Universidad de Huelva

Grado en Ingeniería Informática

Tema 7. Programación Lógica Inductiva

Resumen

Autor: Alberto Fernández Merchán
Asignatura: Aprendizaje Automático

1. Lógica Proposicional

- **Proposición**: Es una afirmación simple que puede tomar el valor de cierto o el de falso. Las proposiciones expresan un conocimiento concreto y no permiten expresar conceptos genéricos como «Todos los gatos son verdes». Para expresar ese concepto habría que enumerar todos los gatos y decir que son verdes ($Gato_1$ *es verde,* $Gato_2$ *es verde,* $Gato_3$ *es verde...*). Para estos casos es mejor utilizar **lógica de primer orden**.

- **Fórmulas Bien Formadas** (FBF): Son expresiones lógicas que se construyen combinando proposiciones simples y conectivas lógicas. Son las expresiones con las que trabaja la lógica proposicional. Una FBF es:

  - Una proposición (p).
  - La negación de una proposición ($\neg p$).
  - La conjunción de dos FBF ($p \wedge q$).
  - La disyunción de dos FBF ($p \vee q$).
  - La implicación entre dos FBF ($p \Rightarrow p$).
  - La doble implicación entre dos FBF ($p \Leftrightarrow q$).

  Puede tomar los valores **cierto** o **falso** en función de las proposiciones simples que la formen. Para representar el significado de la fórmula se utilizan **tablas de verdad** donde se enumeran las combinaciones posibles de las proposiciones simples y el valor de la fórmula para cada combinación.
  Si una FBF tiene N proposiciones, el tamaño de la tabla de verdad será de 2^N filas.

P	Q	P ∨ Q	P ∧ Q	¬ P	P→Q	P↔Q
V	V	V	V	F	V	V
V	F	V	F	F	F	F
F	V	V	F	V	V	F
F	F	F	F	V	V	V

Figura 1: Tabla de Verdad de Conectivos Lógicos

- **Fórmula Válida**: Una FBF es válida si su valor es cierto para **todas** las combinaciones de valores de las proposiciones que la forman. Se conoce también como **tautología**.

- **Fórmula Satisfactible**: Una FBF es satisfactible si su valor es cierto para alguna combinación de valores. Una fórmula es **insatisfactible** si es siempre falsa.

- **Fórmula Decidible**: Una FBF es decidible si se puede demostrar si es válida o no en un número finito de pasos. La lógica proposicional es decidible porque existe un algoritmo (la tabla de verdad) que permite demostrar la validad de cualquier fórmula en un número finito de pasos.

- **Sistema Lógico Consistente**: Un sistema lógico es consistente si no permite demostrar simultáneamente una fórmula y su contraria.

- **Expresiones Lógicas Equivalentes**: Dos expresiones lógicas son equivalentes si tienen la misma tabla de verdad. Algunas expresiones equivalentes son:

  - **Propiedad Distributiva**: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
  - **Elemento Neutro**: $(a \vee F = a)$, $(a \wedge T = a)$
  - **Elemento Absorbente**: $(a \vee T = T)$, $(a \wedge F = F)$
  - **Tercio Excluso**: $(a \vee \neg a) = True.$
  - **Contradicción**: $(a \wedge \neg a) = False.$

- **Forma Normal Conjuntiva**: Aplicando las equivalencias anteriores, es posible convertir cualquier fórmula lógica en FNC. Se caracteriza por:
  - No contiene implicaciones ni dobles implicaciones.
  - Las negaciones solo aparecen en proposiciones simples.
  - Las expresiones están formadas por conjunción de disyunciones.

  Para transformar una fórmula lógica en FNC:

  1. **Sustituir** las implicaciones $(a \Rightarrow b)$ por $(\neg a \vee b)$.
  2. **Sustituir** las dobles implicaciones $(a \leftrightarrow b)$ por $((a \wedge b) \vee (\neg a \wedge \neg b))$
  3. Aplicar la **propiedad distributiva** a las negaciones de FBF: $\neg(a \wedge b) = (\neg a \vee \neg b)$ y $\neg(a \vee b) = (\neg a \wedge \neg b)$
  4. Aplica la **propiedad distributiva** para expresar la fórmula como conjunción de disyunciones.

