

Si además sabemos que $femenino(ana)$, podríamos aplicar otro proceso inverso de resolución para obetner $hija(X,Y) \leftarrow femenino(X), padre(Y,X)$ con $\theta_1^{-1} = \{ana/X\}$. Veamos este proceso gráficamente:

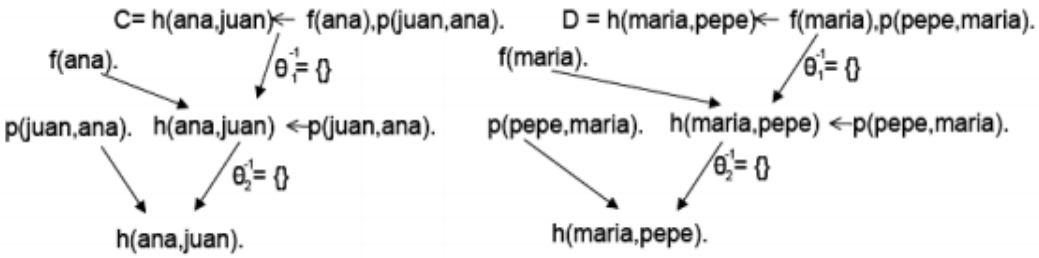


4.4. Relación entre resolución inversa y rlgg

Hay una forma de relacionar la resolución inversa y rlgg. Como sabemos, en cada paso de la resolución inversa, la solución puede variar dependiendo de la elección de la cláusula a ser resuelta y de la sustitución que empleemos. Para solucionar el problema del determinismo, Muggleton propone relacionar los conceptos de *rlgg* y resolución inversa.

Para ello, lo primero que se debe hacer es realizar la resolución inversa con la sustitución más específica (sustitución vacía) y tras esto, hacer el *lgg* de las cláusulas resultantes. En otras palabras: las generalizaciones más específicas con respecto al conocimiento del dominio (*rlgg*) son las generalizaciones más específicas (*lgg*) de los árboles inversos de derivación más específicos.

Ejemplo 4.8. En la figura se muestra un esquema común de generalización, donde "h" se refiere a hija," f" a femenino y "p" a padre.



Y se tiene que: $lgg(C,D) = h(Y,X) \Rightarrow f(X), p(Y,X)$

5. Métodos de general a específico (Top-Down)

Las técnicas de especialización investigan el espacio de hipótesis en forma Top-Down (es decir de general a específico), de las hipótesis más generales a las más específicas, usando un operador de especificación o como usualmente se le llama: operador de refinamiento.

Las técnicas básicas de especialización de ILP se sustentan sobre la filosofía de búsqueda de general a específico en grafos de refinamiento. La idea es ir añadiendo incrementalmente literales (condiciones o restricciones a reglas) siguiendo un proceso de búsqueda, generalmente tipo hill-climbing, usando una medida heurística. Una vez que se cumple el criterio de necesidad por la hipótesis actual, se eliminan los ejemplos positivos cubiertos y se empieza a generar una nueva cláusula. El proceso continua hasta que se cumple un cierto criterio de suficiencia.

5.1. Operador de Refinamiento

Un operador de refinamiento es una función p que mapea una cláusula C a un conjunto de cláusulas $p(C)$ las cuales son especializaciones (o refinamientos) de C. Un operador de refinamiento computa típicamente sólo el conjunto de especializaciones mínimas (más generales) de una cláusula y emplea básicamente dos funcionamientos sintácticos:

1. Aplicar una sustitución a la cláusula.
2. Agregar un literal al cuerpo de la cláusula.

Definición 5.1. *Un operador de refinamiento se dice completo sobre un conjunto de cláusulas, si se pueden obtener todas las cláusulas por medio de refinamientos sucesivos a partir de la cláusula vacía.*

Definición 5.2. *Un operador de refinamiento induce un orden parcial sobre el lenguaje de hipótesis, de modo que se puede hacer un grafo en donde nodos en capas inferiores son especializaciones de nodos en capas superiores.*

5.2. Sistema de Inferencia Modular (MIS)

Los sistemas de Inferencia Modular, denominados MIS, comienzan generando un operador de refinamiento completo para su lenguaje de hipótesis y tras esto, recorren su grafo de refinamiento hasta encontrar la hipótesis deseada.

