# Programación Lógica y Funcional
## Tarea 5. LISP en IA (ID3)

Angel García Báez
*ZS24019400@estudiantes.uv.mx*

Maestría en Inteligencia Artificial

**IIIA** Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
*Campus Sur, Calle Paseo Lote II, Sección 2a, No 112*
*Nuevo Xalapa, Xalapa, Ver., México 91097*

16 de diciembre de 2024

- Utilizando los siguientes conjuntos de entrenamiento del repositorio UCI (`https://archive.ics.uci.edu/ml/index.php`):

  - Car Evaluation
  - Tic-Tac-Toe
  - Zoo

  llevar a cabo las siguientes actividades:

  1. Implementar en Lisp las mejoras al algoritmo ID3 que adoptaron en su implementación en Prolog [65/100].
  2. . Describa brevemente (máximo una página) las diferencias y similitudes del uso de la programación lógica (prolog) y funcional (lisp) para implementar ID3. [35/100]

# 1. Implementación de las mejoras en el ID3 (Split information y gain ratio)

Para realizar las mejoras al ID3 básico fue necesario retomar lo propuesto por Quinlan (1986) en su articulo Induction of Decision Trees con la finalidad de entender la estructura base del arbol así como de las oportunidades de mejora que puede tener la implementación inicial del mismo.

Después de revisar el articulo junto con las notas de clase de Guerra-Hernández (2024) y retomando la implementación hecha en prolog, se opto por mejorar el algoritmo base mediante el criterio la split information y el gain ratio. Estos criiterios se aplican despues de calcular la entropia y la ganancia de información de tal forma que se pueda penalizar a aquellos atributos que tengan muchas clases y que no esten aportando realemnte a realizar la correcta clasificación de las clases objetivo.

Las formulas necesarias se expresan como siguen:

**Entropía de la información por atributo:**

$$\text{entropia(S)} = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i)$$

Donde:

$$p_i = \frac{|S_i|}{|S|}$$

**Ganancia de información:**

$$\text{gain(S, A)} = \text{entropia(S)} - \sum \frac{|S_v|}{|S|} \cdot \text{entropia}(S_v)$$

**División de la información:**

$$\text{splitInformation(S, A)} = \sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

**Ratio de información:**

$$\text{gainRatio(S, A)} = \frac{\text{gain(S,A)}}{\text{splitInformation(S, A)}}$$

Una vez identificada la mejora y recordados los pasos dados en prolog, se procede a buscar la manera de implementar esto en LISP.

Primero se identificaron las funciones del codigo que se encargan de calcular la entropia y la ganancia de información dentro del codigo.

A continuación se muestra la función de entropia dentro del codigo, la cual no fue cambiada dado que es necesaria para poder hacer el calculo del gain ratio:

```
;;;; Entropia ;;;;
(defun entropy (examples attrib)
  "It computes the entropy of EXAMPLES with respect to an ATTRIB"
  (let ((partition (get-partition attrib examples))
  (number-of-examples (length examples)))
    (apply #'+
     (mapcar #'(lambda(part)
           (let* ((size-part (count-if #'atom
               (cdr part)))
           (proportion
            (if (eq size-part 0) 0
          (/ size-part
              number-of-examples))))
     (* -1.0 proportion (log proportion 2))))
     (cdr partition)))))
```

A continuación se muestra la función de ganancia de información básica que de igual forma, no fue modificada dado que es necesaria:

```
;;;; Ganancia de la información ;;;;;
(defun information-gain (examples attribute)
  "It computes information-gain for an ATTRIBUTE in EXAMPLES"
  (let ((parts (get-partition attribute examples))
  (no-examples (count-if #'atom examples)))
    (- (entropy examples *target*)
       (apply #'+
        (mapcar
          #'(lambda(part)
       (let* ((size-part (count-if #'atom
                 (cdr part)))
         (proportion (if (eq size-part 0) 0
             (/ size-part
                no-examples))))
         (* proportion (entropy (cdr part) *target*))))
         (cdr parts))))))
```

Aquí es donde se añaden las mejoras al codigo, comenzando por añadir el calculo del split-information al codigo principal como sigue:

```
;;; Split Information
(defun split-information (examples attribute)
  "It computes the split information for an ATTRIBUTE in EXAMPLES"
  (let ((partition (get-partition attribute examples))
        (no-examples (length examples)))
    (apply #'+
           (mapcar #'(lambda (part)
                       (let* ((size-part (length (cdr part)))
                              (proportion (if (zerop size-part) 0
                                              (/ size-part no-examples))))
                         (if (zerop proportion) 0
                             (* -1.0 proportion (log proportion 2)))))
                   (cdr partition)))))
```

La siguiente mejora necesaria es añadir el calculo del Gain Ratio como sigue:

```
;;; Gain Ratio
(defun gain-ratio (examples attribute)
  "It computes the gain ratio for an ATTRIBUTE in EXAMPLES"
  (let ((info-gain (information-gain examples attribute))
        (split-info (split-information examples attribute)))
    (if (zerop split-info)
        0
        (/ info-gain split-info))))
```

Una vez que se tienen todas las funciones necesarias, es necesario moidificar la forma en que el algoritmo determina la mejor rama del arbol para que se base en el gain ratio. Dicho codigo cambia la selección basada en la ganancia de información para que en su lugar, considere al atributo que maximiza el gain ratio en cada iteración. El resultado se presenta a continuación como modificación de la función base best-partition:

```lisp
;;; Mejor partición basada en Gain Ratio
(defun best-partition (attributes examples)
  "Computar la mejor partición en base al gain ratio"
  (let* ((gain-ratios
          (loop for attrib in attributes collect
           (let ((gr (gain-ratio examples attrib))
                 (p (get-partition attrib examples)))
             (when *trace*
               (format t "Partición inducida por el atributo ~s:~%~s~%"
                       attrib p)
               (format t "Gain Ratio: ~s~%" gr))
             (list gr p))))
         (best (cadar (sort gain-ratios #'(lambda (x y) (> (car x) (car y)))))))
    (when *trace* (format t "Best partition: ~s~%--------------~%" best))
    best))
```

## Referencias

Guerra-Hernández, D. A. (2024). Programación Lógica y Funcional - Programación Lógica.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1:81–106.