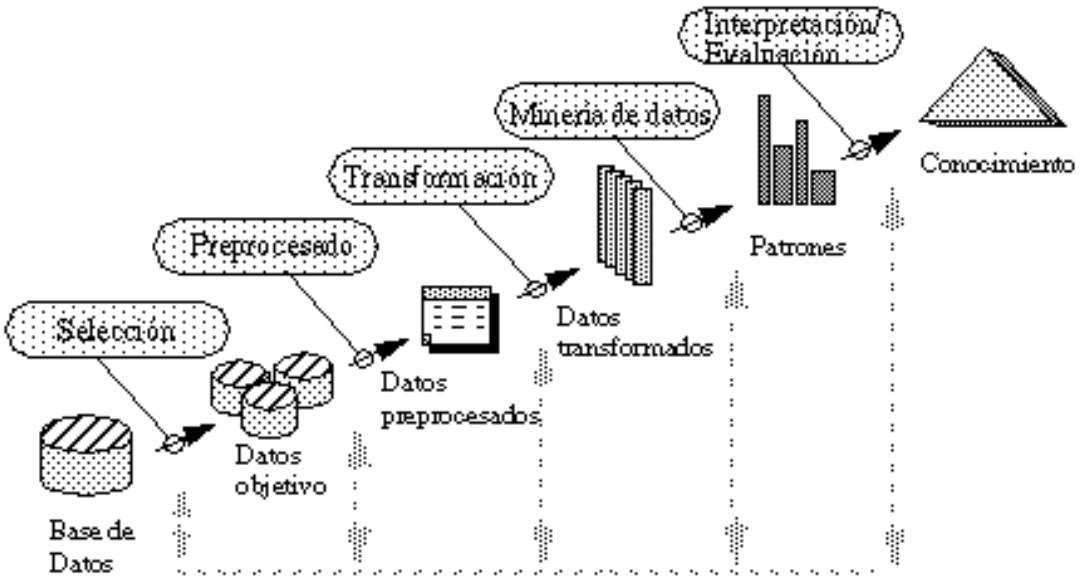
Minería de datos

José A. Alonso Jiménez y Miguel A. Gutiérrez Naranjo
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

1 Introducción

La minería de datos constituye un paso fundamental en el descubrimiento de conocimiento en bases de datos (KDD). Según una de sus definiciones usuales (Fayyad, Piatetsky–Shapiro y Smyth, 1996), KDD es el proceso no trivial de identificar en los datos estructuras válidas, novedosas, potencialmente útiles y en última instancia comprensibles. El proceso de KDD consta de los pasos señalados en el siguiente gráfico



es decir,

1. Determinar los objetivos del proceso de KDD.
2. Crear el conjunto de datos objetivo.
3. Preprocesar los datos.
4. Transformar y reducir de los datos.
5. *Minería de datos.*
6. Interpretar el conocimiento descubierto y posible iteración.
7. Consolidar y usar el conocimiento descubierto.

Como puede observarse, en el KDD intervienen distintas disciplinas: bases de datos, estadística, inteligencia artificial, aprendizaje automático, visualización de datos, etc.

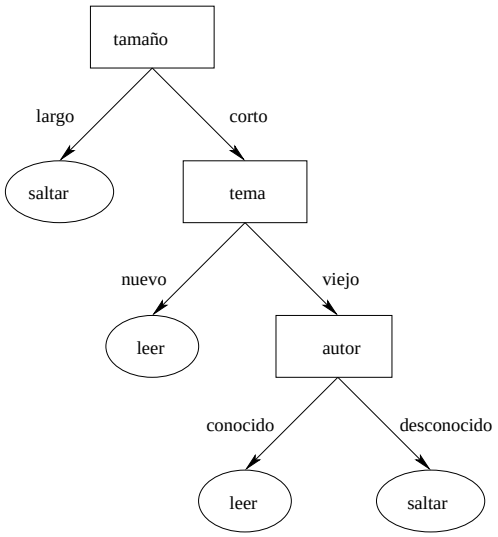
2 Minería de datos y árboles de decisión