Definición 5.3. *Un grafo de refinamiento es un grafo formado por un lattice sobre un conjunto de cláusulas. Este es direecionado y acíclico; sus nodos son cláusulas y sus arcos relaciones de especialización u operaciones básicas de refinamiento, entre las cuales están: substitución de variables con términos y agregación de un literal al cuerpo de una cláusula.*

Veamos un pseudocódigo que muestra los pasos básicos de un algoritmo MIS. Siendo L el lenguaje de cláusulas definidas y las definiciones de predicados del conocimiento del dominio B.

Algoritmo:

Inicializar la hipótesis H a un conjunto(posiblemente vacio) de cláusulas en L.
repetir
  Leer el siguiente ejemplo, positivo o negativo.
  repetir
    si exite una cobertura negativa del ejemplo **entonces**
      Borra las cláusulas incorrectas de H.
    si el ejemplo es positivo y no es cubierto por H **entonces**
      Desarrollar una cláusula que cubra el ejemplo y agregarla a H.
  hasta que H sea completo y consistente.
devuelve H.

En el primer si, C en H es incorrecta si C es responsable de que H sea inconsistente. En el segundo si, la búsqueda en el grafo de refinamiento comienza con la cláusula más general y continúa buscando refinamientos de cláusulas. En cada paso todos los mínimos refinamientos son generados y probados por cobertura. El proceso termina cuando la primera cláusula consistente es encontrada. Obsérvese el siguiente ejemplo:

Ejemplo 5.1. Supongamos que tenemos los siguientes ejemplos:
hija(fernanda,eduardo) (+) hija(camila,rodrigo) (+)
hija(eugenia, ernesto) (-) hija(valentina,roberto) (-)
Y el siguiente conocimiento del dominio:
femenino(fernanda) femenino(camila) femenino(eugenia) femenino(valentina)
padre(eduardo,fernanda) padre(rodrigo,camila)

Vamos a tomar el ejemplo positivo $e_1 = hija(fernanda, eduardo)$ para ser probado. La búsqueda por cláusulas comienza con la definición más general del predicado "hija", como sigue $C = hija(X, Y) \leftarrow$ donde el cuerpo queda vacío. Hasta que C cubra e_1, la hipótesis es inicializada a $H = \{C\}$.

El segundo ejemplo es $e_2 = hija(camila, rodrigo)$ que es positivo, por lo que es cubierto por H, asi que no se modifica la hipótesis.

El tercer ejemplo $e_3 = hija(eugenia, ernesto)$ es negativo, por lo que cubre los ejemplos negativos y anulamos la cláusula C de H.

Ahora se entra a la segunda sentencia “si” del algoritmo en orden de generación de nuevas cláusulas que cubran el primer ejemplo positivo e_1. Cuando C cubre el ejemplo negativo e_3, de acuerdo con el algoritmo, se genera entonces el conjunto de refinamientos de la forma: $p(C) = \{hija(X, Y) \leftarrow L\}$ donde L puede ser:

- Literales que tienen como argumentos: X=Y, femenino (X), femenino (Y), padre (X, X), padre (X, Y), padre (Y, X) o padre (Y, Y).

- Literales que introducen una nueva variable distinta denotada por Z : padre (X, Z), padre (Z, X), padre (Y, Z) o padre (Z, Y).

Se consideran ahora uno por uno de los refinamientos listados en L. El refinamiento $hija(X,Y) \leftarrow X = Y$ es considerado primero, pero es retractado porque no cubre el ejemplo positivo e_1.

Consideramos ahora el refinamiento $hija(X,Y) \leftarrow femenino(X)$ como cubre e_1, se retiene en la hipótesis actual H. Supóngase ahora que otro ejemplo negativo $e_4 = hija(valentina, roberto)$ es encntrado por el algoritmo de aprendizaje. La cláusula C' es inconsistente y es borrada de H, por lo que H vuelve a ser vacía.