1.1. Inferencia

El razonamiento en lógica consiste en obtener nuevo conocimiento a partir del que tenemos en nuestra base de conocimiento. Hay varias estrategias para obtener nuevo conocimiento:

- **Deducción** (razonamiento hacia adelante): Obtiene nuevas afirmaciones a partir de los que ya tenemos utilizando mecanismos de inferencia, se añaden a la base de conocimiento y continua buscando nuevas afirmaciones (**búsqueda a ciegas**).
  Se pueden utilizar diferentes mecanismos para generar las nuevas afirmaciones:
  - **Modus Ponens**: Consiste en afirmar el consecuente si existe una implicación y el antecedente es verdadero (si se da la causa, se da el efecto):

    $$B.C. = \{A \Rightarrow B, A\} \qquad \frac{(A \Rightarrow B) \quad A}{B}$$
    $$B.C. = \{A \Rightarrow B, A, B\}$$

  - **Modul Tollens**: Consiste en negar el antecedente si existe una implicación y el consecuente es falso (si no se da el efecto, es imposible que se de la causa).

    $$B.C. = \{A \Rightarrow B, \neg B\} \qquad \frac{(A \Rightarrow B) \quad \neg B}{\neg A}$$
    $$B.C. = \{A \Rightarrow B, \neg B, \neg A\}$$

  - **Resolución**: La regla de resolución es la siguiente:

    $$\frac{\neg p \vee q \vee r \ldots \quad p \vee s \vee t \ldots}{q \vee r \vee \cdots \vee s \vee t \vee \ldots}$$

    Las dos proposiciones superiores deben tener un solo elemento complementario ($\neg p$ y p).
    A partir de esta regla se pueden deducir las dos anteriores. Se puede demostrar que, dada una BC, una expresión Q es cierta:

    1. Consideramos que la B.C. se puede expresar en FNC. Por tanto, podemos considerar la base de conocimientos completa como una única expresión formada por la conjunción de todas sus cláusulas.
    2. Nos preguntamos, entonces, si $B.C. \Rightarrow Q$ es cierto. Podemos transformar la implicación en: $\neg(a \wedge \neg b)$, y debemos preguntarnos si $B.C. \wedge \neg Q$ es falso.

  Esto se conoce como **demostración por refutación** y consiste en añadir la cláusula $\neg Q$ a la base de conocimientos y, utilizando la resolución como mecanismo de deducción, llegar a una contradicción.

La lógica proposicional es **completa**. Si una expresión es válida, entonces existe algún mecanismo que demuestra que lo es. El estudio de la tabla de verdad es un mecanismo completo porque permite demostrar la validez de cualquier expresión. El problema es que su complejidad es exponencial. La verificación en lógica proposicional es un problema **NP-Completo**.

La demostración por refutación es un mecanismo completo porque, si lo que se trata de demostrar es cierto, entonces el mecanismo de resolución encontrará una contradicción. Por otra parte, si lo que se trata de demostrar es falso, el mecanismo no encuentra la contradicción y no para. Nunca podremos saber si el algoritmo no para porque aún no ha encontrado la contradicción o porque no existe. Este método de resolución es, por tanto, **semideducible**.

- **Demostración** (razonamiento hacia atrás): Consiste en explicitar el conocimiento que deseamos obtener y comprobar si es consecuencia lógica de nuestra base de conocimientos (**búsqueda guiada**).

2. Lógica de Primer Orden

La lógica de primer orden considera objetos, sus propiedades y sus relaciones.

- **Término**: Es la forma de expresar los objetos. Pueden ser:
  - **Constantes**: Permiten identificar los objetos del dominio y se denotan por cadenas que comienzan en mayúscula o números.
  - **Variables**: Permiten referenciar cualquier objeto del dominio. Se denotan mediante cadenas que comienzan en minúscula.
- **Funciones**: Permiten obtener una referencia a un objeto en función de los valores de otros. Se denotan mediante un nombre y un conjunto de argumentos (términos). Las constantes se pueden considerar como funciones de 0 argumentos.
- **Átomo**: Equivale a las proposiciones. Pueden ser:
  - **Predicados**: Permiten describir una relación entre objetos o una propiedad del objeto. Se caracterizan por su nombre (cadena que comienza por mayúscula) y el número de argumentos (aridad), que deben ser términos. Un predicado puede ser cierto o falso.
  - **Igualdad o Identidad**: Permiten incorporar la condición de que un término sea igual a otro. Se pueden considerar como un predicado binario. Requieren un tratamiento especial para la inferencia. Suponen una extensión de la lógica de primer orden.

