# Inteligencia Artificial 2017-2018

## Práctica 1: LISP

Celia San Gregorio Moreno

Álvaro Martínez Morales

Grupo: 2213

Fecha: 03/03/2018

**Índice**

# 1. Similitud Coseno

## Apartado 1.1

**Similitud coseno de forma recursiva**

Para implementar la similitud coseno entre dos vectores de forma recursiva, primero definimos una función auxiliar prod-escalar-rec (x y).

Esta función comprueba primero si x o y son NIL; si lo son devolverá el valor 0, ya que si uno de los vectores es una lista vacía, ello supone multiplicar por 0 cada uno de los elementos del otro vector.

En caso de que ni x ni y sean NIL, calcula la suma de los productos de los primeros elementos de x e y, y de los productos del resto de los elementos. La función termina cuando llega al final de los vectores.

En cuanto a sc-rec(x y), primero comprueba si los módulos de los vectores (o el producto escalar de sí mismos) es 0. Si lo es, devuelve 0 para evitar un caso de división por 0.

Si el módulo de x e y es distinto de 0, calcula la similitud coseno siguiendo la fórmula:

**Similitud coseno usando mapcar**

Para implementar la similitud coseno entre dos vectores usando mapcar, también definimos una función auxiliar prod-escalar-mapcar (x y).

Esta función realiza el producto sobre cada uno de los elementos de x e y utilizando mapcar, y suma cada producto mediante una llamada a reduce. De igual forma que en prod-escalar-rec (x y), si ambos vectores son NIL devuelve 0.

Por último, sc-mapcar(x y) realiza el mismo cálculo que sc-rec(x y),, pero con llamadas a prod-escalar-rec (x y) para realizar las operaciones. De nuevo, si los módulos de x o y son 0, devolverá 0.

## Apartado 1.2

## Apartado 1.3

# 2. Raíces de una función

## Apartado 2.1

## Apartado 2.2

## Apartado 2.3

# 3. Combinación de listas

## Apartado 3.1

## Apartado 3.2

## Apartado 3.3

# 4. Inferencia lógica proposicional

## Apartado 4.1

Apartado 4.1.1

Para comprobar si un literal es positivo, la función positive-literal-p (x) comprueba que x sea un átomo, nunca NIL, T ni cualquier tipo de conector.

Apartado 4.1.2

Para comprobar si un literal es negativo, la función negative-literal-p (x) comprueba que x sea una lista con un conector ~ y un literal positivo x.

Es decir, (~ x).

Apartado 4.1.3

Para comprobar si el argumento es un literal, la función literal-p (x) comprueba que x sea literal positivo o negativo.

Apartado 4.1.4

Para comprobar si una expresión está en formato prefijo (~ FBF) o (FBF <conector> FBF), la función wff-infix-p (wff) sigue una serie de pasos.

Primero comprueba que wff no sea NIL, ya que NIL no es una FBF en formato prefijo. Si wff no es NIL, o bien es un literal o bien es una lista de estos tipos:

* **Lista (~ FBF).** Si tras FBF no hay más elementos, comprueba si FBF está en formato infijo de forma recursiva.
* **Lista (FBF1 <conector-bicondicional> FBF2).** Si después de FBF2 no hay más elementos, comprueba si FBF1 y FBF2 están en formato prefijo de forma recursiva.
* **Lista (<conector-nario>).** Comprueba si se trata de una conjunción o disyunción vacías (‘(v) o ‘(^)).
* **Lista (FBF1 <conector-nario> FBF2 <conector-nario> ... FBFn).** Si la lista es de tipo (FBF1 <conector-nario> FBF2), comprueba si FBF1 y FBF2 están en formato prefijo de forma recursiva.

Si la lista tiene más FBFs, comprueba si tanto FBF1 como (FBF2 ... FBFn) están en formato prefijo de forma recursiva.

Apartado 4.1.5

Para transformar una FBF en formato infijo a una FBF en formato prefijo, la función infix-to-prefix-p (wff) sigue una serie de pasos.

Mientras que wff sea una FBF en formato infijo, comprueba si es un literal. Si lo es, devuelve su valor.

Si wff tiene más elementos (es decir, es una lista de FBFs unidas por conectores), identifica el tipo de conector que las une.

* **Conector unario ~.** Crea una lista con el conector y la FBF convertida a formato infijo de forma recursiva.
* **Conector binario => o <=>.** Crea una lista de tipo (<conector-binario> FBF1 FBF2).
* **Conector n-ario ^ o v.** Si encuentra una lista de tipo ‘(v) o ‘(^), devuelve a conjunción o disyunción vacías.