Repetimos entonces el proceso de generación de la cláusula y el sistema vuelve a intentar buscar una cláusula que cubra a e_1. Se tiene que tanto $hija(X,Y) \leftarrow femenino(X)$ como $hija(X,Y) \leftarrow padre(Y,X)$ se retienen porque cubren e_1, pero no es consistente. Finalmente, los refinamientos de C' son considerados una vez que el sistema detecta que $hija(X,Y) \leftarrow padre(Y,X), femenino(X)$ cubre a e_1 y es consistente. Como esto sucede, la cláusula también cubre a e_2, y por tanto el algoritmo de aprendizaje detiene la búsqueda de cláusulas.

6. Implementación ALEPH

El primer sistema escogido es Aleph, sistema que sigue una estrategia Bottom-Up para construir las hipótesis. Esta elección está dada debido a que, con Aleph, es mas fácil la representación del conocimiento y se logra con pocos ejemplos encontrar una teoría completa y consistente con el conjunto de ejemplos.

Para realizar esta implementación fueron utilizados los archivos implementacion1.b, implementacion1.f y implementacion1.n. El archivo para implementar aleph se ha obtenido de: *https://github.com/friguzzi/aleph/tree/master/prolog* (existen dos archivos, de entre ellos se ha utilizado el que se llama $aleph_o rig.pl$) siguiendo las instrucciones que encontramos aqui: *http://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html♯SEC47*. Podemos ver algunas imagenes de la implementación en el Apéndice.

6.1. Descripción del algoritmo

Aleph (A Learning Engine for Proposing Hypotheses) es un sistema de ILP que implementa un algoritmo de cubrimiento secuencial, ya que aprende una a una las cláusulas y, en cada paso, elimina los ejemplos positivos que quedan cubiertos por la cláusula. Este algoritmo, constra de cuatro pasos importantes:

1. **Selección del ejemplo:** Selecciona un ejemplo a ser generalizado. Si no hay ninguno, se detiene, en otro caso se procede al siguiente paso.

2. **Construcción de la cláusula más específica:** Construye la cláusula más específica que trae consigo el ejemplo seleccionado, y que esté dentro de las restricciones del lenguaje proporcionado.

3. **Búsqueda:** Buscar una cláusula más general que la cláusula anterior.

4. **Eliminar la redundancia:** Agrega a la teoría actual la cláusula con mayor valor (o best score) encontrado y eliminar los ejemplos redundantes.

La solución brindada por el algoritmo del sistema Aleph está completa cuando se satisfacen las dos restricciones siguientes:

1. Tiene una exactitud mínima (número de ejemplos positivos cubiertos por encima del total de números de ejemplos cubiertos).

2. Cubre un número máximo de ejemplos negativos (0 por defecto).

6.2. Representación del problema

Para construir una teoría, Aleph requiere tres archivos: *implementacion1.b* (incluye el conocimiento base), *implementacion1.f* (incluye los ejemplos positivos) y *implementacion1.n* (incluye los ejemplos negativos). Vamos a estudiar los archivos utilizados para nuestro trabajo:

Archivo con conocimiento base

Es el archivo fundamental, ya que incluye el conocimiento base del problema. Vamos a verlo con detalle:

$:-modeh(1, expr(+parte, +parte)).$
$:-modeb(*, genero(+parte, -gen)).$ % nos va a decir el genero de 'parte'
$:-modeb(*, numero(+parte, -num)).$ % nos va a decir si 'parte' es singular o plural
$:-modeb(*, tipo(+parte, \sharp clasif)).$ % nos va a decir si 'parte' es articulo o sustantivo
$:-determination(expr/2, tipo/2).$
$:-determination(expr/2, genero/2).$
$:-determination(expr/2, numero/2).$
$:-set(clauselength, 10).$
$:-set(noise, 1).$

Permiten especificar, por medio de *modeh*, cuál o cuales son los predicados tipo header, y mediante *modeb* él o los predicados tipo body. Los primeros podrán ser usados como objetivo, mientras que los tipos últimos conforman el cuerpo de la definición del objetivo. Además se especifican los tipos de los argumentos de los predicados. Las declaraciones de determinación especifican la estructura de la hipótesis final. El primer *set* fija un máximo de 10 literales en una cláusula y el segundo sirve para variar la función de evaluación de las reglas, es este caso, $coverage = P - N$, donde P y N son el numero de elementos positivos o negativos cubiertos por la regla.

