

# Programación para la Inteligencia Artificial

## Proyecto Final

David Galicia

David Hernández

Alberto López

Arturo Márquez Flores

Maestría en Inteligencia Artificial

Universidad Veracruzana

CIIA – Centro de Investigación en Inteligencia Artificial

Sebastián Camacho No 5, Xalapa, Ver., México 91000

[Repositorio en GitHub](#)

8 de Enero del 2019

## Introducción

En este trabajo de fin de curso se implementa una serie de modificaciones y extensiones al paquete *cl-id3* que implementa un algoritmo ID3 en Lisp. En particular, estas modificaciones se enfocan en incluir un método de evaluación y estimación del *error de predicción* del modelo. Este método se llama validación cruzada y es uno de los métodos más populares en la literatura de aprendizaje estadístico [HTF01]. En general, este método es más útil entre más haya disponibilidad de datos para entrenamiento ya que consiste en *apartar* parte de los ejemplos disponibles para poder validar la bondad del algoritmo. Un caso particular de la validación cruzada es cuando esta se repite  $K$  veces. Esto elimina la restricción de necesitar muchos datos para tener una estimación del error de predicción efectiva. La validación cruzada de  $K$  iteraciones consiste en dividir el conjunto de ejemplos de entrenamiento en  $K$  partes aproximadamente iguales y utilizar una para validación y el resto para entrenamiento. El entrenamiento entonces puede hacerse  $K$  veces y, de tal modo, es posible hacer una estimación del error promediando las  $K$  validaciones. En la Figura 1 podemos observar un ejemplo con  $K = 5$ .

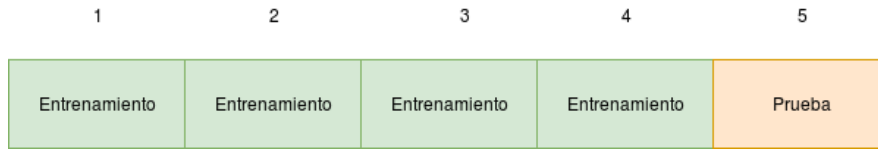


Figura 1: Validación cruzada con 5 iteraciones

En este proyecto se implementa la validación cruzada en el algoritmo de clasificación ID3. Utilizando los árboles generados en esta implementación, comparamos el error de predicción promedio de la validación cruzada con el error de predicción del *mejor* árbol inducido y el error de predicción de una clasificación por votación con todos los árboles inducidos. El proyecto tiene la estructura que fue detallada en el documento del proyecto. La primera sección describe la inclusión de la validación cruzada en el paquete *cl-id3*, la sección dos describe la función de clasificación por votación y la comparación de las estimaciones del error de predicción en los diferentes métodos. Finalmente, la tercera sección establece una manera de automatizar la traducción del mejor árbol obtenido en el proceso a Prolog para poder clasificar nuevos ejemplos en ese lenguaje.

## 1. Validación Cruzada

Para incluir el método de validación cruzada en el paquete *cl-id3* se modificó el archivo **cl-id3-gui.lisp**. En particular, se modificaron las siguientes funciones que se muestran en el Cuadro 1.

```

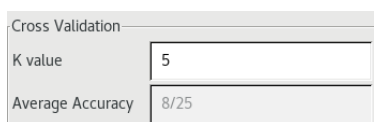
1      ;;; Modificaciones
2      (defvar *classified-int* ())
3      (defvar *c-classified-int* ())
4
5      (defun gui-cross-validation (data interface)
6        (declare (ignore data))
7        (progn
8          ;; Toma K de la interfaz
9          (setf k (text-input-pane-text (k-pane interface)))
10         (setf *classified-int* '() *c-classified-int* '() *k-validation-trees* '() *classify-on* t)
11         ;; Lanza la funcion con K
12         (cross-validation (parse-integer k))
13         ;; Escribe el mejor arbol
14         (traducir *best-tree*)
15         ;; Calcula y define la eficiencia
16         (setf (text-input-pane-text (e-pane interface))
17               (princ-to-string (/ (apply #' + *c-classified-int*)
18                                   (apply #' + *classified-int*))))
19         (setf (text-input-pane-text (e1-pane interface))
20               (princ-to-string (calculate-voting-accuracy *k-validation-trees*)))
21         (setf (text-input-pane-text (e2-pane interface))
22               (princ-to-string (calculate-best-tree-accuracy *best-tree*))))