Las **fórmulas bien formadas** en la lógica de primer orden son:

- **Átomo**: p
- **Negación**: $\neg p$
- **Conjunción**: $p \wedge q$
- **Disyunción**: $p \vee q$
- **Implicación**: $p \Rightarrow q$
- **Doble implicación**: $p \equiv q$
- **Cuantificador Universal**: $\forall x : P$
- **Cuantificador Existencial**: $\exists x : P$

Algunas equivalencias asociadas con los cuantificadores son:

- $\forall x : \neg P \equiv \neg(\exists x : P)$
- $\neg(\forall x : P) \equiv \exists x : \neg P$
- $\forall x : P \equiv \neg(\exists x : \neg P)$
- $\neg(\forall x : \neg P) \equiv \exists x : P$
- $\forall x : P \wedge Q \equiv (\forall x : P) \wedge (\forall x : Q)$
- $\exists x : P \vee Q \equiv (\exists x : P) \vee (\exists x : Q)$

Para transformar la expresión lógica de primer orden en FNC:

1. Eliminar las implicaciones
2. Reducir el ámbito de las negaciones para que solo afecten a átomos.
3. Asociar variables distintas a cada cuantificador.
4. Mover los cuantificadores al comienzo, preservando el orden.
5. Eliminar los cuantificadores existenciales.
6. Convertir la fórmula en conjunción de disyunciones.
7. Generar una cláusula a partir de cada disyunción.

2.1. Inferencia

- **Sustitución**: Es una secuencia finita de asociaciones entre variables y términos. $\theta = V_1/t_1, V_2/t_2, \ldots, V_n/t_n$
- **Aplicación de una sustitución θ a una cláusula C**: Consiste en sustituir cada instancia de la variable V_i en la cláusula C por el término t_i asociado en la sustitución. Se denota $C\theta$
- **Unificación**: Dos átomos (P_1 y P_2) unifican si existe una sustitución θ tal que $P_1\theta \equiv P_2\theta$.
- **Unificador más general**: Es la sustitución θ que permite unificar los átomos y cuyos términos son lo más generales posibles (una variable es más general que una constante).

Las reglas de la lógica de primer orden son las siguientes:

- **Regla de Resolución**:
  1. $\neg p \vee \alpha(x)$
  2. $p(y) \vee \beta(y)\theta = UnificadorMasGeneral(p(x)/p(y))$
  3. $\alpha(x) \vee \beta(y)\theta$
- **Regla de Eliminación Universal**:
  1. $\forall x : a(x)$ - $\forall x : Gusta(x, helado)$
  2. $\theta(x/Constante, a(x))$ - $Gusta(Juan, helado)$
- **Regla de Introducción Universal**:
  1. $a(Constante)$
  2. $\exists x : a(x)$

La lógica de primer orden es completa. Dado una base de conocimiento y una FBF (Q) que sea consecuencia lógica de B.C. ($B.C. \models Q$), se puede demostrar Q a partir de B.C ($B.C \vdash Q$) por medio de la demostración por refutación.

Existen lógicas de orden superior cuyos predicados admiten como argumentos otros predicados, funciones o cuantificadores.

Sin embargo, si $BC \not\models Q$, entonces la demostración podría no terminar nunca (semidecidible). Para poder demostrar cualquier fórmula en tiempo lineal, es necesario introducir unas restricciones llamadas **cláusulas de Horn**.

2.2. Cláusulas de Horn

Una cláusula de Horn es una disyunción de literales en la que hay, como mucho, un literal positivo: $\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee ... \vee \neg p_n \vee q$. Si aplicamos las leyes de Morgan quedaría que: $\neg(p_1 \wedge p_2 \wedge p_3 \wedge ... \wedge p_n) \vee q$ y , por definición, esto equivale a : $(p_1 \wedge p_2 \wedge p_3 \wedge ... \wedge p_n) \Rightarrow q$.

- La **cabeza** es el literal positivo (q) y el **cuerpo** al conjunto de literales negativos.
- La lógica basada en cláusulas de Horn permite definir un algoritmo de inferencia que es **completo**, **decidible** y de **orden lineal**.

2.3. SWI-Prolog

SWI-Prolog es un lenguaje de programación lógica basado en cláusulas de Horn. En él, las cláusulas se escriben en primer lugar la cabeza y a continuación el cuerpo.
La implicación se denota por «:-», la conjunción por una coma y las cláusulas deben terminar en un punto.

- **Hecho**: Cláusula sin cuerpo.
- **Regla**: Cláusula con cuerpo.
- **Variables**: Comienza con una letra mayúscula o subrayado.
- **Constantes**: Comienzan con una letra minúscula o entre comillas simples.
- **Predicados**: Comienza con letra minúscula.
- **Términos**: Pueden ser variables, constantes, funciones, predicados y listas.
- Los átomos pueden ser preducados y operadores booleanos.