$parte(el).$	$parte(los).$	$parte(\text{niña}).$	$parte(\text{niña}s).$
$parte(la).$	$parte(las).$	$parte(\text{niño}).$	$parte(\text{niño}s).$
$parte(un).$	$parte(unos).$	$parte(mujer).$	$parte(mujeres).$
$parte(una).$	$parte(unas).$	$parte(hombre).$	$parte(hombres)$

$clasif(articulo).$	$gen(fem).$	$num(sing).$
$clasif(sustantivo).$	$gen(masc).$	$num(prural).$

$tipo(la, articulo).$	$genero(la, fem).$	$numero(la, sing).$
$tipo(una, articulo).$	$genero(una, fem).$	$numero(una, sing).$
$tipo(las, articulo).$	$genero(las, fem).$	$numero(las, prural).$
$tipo(unas, articulo).$	$genero(unas, fem).$	$numero(unas, prural).$
$tipo(\text{niña}, sustantivo).$	$genero(\text{niña}, fem).$	$numero(\text{niña}, sing).$
$tipo(mujer, sustantivo).$	$genero(mujer, fem).$	$numero(mujer, sing).$
$tipo(\text{niña}s, sustantivo).$	$genero(\text{niña}s, fem).$	$numero(\text{niña}s, prural).$
$tipo(mujeres, sustantivo).$	$genero(mujeres, fem).$	$numero(mujeres, prural).$
$tipo(el, articulo).$	$genero(el, masc).$	$numero(el, sing).$
$tipo(un, articulo).$	$genero(un, masc).$	$numero(un, sing).$
$tipo(los, articulo).$	$genero(\text{niño}, masc).$	$numero(\text{niño}, sing).$
$tipo(unos, articulo).$	$genero(hombre, masc).$	$numero(hombre, sing).$
$tipo(\text{niño}, sustantivo).$	$genero(\text{niño}s, masc).$	$numero(\text{niño}s, prural).$
$tipo(hombre, sustantivo).$	$genero(hombres, masc).$	$numero(hombres, prural).$
$tipo(\text{niño}s, sustantivo).$	$genero(los, masc).$	$numero(los, prural).$
$tipo(hombres, sustantivo).$	$genero(unos, masc).$	$numero(unos, prural).$

Archivo con ejemplos positivos

Nuestro archivo consta de los siguientes ejemplos:

$expr(una, mujer).$
$expr(el, hombre).$
$expr(las, nias).$
$expr(los, hombres).$

Archivo con ejemplos negativos

Nuestro archivo consta de los siguientes ejemplos:

$expr(hombre, nios).$	$expr(los, nio).$
$expr(mujeres, nias).$	$expr(hombre, el).$
$expr(un, el).$	$expr(nia, la).$
$expr(una, el).$	$expr(hombre, nio).$
$expr(un, nia).$	$expr(mujer, hombre).$
$expr(la, nio).$	$expr(el, el).$
$expr(la, mujeres).$	$expr(la, el).$
$\%expr(la, mujer).$	

El último ejemplo se añade en una segunda prueba para realizar un experimento si metemos un dato correcto en los ejemplos negativos.

6.2.1. Resultado: sin ejemplos erróneos

Tomando los archivos anteriores como punto de partida, cuando ejecutamos el algoritmo Aleph, obtenemos los siguientes resultados:

[positive examples left][0]
[estimated time to finish <secs>][0.0]

[theory]

[Rule 1] [Pos cover = 4 Neg cover = 0]
expr<A,B> :-
tipo <B,sustantivo>, tipo<A,sustantivo>, genero<B,C>, genero<A,C>, numero<B,D>, numero<A,D>.