```

Cuadro 1: Modificaciones al Código en **cl-id3-gui.lisp**

Se utilizó el código que ya estaba disponible en **cl-id3-cross-validation.lisp** para realizar la validación cruzada. Este resultado se integró en la interfaz creando las nuevas variables *classified-int* y *c-classified-int*, que almacenan las clasificaciones correctas y las clasificaciones totales realizadas respectivamente por los  $K$  árboles generados. Estas listas son utilizadas para representar la eficiencia (accuracy) del algoritmo como una estimación del error de predicción que se calcula como el promedio de eficiencias de cada árbol; es decir, la suma de instancias correctamente clasificadas sobre el total de instancias clasificadas.

Para activar la validación cruzada se ingresa el valor de  $K$  en el *text-box* de la interfaz que se muestra en la primera casilla de la Figura 2 y seleccionando en el panel la opción *cross-validation* bajo la opción *id3*. El resultado se muestra en la segunda casilla de la misma Figura 2 donde se muestra el caso de  $K = 5$ .



The image shows a graphical user interface for cross-validation. It has a title bar 'Cross Validation'. Below the title bar, there are two rows. The first row is labeled 'K value' and has a text input field containing the number '5'. The second row is labeled 'Average Accuracy' and has a text input field containing the fraction '8/25'.

Cross Validation	
K value	5
Average Accuracy	8/25

Figura 2: Validación cruzada con 5 iteraciones

Es importante notar aquí que la validación cruzada en **cl-id3-cross-validation.lisp** funciona de manera diferente que en la explicación de la introducción, pues, en este caso, el número  $K$  asigna el número de ejemplos que se utilizan para validación y no el número por el cual se divide el número de ejemplos. En este caso, de los 14 ejemplos disponibles, cada uno de los 5 árboles inducidos toma 5 ejemplos para la validación y los 9 restantes para entrenamiento.

## 2. Clasificación por Votación

Existen métodos para reducir la varianza de las estimaciones de predicción y mejorar su precisión como la agregación de bootstrap [HTF01]. Este método, entre otros, evita un sobreajuste del modelo a los datos. En este caso en particular, existe la posibilidad de crear un proceso de votación entre los árboles generados en la validación cruzada para que la clase elegida para un ejemplo nuevo sea aquella que fue más votada por el conjunto de los árboles. Para lograr esto se realizaron las adiciones a los archivos **cl-id3-classify.lisp**, **cl-id3-cross-validation.lisp** y **cl-id3-gui.lisp** mostradas en los Cuadros 2, 3, 4 y 5.

```

1 (defun classify-new-instance-votacion (ninstance arboles)
2   (car (repetidos
3     (loop for x in arboles
4       collect (classify ninstance x)
5     )
6   ))
7 )