3. Programación Lógica Inductiva

Consiste en la aplicación de técnicas de aprendizaje inductivo sobre la lógica de primer orden. Construye de forma automática las cláusulas lógicas que describan un cierto predicado en base a un conjunto de hechos positivos y negativos de dicho predicado y a un conjunto de cláusulas auxiliares que describan el conocimiento del dominio.
Un programa lógico es un conjunto de cláusulas de Horn. Se dice que es **consistente** respecto a un conjunto de ejemplos negativos si no existe ningún ejemplo negativo que pueda ser deducido a partir del programa. Se dice que es **completo** respecto a un conjunto de ejemplos positivos si todos los ejemplos pueden ser deducidos a partir del programa. Por tanto, los componentes de un programa lógico serán:

- Conjunto de ejemplos negativos (ϵ^+)
- Conjunto de ejemplos positivos (ϵ^-)
- Un programa lógico consistente (T) a partir del que no se pueda deducir al menos uno de los ejemplos positivos. T representa el conocimiento del dominio.

El objetivo de la programación lógica es encontrar un programa lógico H, tal que, $H \bigcup T$ sea completo y consistente. Esta solución puede ser un programa lógico formado por cláusulas de Horn que describen el predicado al que se refieren los ejemplos del problema.

3.1. Algoritmo FOIL

El **algoritmo FOIL** (*Quinlan*) propone una cláusula lo más general posible (no será consistente con los ejemplos negativos) y va especificándola (añadiendo términos al cuerpo) hasta hallar la consistencia.

```
1  function  FOIL returns listaDeReglas
2  input  Ep // Conjunto de ejemplos positivos
3  input  En // Conjunto de ejemplos negativos
4  input  Domain // Conjunto de predicados del dominio
5  input  Pred // Predicado objetivo
6
7  begin
8          listaDeReglas := Vacia
9          while NoVacio(Ep)
10                 regla := ReglaVacia(Pred)
11                 while Cubre(regla,En)
12                         regla := AnadirLiteral(regla,Domain)
13                 endwhile
14                 Ep := EliminarEjemplosCubiertos(Ep,regla)
15                 listaDeReglas := AnadirRegla(listaDeReglas,regla)
16         endwhile
17         return listaDeReglas
18 end
```

Figura 2: Algoritmo FOIL

Para generar los candidatos a reglas se utiliza el siguiente algoritmo: El algoritmo FOIL se basa en la

FOIL(Ejemplos,Conocimiento-base,Predicado-objetivo(=P))

```
1  R = {}
2  E = Ejemplos
3  mientras Ep in E :
4          Regla = CrearRegla(P(X1,...,Xn),{});
5          mientras cubre(Regla, En):
6                  Literales = GenerarLiterales(BC,Regla)
7                  L = Mejor(Literales)
8                  Regla.add(L)
9          R.add(Regla)
10         E = QuitarCubiertos(E,Regla)
11 devolver R
```

Figura 3: Pseudocódigo para generar los candidatos a reglas

suposición de mundo cerrado, es decir, no es necesario enumerar los ejemplos negativos. Cualquier instancia del predicado objetivo que esté formada por las constantes del conocimiento del dominio y que no se encuentren entre los ejemplos positivos, se considerará negativa.

Para elegir el término candidato, FOIL estudia todas las sutituciones posibles de las variables de la regla. El término que selecciona es el que tenga mayor ganancia de información:

$$ganancia = t \cdot (\log_2(\frac{p_{R'}}{p_{R'} + n_{R'}}) - \log_2(\frac{p_R}{p_R + n_R}))$$

donde:

- R representa la regla inicial.
- R' representa la regla al añadir el término.
- p_R el número de sustituciones positivas que hacen cierta a la regla.
- n_R el número de sustituciones negativas que hacen cierta a la regla.
- t es el número de ejemplos positivos cubiertos por R que permanecen cubiertos por R' (nosotros asumiremos que $t = p_R$)

Para que el programa sea consistente, tendremos que obtener $n_R = 0$.

3.1.1. Mejoras

El algoritmo FOIL se puede mejorar de las siguientes formas:

- **Posibilidad de incorporar constantes**: Los literales estudiados solo utilizan variables. Utilizando constantes podríamos añadir casos base para resolver muchos problemas.
- **Posibilidad de generar reglas recursivas**: Es necesario considerar entre los literales al predicado de la cabeza. En este tipo de reglas es necesario llamar a la función con un tamaño menor cada vez.