[Training set performance]

Pred \ Actual	+	-	total
+	4	0	4
-	0	14	14
total	4	14	18

Accuracy = 1.0
[Training summary] [[4,0,0,14]]
[time taken] [0.140625]
[total clauses constructed] [64]
yes

Interpretación:

- La teoría resultante tiene una única regla, que es la siguiente: .^{A} y B forman una expresion si se cumple que: B es un sustantivo, A es un sustantivo, el género de A y B es el mismo, el número de A y B es el mismo"
- Del conjunto de entrenamiento se clasifican como positivos 4 ejemplos, los cuatro positivos y 0 negativos.
- Del conjunto de entrenamiento se clasifican como negativos 14 ejemplos, 0 positivos y 14 negativos.
- El rendimiento de la hipótesis es del 100 %

6.2.2. Resultado: con un ejemplo erróneo

Tomando los archivos anteriores como punto de partida, añadiendo el elento erróneo previamente mencionado al fichero de ejemplos negativos, cuando ejecutamos el algoritmo Aleph, obtenemos los siguientes resultados:

[positive examples left][0]
[estimated time to finish <secs>][0.0]

[theory]

[Rule 1] [Pos cover = 4 Neg cover = 1]
expr<A,B> :-
tipo <B,sustantivo>, tipo<A,artículo>, genero<B,C>, genero<A,C>, numero<B,D>, numero<A,D>.

[Training set performance]

Pred \ Actual	+	-	total
+	4	1	5
-	0	14	14
total	4	15	19

Accuracy = 0.9473684210526315
[Training summary] [[4,1,0,14]]
[time taken] [0.15625]
[total clauses constructed] [64]
yes

Interpretación:

- La teoría resultante tiene una única regla, que es la siguiente: .^{A} y B forman una expresion si se cumple que: B es un sustantivo, A es un artículo, el género de A y B es el mismo, el número de A y B es el mismo"
- Del conjunto de entrenamiento se clasifican como positivos 5 ejemplos, 4 positivos y 1 negativo.
- Del conjunto de entrenamiento se clasifican como negativos 14 ejemplos, 0 positivos y 14 negativos.
- El rendimiento de la hipótesis es del 94.74 %

6.2.3. Resultados que devuelven mas de una regla

Para este ejemplo vamos a utilizar un nuevo conjunto de datos. Vamos a presentar y comentar el archivo que contiene el conocimiento base (*implemantación3.b*), los archivos de ejemplos positivos (*implemantación3.f*) y negativos (*implemantación3.n*) se adjuntan ya que son demasiado largos para incluirlos aquí.

El archivo, al igual que el anterior comienza definiendo los predicados:

$$
\begin{aligned}
&:-modeh(*, grandparent(+person, -person)).\\
&:-modeh(*, parent(+person, -person)).\\
&:-modeb(*, mother(+person, -person)).\\
&:-modeb(*, father(+person, -person)).\\
&:-modeb(*, parent(+person, -person)).\\
&:-set(abduce, true).\\
&:-abducible(parent/2).\\
&:-determination(grandparent/2, father/2).\\
&:-determination(grandparent/2, parent/2).\\
&:-determination(grandparent/2, mother/2).\\
&:-dynamicgrandparent/2.
\end{aligned}
$$

La diferencia que existe con el código anterior es que en este, se especifica que queremos deducir las reglas necesarias para *'parent'*.

A continuación, introducimos los datos iniciales (nombres y parentescos):

$$
\begin{array}{ll}
person(bob). & person(jo).\\
person(dad(bob)). & person(dad(jo)).\\
person(mum(bob)). & person(mum(jo)).\\
person(dad(dad(bob))). & person(dad(dad(jo))).\\
person(dad(mum(bob))). & person(dad(mum(jo))).\\
person(mum(dad(bob))). & person(mum(dad(jo))).\\
person(mum(mum(bob))). & person(mum(mum(jo))).
\end{array}
$$

$$
\begin{array}{ll}
person(peter). & person(jane).\\
person(dad(peter)). & person(dad(jane)).\\
person(mum(peter)). & person(mum(jane)).\\
person(dad(dad(peter))). & person(dad(dad(jane))).\\
person(dad(mum(peter))). & person(dad(mum(jane))).\\
person(mum(dad(peter))). & person(mum(dad(jane))).\\
person(mum(mum(peter))). & person(mum(mum(jane))).
\end{array}
$$