Una forma de representación del conocimiento es mediante árboles de decisión. En esta sección ilustraremos cómo puede representarse un conjunto de datos mediante un árbol de decisión y un algoritmo que dado una tabla de datos genere el árbol de decisión más "simple" que explique los datos. Lo haremos aplicándolo al siguiente caso. Supongamos que hemos observado el comportamiento de un usuario cuando recibe un correo electrónico y hemos resumido las observaciones en la siguiente tabla:

Ejemplo	Acción	Autor	Tema	Tamaño	Sitio
e1	saltar	conocido	nuevo	largo	casa
e2	leer	desconocido	nuevo	corto	trabajo
e3	saltar	desconocido	viejo	largo	trabajo
e4	saltar	conocido	viejo	largo	casa
e5	leer	conocido	nuevo	corto	casa
e6	saltar	conocido	viejo	largo	trabajo
e7	saltar	desconocido	viejo	corto	trabajo
e8	leer	desconocido	nuevo	corto	trabajo
e9	saltar	conocido	viejo	largo	casa
e10	saltar	conocido	nuevo	largo	trabajo
e11	saltar	desconocido	viejo	corto	casa
e12	saltar	conocido	nuevo	largo	trabajo
e13	leer	conocido	viejo	corto	casa
e14	leer	conocido	nuevo	corto	trabajo
e15	leer	conocido	nuevo	corto	casa
e16	leer	conocido	viejo	corto	trabajo
e17	leer	conocido	nuevo	corto	casa
e18	leer	desconocido	nuevo	corto	trabajo

en el ejemplo 1, el autor del correo es conocido, el tema es nuevo, el tamaño es largo, el sitio donde lo lee es en su casa y la acción que realiza es saltárselo.

La acción del usuario recogida en los datos anteriores puede explicarse mediante el siguiente árbol de decisión:



donde las hojas representan las posibles clasificaciones (saltar o leer), los nodos internos representan los distintos atributos (longitud, tema y autor) y las etiquetas de los arcos representan los posibles valores de los atributos. Las ramas del árbol representan las distintas decisiones, por ejemplo, la segunda rama significa que si el tamaño es corto y el tema es nuevo la acción que realiza es leer (esta rama engloba los ejemplos 2, 5, 8, 14, 15, 17 y 18).

Utilizando el ejemplo, vamos a ilustrar un procedimiento de generación del árbol de decisión. La primera cuestión que nos planteamos es si todos los ejemplos están en la misma clase; es decir, si en todos realiza la misma acción. La respuesta es negativa (hay 9 casos en los que lee el correo (ejemplos positivos) y 9 en que lo salta (ejemplos negativos)). Para apreciar lo lejos que estamos del árbol de decisión, podemos usar como medida de la información la obtenida mediante la fórmula

$$I(P,N) = \begin{cases} 0, & \text{si } N*P = 0; \\ -\frac{P}{T}\log_2\frac{P}{T} - \frac{N}{T}\log_2\frac{N}{T}, & \text{si } N*P \neq 0. \end{cases}$$

donde P es el número de ejemplos positivos, N es el número de ejemplos negativos y T es el número total de ejemplos. En nuestro caso, resulta

$$I(9,9) = -\frac{9}{18}\log_2\frac{9}{18} - \frac{9}{18}\log_2\frac{9}{18} = 1$$

Puesto que todos los ejemplos no están en la misma clase debemos de elegir un atributo para dividir los ejemplos. Al elegir un atributo se forman dos clases (ya que estamos considerando atributos con sólo dos posibles valores). La información correspondiente a la división por un atributo se calcula mediante la fórmula

$$I(Atributo) = \frac{N_1 * I_1 + N_2 * I_2}{N_1 + N_2},$$

donde N_1 es el número de ejemplos en la clase 1, N_2 es el número de ejemplos en la clase 2, I_1 es la cantidad de información en los ejemplos de la clase 1 e I_2 es la cantidad de información en los ejemplos de la clase 2.