```

Cuadro 2: Modificaciones al Código en `cl-id3-classify.lisp`

```

1
2 (defun calculate-voting-accuracy (trees)
3   (/ (count-voting-positives trees *examples*) (length *examples*)))
4
5 (defun calculate-best-tree-accuracy (best-tree)
6   (/ (count-positives best-tree *examples*) (length *examples*)))
7
8 (defun count-voting-positives (tree data)
9   (apply #' +
10     (mapcar #'(lambda (e)
11       (if (eql (classify-new-instance-votacion e tree)
12         (get-value *target* e))
13         1 0)) data)))
14
15 ;;; Funcion para obtener los elementos mas repetidos de una lista
16 (defun repetidos (lst &optional (resultado '()))
17   (if (null lst)
18     (reverse resultado)
19     (if (member (first lst) (rest lst))
20       (repetidos (rest lst) (adjoin (first lst) resultado))
21       (repetidos (rest lst) resultado))))
22
23 ;;; Selecciona el mejor arbol
24 (defun select-bt (lst)
25   (let ((cont 0) (index 0) (acc 0))
26     (progn
27       (loop for x in lst
28         do (progn
29           (when (> x acc) (and (setf acc x) (setf index cont)))
30           (setf cont (+ cont 1))))
31     index)))

```

Cuadro 3: Modificaciones al Código en `cl-id3-cross-validation.lisp`

```

1
2 (voting-accuracy-pane text-input-pane
3   :text ""
4   :accessor e1-pane
5   :enabled nil)
6 (best-tree-accuracy-pane text-input-pane
7   :text ""
8   :accessor e2-pane
9   :enabled nil)
10
11 (defun gui-cross-validation (data interface)
12   (declare (ignore data))
13   (progn
14     ;; Toma K de la interfaz
15     (setf k (text-input-pane-text (k-pane interface)))
16     (setf *classified-int* '() *c-classified-int* '() *k-validation-trees* '() *classify-on* t)
17     ;; Lanza la funcion con K
18     (cross-validation (parse-integer k))
19     ;; Escribe el mejor arbol en un .txt
20     (traducir *best-tree*)
21     ;; Calcula y define la eficiencia
22     (setf (text-input-pane-text (e-pane interface))
23           (princ-to-string (/ (apply #' + *c-classified-int*)
24                               (apply #' + *classified-int*))))
25     (setf (text-input-pane-text (e1-pane interface))
26           (princ-to-string (calculate-voting-accuracy *k-validation-trees*)))
27     (setf (text-input-pane-text (e2-pane interface))
28           (princ-to-string (calculate-best-tree-accuracy *best-tree*))))
29
30 (defun gui-classify (data interface)
31   (declare (ignore data interface))
32   (define-interface new-classify-int () ()
33     (:panes
34       (ex1-pane text-input-pane
35         :title "Cielo:_"
36         :accessor ex1-pane
37         :text ""
38         :enabled t)
39       (ex2-pane text-input-pane
40         :title "Temperatura:_"
41         :accessor ex2-pane
42         :text ""
43         :enabled t)
44       (ex3-pane text-input-pane
45         :title "Humedad:_"
46         :accessor ex3-pane
47         :text ""
48         :enabled t)
49       (ex4-pane text-input-pane
50         :title "Viento:_"
51         :accessor ex4-pane
52         :text ""
53         :enabled t)
54       (most-voted-class text-input-pane
55         :title "Clase_votada:_"
56         :accessor most-voted-class
57         :text ""
58         :enabled NIL)
59       (bttm-classify push-button
60         :text "Clasificar"
61         :callback 'classify-n-gui)
62       (bttm-quit push-button
63         :text "Cerrar"
64         :callback 'gui-quit))
65     (:default-initargs
66       :title "CL-ID3:_Classify"))
67   (display (make-instance 'new-classify-int)))

```

Cuadro 4: Modificaciones al Código en `cl-id3-gui.lisp`

```

1
2 (defun classifyn-gui (data interface)
3   (declare (ignore data))
4   (progn
5     (setf nsi 0 nno 0)
6     (setf new-lst (list (read-from-string (text-input-pane-text (ex1-pane interface)))
7                         (read-from-string (text-input-pane-text (ex2-pane interface)))
8                         (read-from-string (text-input-pane-text (ex3-pane interface)))
9                         (read-from-string (text-input-pane-text (ex4-pane interface)))))
10    (setf new-l (loop for arbol in *k-validation-trees*
11                    collect (classify-new-instance new-lst arbol)))
12    (setf nsi (length (loop for class in new-l
13                          when (equal (string class) "SI")
14                          collect class)))
15    (setf nno (- (length new-l) nsi))
16    (if (> nsi nno)
17      (setf (text-input-pane-text (most-voted-class interface)) (princ-to-string 'si))
18      (setf (text-input-pane-text (most-voted-class interface)) (princ-to-string 'no))))