Si se trata de una lista de tipo (FBF1 <conector-nario> FBF2 <conector-nario> FBFn), crea una nueva lista con el conector n-ario seguido del resto de FBFs. Todas ellas se han convertido a formato prefijo mediante la función:

(mapcar #'infix-to-prefix (remove (second wff) wff)

La llamada a remove elimina los conectores sobrantes de la lista de FBFs, dando como resultado (FBF2 FBF3 … FBFn), lista que se evaluará elemento a elemento.

Apartado 4.1.5

La función clause-p (wff) determina si una FBF es una cláusula mediante estos criterios:

* El argumento wff es una lista precedida por el conector v.
* La lista contiene una disyunción vacía ‘(v), una disyunción con un literal ‘(v lit) o una disyunción con más de un literal. En este último caso se comprueba si cada elemento es un literal de forma recursiva.

Apartado 4.1.6

La función cnf-p (wff) determina si una FBF es una FNC mediante estos criterios:

* El argumento wff es una lista precedida por el conector ^.
* La lista contiene una disyunción vacía ‘(^), una conjunción de tipo ‘(^ (v)) o ‘(^ (v lit1 lit2 ... litn) , o una conjunción con más de una cláusula. En este último caso se comprueba si cada elemento es una cláusula de forma recursiva.

## Apartado 4.2

Apartado 4.2.1

Para convertir la FBF pasada como argumento a una FBF equivalente sin el conector <=>, la función eliminate-biconditional (wff) sigue una serie de pasos.

En primer lugar, si wff es NIL o un literal devuelve su valor. Si se trata de una lista, tendrá la estructura (<conector> FBF1 FBF2).

Si el conector es bicondicional, trasforma FBF1 y FBF2 a FBFs equivalentes sin <=> de forma recursiva, y crea una lista de tipo (^ (=> FBF1 FBF2) (=> FBF2 FBF1)).

Si el conector es unario o enario, devuelve la misma estructura de FBFs (~ FBF) o (FBF1 <conector-nario> FBF2 <conector-nario> ... FBFn), sin el conector <=>. Cada FBF se evalúa con la función mapcar.

Apartado 4.2.2

Para convertir la FBF pasada como argumento a una FBF equivalente sin el conector =>, la función eliminate-conditional (wff) sigue una serie de pasos.

Si wff es NIL o un literal, devuelve su valor. Si se trata de una lista, tendrá la estructura (<conector> FBF1 FBF2).

Si el conector es condicional, trasforma FBF1 y FBF2 a FBFs equivalentes sin => de forma recursiva, y crea una lista de tipo (v (~ FBF1) FBF2).

Si el conector es unario o enario, devuelve la misma estructura de FBFs (~ FBF) o (FBF1 <conector-nario> FBF2 <conector-nario> ... FBFn), sin el conector =>. Cada FBF se evalúa con la función mapcar.

Apartado 4.2.3

La función reduce-scope-of-negation (wff) convierte una FBF sin conectores bicondicionales a una FBF equivalente donde la negación se realiza a nivel de literal.

Primero, la función comprueba si wff es NIL o un literal. Si lo es, devuelve su valor.

En caso de que wff sea una lista, la función considerará los siguientes casos:

* **Lista de tipo (~ (<conector-nario> FBFs).** Niega cada FBF de la lista mediante la función:

(mapcar #'(lambda(x) (list +not+ x)) (rest elems))

Y construye una lista con estos elementos:

1. Conector n-ario ^ cambiado por v (o al revés) llamando a la función auxiliar exchange-and-or (connector).
2. FBFs negadas previamente con el ámbito de negación reducido. El ámbito de la negación de cada una de ellas se reduce mediante la función mapcar.

* **Lista de tipo (~ (~ FBF)).** Devuelve la FBF con el ámbito de negación reducido. Para ello utiliza una llamada recursiva a reduce-scope-of-negation.
* **Lista de tipo (<conector-nario> FBFs).** Crea una lista con el conector n-ario y cada una de las FBFs reducidas en ámbito de negación mediante la función mapcar.

Apartado 4.2.5

Para eliminar los conectores de una FNC, la función eliminate-connectors (cnf) comprueba primero si la FNC pasada como argumento es un literal. En ese caso devolverá su valor.

Sino, cada elemento de la FNC que contenga un conector n-ario se evaluará mediante la función:

(mapcar #'eliminate-connectors (rest cnf))

Y se convertirá a una lista de literales sin conectores.

Apartado 4.2.6

Para convertir una FBF en formato infijo a una FNC en formato prefijo que no tenga conectores, la función wff-infix-to-cnf (wff) realiza llamadas a funciones auxiliares que van transformando la FNC paso a paso como se indica en los apartados anteriores.

## Apartado 4.3

## Apartado 4.4

## Apartado 4.5

## Apartado 4.6