En nuestro caso, la división por el atributo autor genera la siguiente distribución de los ejemplos

```
             | leer               | saltar
-------------+--------------------+---------------------
conocido     | 05,13,14,15,16,17  | 01,04,06,09,10 12
desconocido  | 02,08,18           | 03,07,11
```

La clase 1 es la formada por los correos de autor conocido y la clase 2, las de autor desconocido. Por tanto,

$$\begin{array}{ll} N_1 & = 12 \\ N_2 & = 6 \\ I_1 & = I(6,6) = -\frac{6}{12}\log_2\frac{6}{12} - \frac{6}{12}\log_2\frac{6}{12} = 1 \\ I_2 & = I(3,3) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1 \\ I(autor) & = \frac{12*1+6*1}{12+6} = 1 \end{array}$$

Procediendo de la misma manera con los restantes atributos obtenemos la siguientes informaciones:

Atributo	Información
autor	1
tema	0.850
longitud	0.418
sitio	1

El atributo que aporta una mayor ganancia de información es longitud. Por tanto, elegimos dividir por el tamaño y nos queda por clasificar dos conjuntos de ejemplos: los artículos largos y los artículos cortos. Los artículos largos pertenecen a la misma clase de acción (siempre se saltan) y corresponden a una hoja del árbol. No pasa lo mismo con los cortos y hay que repetir el proceso con estos ejemplos.

En resumen, el problema que hemos estudiado consiste en: Dado un conjunto de ejemplos (donde cada ejemplo es una lista de atributos y valores) y uno de dicho atributos (el atributo objetivo), construir un árbol de decisión que explique los ejemplos de la mejor manera posible. El procedimiento que hemos utilizado es el siguiente:

1. Si todos los ejemplos coinciden (es decir, si todos tienen el mismo valor en el atributo objetivo, entonces el árbol de decisión se reduce a una hoja con dicho valor.

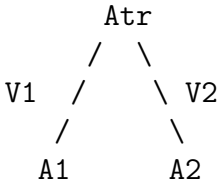
2. En caso contrario,

   (a) elegir el atributo (`Atr`) con mayor ganancia de información,

   (b) sea `C1` el conjunto de ejemplos donde el atributo seleccionado toma un valor (`V1`) y `C2` el de los ejemplos donde toma el otro valor (`V2`),

(c) sea `A1` el árbol construido aplicando el procedimiento a los ejemplos de `C1` y con el mismo atributo objetivo y `A2` el árbol construido aplicando el procedimiento a los ejemplos de `C2`,

(d) el árbol de decisión es



El anterior algoritmo es el ID3 (Interactive Dichotomizer), fue introducido por Quinlan en 1986 y se encuadra dentro de la familia de algoritmos TDIDT (Top–Down Induction of Decision Tree) a la que también pertenece al C4.5. Para una presentación detallada de estos algoritmos y sus aplicaciones puede consultarse el libro de J.R. Quinlan *C4.5: Programs for Machine Learning* (Morgan Kaufmann, 1993).

La utilización de los árboles de decisión presenta una serie de limitaciones:

- la representación formal es limitada ya que el lenguaje de pares atributo–valor es equivalente al de la lógica proposicional;
- los árboles tienden a ser demasiado grandes en aplicaciones reales y
- es difícil utilizar conocimiento base.

Una forma de superar dichas dificultades es la utilización de reglas. Las reglas pueden extraerse de los árboles de decisión, por ejemplo, del árbol de las observaciones sobre las acciones con el correo electrónico se extraen las siguientes reglas

```
si longitud(E)=largo
entonces accion(E)=saltar

si longitud(E)=corto
   y trama(E)=nuevo
entonces accion(E)=leer
```

```
si longitud(E)=corto
   y trama(E)=viejo
   y autor(E)=conocido
entonces accion(E)=leer

si longitud(E)=corto
   y trama(E)=viejo
   y autor(E)=desconocido
entonces accion(E)=saltar
```

donde cada una de las reglas corresponde a una rama del árbol. Pero las reglas pueden generarse sin necesidad del árbol como veremos en la siguiente sección.