Introducimos las definiciones de padre, madre y abuelo:

$$father(dad(X), X) :- \\ person(X).$$

$$mother(mum(X), X) :- \\ person(X).$$

$$grandparent(X, Z) :- \\ person(X), \\ person(Y), \\ X \backslash = Y, \\ parent(X, Y), \\ person(Z), \\ Y \backslash = Z, \\ parent(Y, Z).$$

Las reglas que pretendemos buscar con la implementación son:

$$\% \quad parent(X, Y) :- \quad father(X, Y).$$
$$\% \quad parent(X, Y) :- \quad mother(X, Y).$$

Cuando ejecutamos el algoritmo Aleph, obtenemos los siguientes resultados:

[theory]

[Rule 1] [Pos cover = 4 Neg cover = 1]
parent(dad(dad(bob)),mum(bob))

[Rule 2] [Pos cover = 4 Neg cover = 1]
parent(A,B) :- mother(A,B)

[Rule 3] [Pos cover = 4 Neg cover = 1]
parent(A,B) :- father(A,B)

[Training set performance]

		Actual		
Pred		+	-	total
	+	16	0	16
	-	0	768	768
	total	16	768	784

Accuracy = 1
[Training summary] [[16,0,0,14]]
[time taken] [2.90625]

[total clauses constructed] [216]

Interpretación:

- La teoría resultante tiene tres reglas, que son:
  - Regla 1: *'Es progenitor el padre del padre de bob de la madre de bob'*. Podemos observar que la regla parece no ser correcta, pero como no hay ningun ejemplo negativo que la refute, aparece en nuestro conjunto de reglas.
  - Regla 2: *'A es progenitor de B si A es madre de B'*
  - Regla 2: *'A es progenitor de B si A es madre de B'*
- Las reglas 2 y 3 son las que estabamos buscando desde el principio.
- Del conjunto de entrenamiento se clasifican como positivos 16 ejemplos, 16 positivos y 0 negativos.
- Del conjunto de entrenamiento se clasifican como negativos 768 ejemplos, 0 positivos y 768 negativos.
- El rendimiento de la hipótesis es del 100 % (basandonos en los ejemplos positivos y negativos dados.)

7. Apéndice

Implementación 1 de Aleph

Para cargar el fichero:

```
?- working_directory(_,'C:/Users/SARA/Documents/MASTER/PL- PROGRAMACION LOGICA/ALEPH').
true.

?- load(aleph).
Correct to: "win_menu:load(aleph)"? yes


A L E P H
Version 5
Last modified: Sun Mar 11 03:25:37 UTC 2007

Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html

true.

?- read_all(implementacion1).
[consulting pos examples] [implementacion1.f]
[consulting neg examples] [implementacion1.n]
true .

?- induce.
```

Los resultados tras la implementación:

```
[4/2]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(A,C), numero(B,D),
   numero(A,D).
[4/2]
expr(A,B) :-
   tipo(A,articulo), genero(B,C), genero(A,C), numero(B,D),
   numero(A,D).
[4/2]
expr(A,B) :-
   tipo(B,sustantivo), genero(B,C), genero(A,C), numero(B,D),
   numero(A,D).
[4/2]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[4/0]
[----------------------------------]
[found clause]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[pos cover = 4 neg cover = 0] [pos-neg] [4]
[clause label] [[4,0,7,4]]
[clauses constructed] [64]
[----------------------------------]
[clauses constructed] [64]
[search time] [0.140625]
[best clause]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[pos cover = 4 neg cover = 0] [pos-neg] [4]
[atoms left] [0]
[positive examples left] [0]
[estimated time to finish (secs)] [0.0]

[theory]

[Rule 1] [Pos cover = 4 Neg cover = 0]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).

[Training set performance]
          Actual
        +          -
     +  4          0          4
Pred
     -  0         14         14

        4         14         18

Accuracy = 1
[Training set summary] [[4,0,0,14]]
[time taken] [0.140625]
[total clauses constructed] [64]
true.
```