```

Cuadro 5: Modificaciones al Código en `cl-id3-gui.lisp`

Los cambios y adiciones señalados consisten, en pocas palabras, en crear una función de clasificación por votación y poder comparar su eficiencia con la del mejor árbol de la validación cruzada. Para lograr esto, como se muestra en el Cuadro 2 la función *classify-new-instance-votacion* recibe un nuevo ejemplo, junto con un conjunto de árboles y regresa la clase que más ocurre en la clasificación de este ejemplo. Similarmente, como se muestra en los Cuadros 4 y 5 la función *classifyn-gui* toma un ejemplo construido desde los inputs en la interfaz del paquete y clasifica el nuevo ejemplo mediante la votación de los *k-validation-trees* que denotan los árboles inducidos durante la validación cruzada. Para hacer esto, uno puede seleccionar en el menú (bajo id3) la opción de *classify* para obtener las casillas de input como se muestra en la Figura 3.

Figura 3: Clasificar otro Ejemplo

Además, la función *select-bt* toma el mejor árboles de este conjunto de árboles inducidos. Utilizando esto y las funciones *calculate-voting-accuracy* y *calculate-best-tree-accuracy* del Cuadro 3 podemos incluir en la interfaz las estimaciones

del error de predicción del método de votación y del mejor árbol inducido en la validación cruzada. Este error se calcula como el número de ejemplos clasificados correctamente sobre el número total de ejemplos clasificados. En este caso todos los 14 ejemplos son utilizados. En la Figura 4 podemos ver un ejemplo de una corrida con  $K = 5$ .

Cross Validation	
K value	5
Average Accuracy	11/25
Voting Accuracy	1
Best-Tree Accuracy	6/7

Figura 4: Clasificación por votación con 5 iteraciones

En este caso, como se esperaba, el desempeño de la votación es mejor que el mejor árbol y aún mejor que el promedio de las estimaciones de todo el conjunto de árboles.

### 3. Traducción a Prolog

Por último, se agregaron funciones al paquete para poder hacer una *traducción automática* del mejor árbol inducido por la validación cruzada a Prolog y poder clasificar nuevos ejemplos. Las adiciones se muestran en los Cuadros 6 y 7. Estas funciones se activan cuando la validación cruzada se realiza y automáticamente toman el mejor árbol para escribir el programa **arbol.pl** que se muestra en 8.

```

1  (defun traducir2 (arbol padre etiqueta)
2
3      (let ((*print-case* :downcase))
4      (progn
5      (if (listp arbol)
6          (loop for i in (cdr (crear-nodos arbol)) do
7              (progn
8                  (if (eq padre 0)
9                      (with-open-file (st "~/quicklisp/local-projects/cl-id3/cl-id3/arbol.pl"
10                          :direction :output
11                          :if-exists :append
12                          :if-does-not-exist :create)
13                          (format str "nodo(~A,~A=~A,raiz).~%"
14                              (setf etiqueta ( etiqueta 1)) (car (crear-nodos arbol)) i))
15                      (with-open-file (st "~/quicklisp/local-projects/cl-id3/cl-id3/arbol.pl"
16                          :direction :output
17                          :if-exists :append
18                          :if-does-not-exist :create)
19                          (format str "nodo(~A,~A=~A,~A).~%"
20                              (setf etiqueta ( etiqueta 1)) (car (crear-nodos arbol)) i padre))
21                      )
22                  (loop for j in (cdr arbol) do
23                      (if (equal i (car j))
24                          (progn
25                              (if (> (length j) 1)
26                                  (traducir2 (cadr j) etiqueta etiqueta)
27                                  (traducir2 (car j) etiqueta etiqueta))
28                              (setf etiqueta (+ etiqueta 1))
29                          )
30                      )
31                  )
32              )
33          )
34      (progn
35      (with-open-file (str "~/quicklisp/local-projects/cl-id3/cl-id3/arbol.pl"
36          :direction :output
37          :if-exists :append
38          :if-does-not-exist :create)
39      (format str "nodo(hoja,[~A/_],~A).~%" arbol etiqueta))
40      )
41      )
42      )
43      )
44      )