Programación lógica inductiva

La programación lógica inductiva (PLI) nace de la combinación de la programación lógica con el aprendizaje automático. El problema fundamental de la PLI consiste en dado un conjunto de ejemplos positivos ($E^{\oplus}$), un conjunto de ejemplos negativos ($E^{\ominus}$), un conocimiento base del dominio de estudio (T) y un lenguaje de para expresar las hipótesis (L) tales que se verifican las siguientes condiciones

- Necesidad a priori: $(\exists e^{\oplus} \in E^{\oplus})[T \nvdash e^{\oplus}]$
- Consistencia a priori: $(\forall e^{\ominus} \in E^{\ominus})[T \nvdash e^{\ominus}]$

encontrar un conjunto finito $H \subset L$ tal que se cumplan

- Suficiencia a posteriori: $(\forall e^{\oplus} \in E^{\oplus})[T \bigcup H \vdash e^{\oplus}]$
- Consistencia a posteriori: $(\forall e^{\ominus} \in E^{\ominus})[T \bigcup H \nvdash e^{\ominus}]$

Existen principalmente dos vías para resolver el problema:

- descendente: comenzar con la hipótesis más general y particularizarla para resolver el problema y
- ascendente: comenzar con la hipótesis más específica y generalizarla para resolver el problema.

En los próximos apartados comentaremos los dos vías mediante un sistema representativo de cada una: FOIL para la descendente y Progol para la ascendente.

2.1 Programación lógica inductiva descendente con FOIL

En este apartado vamos a ver varios ejemplos de uso de FOIL y comentar el algoritmo.

En el primer caso de estudio partimos de dos ejemplos positivos:

```
padre(carlos,juan).
padre(carlos,eva).
```

(que indican que Carlos es el padre de Juan y Eva), de dos ejemplos negativos

```
no(padre(carlos,aurora)).
no(padre(aurora,juan)).
```

(que indican que Carlos no es el padre de Aurora y que Aurora no es el padre de Juan), y del siguiente conocimiento básico del dominio de las relaciones familiares

```
hombre(carlos).              hombre(juan).
mujer(eva).                  mujer(aurora).
progenitor(carlos,juan).     progenitor(carlos,eva).
progenitor(aurora,juan).     progenitor(aurora,eva).
```

(que indican que Carlos y Juan son hombres, que Eva y Aurora son mujeres, que Carlos es un progenitor de Juan y Eva y Aurora también lo es). Supongamos que hemos escrito un fichero (`familia.pl`) con la anterior información junto con los siguientes parámetros de FOIL

```
foil_predicates([padre/2,hombre/1,mujer/1,progenitor/2]).
foil_cwa(false).
foil_use_negations(false).
foil_det_lit_bound(0).
```

en los que se describen el lenguaje (compuesto de las relaciones binarias `padre` y `progenitor` y de los predicados `hombre` y `mujer`) y se indica que no use la hipótesis del mundo cerrado ni use información negativa en el cuerpo de las reglas inducidas. Nuestro objetivo es usar FOIL para inducir la definición de la relación `padre`.

La sesión de inducción es la siguiente (que comentaremos a continuación)

```
?- [foil,familia].
Yes

?- foil(padre/2).
% 1
Uncovered positives: [padre(carlos,juan),padre(carlos,eva)]

% 2
Adding a clause: padre(A,B).

% 3
Covered positives: [padre(carlos,juan),padre(carlos,eva)]
Covered negatives: [padre(carlos,aurora),padre(aurora,juan)]

% 4
Specializing current clause:
Ganancia:  0.830 Cláusula: padre(A,B):-hombre(A)
....
Ganancia:  0.000 Cláusula: padre(A,B):-progenitor(B,B)
Specializing current clause: padre(A,B) :- hombre(A).

% 5
Covered positives: [padre(carlos,juan),padre(carlos,eva)]
Covered negatives: [padre(carlos,aurora)]

% 6
Specializing current clause:
Ganancia:  0.000 Cláusula: padre(A,B):-hombre(A),hombre(A)
...
Ganancia:  1.170 Cláusula: padre(A,B):-hombre(A),progenitor(A,B)
...
Ganancia:  0.000 Cláusula: padre(A,B):-hombre(A),progenitor(B,B)
Clause found: padre(A,B) :- hombre(A),progenitor(A,B).

% 7
Found definition: padre(A,B) :- hombre(A),progenitor(A,B).
```