Implementación 2 de Aleph

Para cargar el fichero:

```
?- working_directory(_,'C:/Users/SARA/Documents/MASTER/PL- PROGRAMACION LOGICA/ALEPH').
true.

?- load(aleph).
Correct to: "win_menu:load(aleph)"? yes


A L E P H
Version 5
Last modified: Sun Mar 11 03:25:37 UTC 2007

Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html

true.

?- read_all(implementacion2).
[consulting pos examples] [implementacion2.f]
[consulting neg examples] [implementacion2.n]
true .

?- induce.
```

Los resultados tras la implementación:

```
[4/3]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(A,C), numero(B,D),
   numero(A,D).
[4/3]
expr(A,B) :-
   tipo(A,articulo), genero(B,C), genero(A,C), numero(B,D),
   numero(A,D).
[4/3]
expr(A,B) :-
   tipo(B,sustantivo), genero(B,C), genero(A,C), numero(B,D),
   numero(A,D).
[4/3]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[4/1]
[----------------------------------------]
[found clause]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[pos cover = 4 neg cover = 1] [pos-neg] [3]
[clause label] [[4,1,7,3]]
[clauses constructed] [64]
[----------------------------------------]
[clauses constructed] [64]
[search time] [0.140625]
[best clause]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).
[pos cover = 4 neg cover = 1] [pos-neg] [3]
[atoms left] [0]
[positive examples left] [0]
[estimated time to finish (secs)] [0.0]

[theory]

[Rule 1] [Pos cover = 4 Neg cover = 1]
expr(A,B) :-
   tipo(B,sustantivo), tipo(A,articulo), genero(B,C), genero(A,C),
   numero(B,D), numero(A,D).

[Training set performance]
          Actual
         +          -
     +   4          1          5
Pred
     -   0         14         14

         4         15         19

Accuracy = 0.9473684210526315
[Training set summary] [[4,1,0,14]]
[time taken] [0.15625]
[total clauses constructed] [64]
true.
```

Implementación 3 de Aleph

Para cargar el fichero:

```
?- working_directory(_,'C:/Users/SARA/Documents/MASTER/PL- PROGRAMACION LOGICA/ALEPH').
true.

?- load(aleph).
Correct to: "win_menu:load(aleph)"? yes


A L E P H
Version 5
Last modified: Sun Mar 11 03:25:37 UTC 2007

Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html

true.

?- read_all(implementacion3).
[consulting pos examples] [implementacion3.f]
[consulting neg examples] [implementacion3.n]
true .

?- induce.
```

Los resultados tras la implementación:

```
[best clause]
parent(mum(mum(jane)),bob).
[pos cover = 1 neg cover = 0] [pos-neg] [1]
[atoms left] [0]
[positive examples left] [0]
[estimated time to finish (secs)] [0.0]

[theory]

[Rule 1] [Pos cover = 0 Neg cover = 0]
parent(dad(dad(bob)),mum(bob)).

[Rule 2] [Pos cover = 0 Neg cover = 0]
parent(A,B) :-
   mother(A,B).

[Rule 3] [Pos cover = 0 Neg cover = 0]
parent(A,B) :-
   father(A,B).

[Training set performance]
            Actual
          +       -
      +  16       0          16
Pred
      -  0      768         768

         16       768         784

Accuracy = 1
[Training set summary] [[16,0,0,768]]
[time taken] [2.90625]
[total clauses constructed] [216]
true.

?-
```

Referencias

[1] Daniel Schafer. *Inductive Logic Programming: Theory and Methods* , 2018.

[2] Stephen MUGGLETON. *Inductive Logic Programming. The Turing Institute*,1990.

[3] Juan María Hernandez. *¿Qué es la programación lógica?* ,2013.