```

Cuadro 6: Modificaciones al Código en `cl-id3-cross-validation.lisp`



```

1
2      (defun crear-nodos (lista)
3        (loop for i in lista collect
4          (if (listp i) (car i) i)
5        )
6      )
7
8      (defun traducir (arbol)
9
10       (progn
11         (traducir2 arbol 0 0)
12         (with-open-file (str "~/quicklisp/local-projects/cl-id3/cl-id3/arbol.pl"
13           :direction :output
14           :if-exists :append
15           :if-does-not-exist :create)
16           (format str "~%%Ejemplo:[cielo=soleado,temperatura=alta,humedad=alta,viento=debil].~%
17 ~~~~jugarTennis(Ejemplo)~:-~member(X=Y,Ejemplo),~nodo(N,X=Y,raiz),~njugarTennis(Ejemplo,N),!.~%
18 ~~~~jugarTennis(Ejemplo,N)~:-~member(X=Y,Ejemplo),~nodo(N2,X=Y,N),~njugarTennis(Ejemplo,N2).~%
19 ~~~~jugarTennis(_,N)~:-~nodo(hoja,[X/_],N),~nwrite(X).")
20         )
21       )

```

Cuadro 7: Más modificaciones al Código en `cl-id3-cross-validation.lisp`

```

1
2      nodo(1,cielo=lluvioso,raiz).
3      nodo(2,viento=fuerte,1).
4      nodo(hoja,[no/2],2).
5      nodo(3,viento=debil,1).
6      nodo(hoja,[si/3],3).
7      nodo(4,cielo=nublado,raiz).
8      nodo(hoja,[si/4],4).
9      nodo(5,cielo=soleado,raiz).
10     nodo(6,humedad=normal,5).
11     nodo(hoja,[si/2],6).
12     nodo(7,humedad=alta,5).
13     nodo(hoja,[no/3],7).
14
15     %Ejemplo:[cielo=soleado,temperatura=alta,humedad=alta,viento=debil].
16
17
18     jugarTennis(Ejemplo) :- member(X=Y,Ejemplo), nodo(N,X=Y,raiz), jugarTennis(Ejemplo,N),!.
19
20     jugarTennis(Ejemplo,N) :- member(X=Y,Ejemplo), nodo(N2,X=Y,N), jugarTennis(Ejemplo,N2).
21
22     jugarTennis(_,N) :- nodo(hoja,[X/_],N), write(X).
23

```

Cuadro 8: Mejor árbol en Prolog

Después de haber ejecutado la validación cruzada automáticamente se tendrá el programa de Prolog en el proyecto. Como muestra el Cuadro 9, este programa puede clasificar nuevos ejemplos con el mejor árbol inducido.

```
1  %%% ?- jugarTennis([cielo=nublado, temperatura=alta, humedad=alta, viento=debil]).
2  %%% si
3  %%% true.
```

#### Cuadro 9: Ejecución de **arbol.pl**

Uno de los problemas principales de esta traducción es que no existe un puente directo entre Lisp y Prolog que nos permita hacer queries desde Lisp en Prolog y poder integrarlos a la interfaz para un mejor análisis. En este caso la conexión entre los dos lenguajes es más rígida.

## Referencias

- [HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.