En primer lugar comenzamos la sesión con el intérprete Prolog y cargamos los

ficheros que contienen a FOIL y nuestra descripción del problema (`[foil,familia]`), a continuación le indicamos la relación que deseamos aprender (`foil(padre/2)`) y comienza la ejecución del procedimiento de FOIL:

1. si no considera ninguna regla, existen ejemplos positivos no cubiertos (por ejemplo, Carlos es padre de Juan)

2. introduce la regla más general posible para explicarlo: `padre(A,B)`, que significa que todos son padres de todos

3. con la regla introducida se cubre todos los ejemplos positivos, pero cubre ejemplos negativos (por ejemplo, Aurora no es padre de Juan)

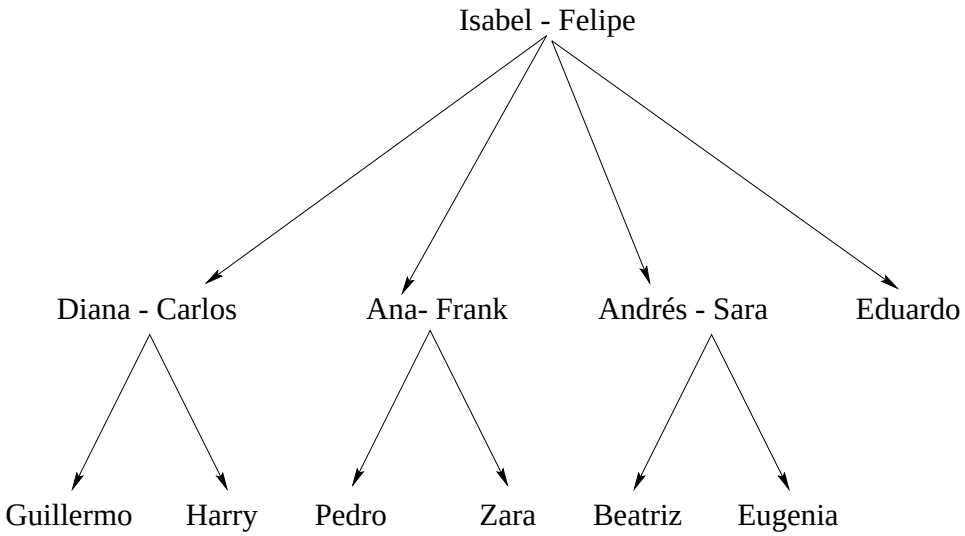
4. particulariza la regla añadiéndole una condición y elige la mejor particularización: `padre(A, B) :- hombre(A)`, que significa que si `A` es un hombre y `B` es cualquiera, entonces `A` es padre de `B`-

5. con la nueva regla se sigue cubriendo todos los ejemplos positivos y cubre un ejemplo negativo (porque aunque Carlos es un hombre, no es padre de Aurora)

6. particulariza la regla añadiéndole otra condición y elige la mejor particularización: `padre(A, B) :- hombre(A), progenitor(A, B)`, que significa que si `A` es un hombre que es progenitor de `B`, entonces `A` es padre de `B`.

7. con la nueva regla se cubren los ejemplos positivos y ninguno de los negativos, con lo que se concluye el proceso de aprendizaje.

En el caso anterior usamos ejemplos negativos, pero podemos usar sólo ejemplos positivos con la hipótesis del mundo cerrado (las relaciones básicas no declaradas son falsas). Vamos a verlo en el siguiente caso también de relaciones familiares. Consideremos el árbol genealógico



La información del árbol genealógico la podemos representar mediante la siguiente base de datos

```
padre(felipe,carlos).       padre(felipe,ana).      ...
madre(isabel,carlos).       madre(isabel,ana).      ...
abuelo(felipe,guillermo).   abuelo(felipe,harry).   ...
```

A partir de los datos anteriores nuestro objetivo es aprender la relación `abuelo`, utilizando el algoritmo de FOIL. El proceso de aprendizaje es el siguiente:

1. si no considera ninguna regla, existen ejemplos positivos no cubiertos (por ejemplo, Felipe es abuelo de Guillermo)

2. introduce la regla más general posible para explicarlo: `abuelo(A,B)`, que significa que todos son abuelos de todos

3. con la regla introducida se cubre los 6 ejemplos positivos, pero cubre ejemplos negativos introducidos por la hipótesis del mundo cerrado (por ejemplo, Ana no es abuelo de Andrés)

4. particulariza la regla añadiéndole una condición y elige la mejor particularización: `abuelo(A,B) :- padre(A,C)` que significa que si `A` es abuelo si `A` es padre

5. con la nueva regla se sigue cubriendo los 6 ejemplos positivos pero sigue cubriendo ejemplos negativos (por ejemplo, Andrés no es abuelo de Ana aunque es padre de Beatriz)

6. particulariza la regla añadiéndole otra condición y elige la mejor particularización: `abuelo(A,B) :- padre(A,C),padre(C,B).`, que significa que si `A` es abuelo de `B` si es el padre del padre de `B`)

7. con la nueva regla se cubren 4 ejemplos positivos y ninguno de los negativos, con lo que se añade a la teoría

8. existen 2 ejemplos positivos no cubiertos con la teoría actual: Felipe es abuelo de Pedro y Zara, aunque no es el padre de ninguno de sus padres

9. se repite el proceso con los 2 ejemplos positivos no cubiertos

10. introduce la regla más general posible para explicarlo: `abuelo(A,B)`, que significa que todos son abuelos de todos

11. con la regla introducida se cubre los 2 ejemplos positivos, pero cubre ejemplos negativos introducidos por la hipótesis del mundo cerrado (por ejemplo, Ana no es abuelo de Andrés)

12. particulariza la regla añadiéndole una condición y elige la mejor particularización: `abuelo(A,B) :- padre(A,C)` que significa que si `A` es abuelo si `A` es padre

13. con la nueva regla se sigue cubriendo los 2 ejemplos positivos pero sigue cubriendo ejemplos negativos (por ejemplo, Andrés no es abuelo de Ana aunque es padre de Beatriz)

14. particulariza la regla añadiéndole otra condición y elige la mejor particularización: `abuelo(A,B) :- padre(A,C),madre(C,B).`, que significa que si `A` es abuelo de `B` si es el padre de la madre de `B`)

15. con la nueva regla se cubren los ejemplos positivos y ninguno de los negativos, con lo que se concluye el proceso de aprendizaje.

16. la definición aprendida de la relación `abuelo` es

```
abuelo(A,B) :- padre(A,C),madre(C,B).
abuelo(A,B) :- padre(A,C),padre(C,B).
```

que significa que `A` es abuelo de `B` si es el padre o la madre de `B`.

Para una información más detallado de FOIL y su aplicación a bases de conocimiento con incertidumbre puede consultarse la tesis doctoral de A.J. Gómez *Inducción de conocimiento con incertidumbre en bases de datos relacionales borrosas* (Univ. Politécnica de Madrid, 1998).

3 Programación lógica inductiva ascendente con Progol

En este apartado vamos a ilustrar el proceso de aprendizaje ascendente: partir de la hipótesis más específica y generalizarla hasta obtener la explicación. Lo haremos usando el sistema Progol con el ejemplo anterior de los correos electrónicos.

La base de datos correspondiente a las 18 observaciones se representa mediante los siguiente hechos

```
autor(e1,conocido). ... autor(e18,desconocido).
tema(e1,nuevo).     ... tema(e18, nuevo).
longitud(e1,largo). ... longitud(e18,corto).
sitio(e1,casa).     ... sitio(e18, trabajo).

accion(e1,saltar).  ... accion(e18,leer).
```

El fichero se amplía con parámetros indicando que

- sólo se usan ejemplos positivos,
- la forma de las reglas de explicación tienen como conclusión
  `accion(Ejemplo, Tipo_de_accion)`
  y en las condiciones pueden aparecer
  - `autor(Ejemplo, Tipo_de_autor)`,
  - `autor(Ejemplo, Tipo_de_autor)`,

  - `tema(Ejemplo, Tipo_de_tema))`,
  - `longitud(Ejemplo, Tipo_de_longitud)` y
  - `sitio(Ejemplo, Tipo_de_sitio)`

- los tipos de los predicados: los tipos de autor son conocido y desconocido, ...
- restricción de que las acciones leer y saltar son incompatibles

Partiendo de dicha información, el proceso de aprendizaje de Prolog es

1. empieza con la teoría vacía: $T = \emptyset$
2. elige el primer ejemplo no cubierto por la teoría T: e_1
3. construye la regla más específica que junto T cubre el ejemplo e_1:

   ```
   accion(A,saltar) :-
     autor(A,conocido), tema(A,nuevo),
     longitud(A,largo), sitio(A,casa).
   ```

   que significa que si el autor de correo es conocido, el tema es nuevo, es correo es largo y está en su casa, se salta el correo
4. busca la mejor generalización (eliminando condiciones) de la cláusula anterior (`accion(A,saltar) :- longitud(A,largo)`, que significa que se salta los artículos largos) y se lo añade a la teoría T
5. con T explica 7 ejemplos, pero quedan ejemplos sin cubrir
6. elige el primer ejemplo no cubierto por la teoría T: e_2
7. construye la regla más específica que junto T cubre el ejemplo e_2:

   ```
   accion(A,leer) :-
     autor(A,desconocido), tema(A,nuevo),
     longitud(A,corto), sitio(A,trabajo).
   ```

8. busca la mejor generalización (eliminando condiciones) de la cláusula anterior: `accion(A,leer) :- tema(A,nuevo), longitud(A,corto)` y se lo añade a la teoría T

9. con T explica 14 ejemplos, pero quedan ejemplos sin cubrir

10. elige el primer ejemplo no cubierto por la teoría T: e_7

11. construye la regla más específica que junto T cubre el ejemplo e_7:

```
accion(A,saltar) :-
  autor(A,desconocido), tema(A,viejo),
  longitud(A,corto), sitio(A,trabajo).
```

12. busca la mejor generalización (eliminando condiciones) de la cláusula anterior: `accion(A,leer) :- autor(A,desconocido), tema(A,viejo)` y se lo añade a la teoría T

13. con T explica 16 ejemplos, pero quedan ejemplos sin cubrir

14. elige el primer ejemplo no cubierto por la teoría T: e_{13}

15. construye la regla más específica que junto T cubre el ejemplo e_{13}:

```
accion(A,leer) :-
  autor(A,conocido), tema(A,viejo),
  longitud(A,corto), sitio(A,casa).
```

16. busca la mejor generalización (eliminando condiciones) de la cláusula anterior: `accion(A,leer) :- autor(A,conocido), longitud(A,corto)` y se lo añade a la teoría T

17. con T explica los 18 ejemplos

18. la teoría T encontrada es

```
accion(A,saltar) :- longitud(A,largo).
accion(A,leer) :- tema(A,nuevo), longitud(A,corto).
accion(A,saltar) :- autor(A,desconocido), tema(A,viejo).
accion(A,leer) :- autor(A,conocido), longitud(A,corto).
```

Comparando las reglas obtenidas con Progol con las obtenidas a partir del ID3 observamos que ambas explican el mismo árbol de decisión pero que las reglas obtenidas por Progol son más simples.

Para más información sobre Progol puede consultarse el trabajo de S. Muggleton y J. Firth *CProgol4.4: a tutorial introduction* (en "Inductive Logic Programming and Knowledge Discovery in Databases". Springer–Verlag, 2000)

4 Otros métodos de minería de datos

Además de las formas de minería de datos consideradas existen métodos basados en redes neuronales, razonamiento basado en casos, redes bayesianas y algoritmos genéticos. Las redes neuronales se trata en los trabajos de J. Muñoz y V. Quesada. Para más información sobre los otros métodos puede consultarse la tesis de A. J. Gómez citada anteriormente y el trabajo de M. Goebel y L. Gruenwald, L. "A survey of data mining and knowledge discovery software tools" (SIGKDD Explorations, Vol 1.1, 1999, pp. 20–33) disponible en la Red
(`http://www.acm.org/sigs/sigkdd/explorations/issue1-1/survey.pdf`)