PRÁCTICA Nº 2

EL SISTEMA **AQ**

1º) ***AQ*** es un sistema de obtención de reglas por recubrimiento, cuyo autor es Mitchalski, que originó una saga conocida como la familia *AQn* basada en la metodología *star*, en la que también se basa la familia *CLUSTER/n* para aprendizaje inductivo no supervisado. Se trabajará aquí con una implementación del algoritmo AQ cuya autoría se debe al Sr. Mooney de la Universidad de Texas en Aston[1]. Se carece de documentación precisa, por lo que la información disponible es básicamente el código, un pequeño fichero explicativo, indicaciones explicativas incorporadas al código fuente y la experiencia acumulada. En esta versión se utiliza el símbolo *?* en la representación de los (conjuntos del recubrimiento) para denotar que el valor del atributo no presenta restricción alguna; es decir, puede ser cualquiera. Además la función LEF que utiliza no soporta tolerancias. En realidad, una función LEF es una lista de criterios para determinar un orden en la preferencia de elección de los *complex* disponibles en un momento dado del proceso; en esta implementación la función LEF elige aquel que maximiza el número de ejemplos cubiertos de entre los disponibles en ese momento.

2º) La implementación que se utilizará está realizada en *Common Lisp* y todos los ficheros *lisp* necesarios se encuentran con permiso de lectura en el ordenador **centauro.aulario.uniovi.es** en la dirección //**profesores/alguero/Mooney**, (o también en Hermes //asignaturas/ftp/EnseyApren/Mooney). Cópiense a un directorio propio, desde el que se ejecutarán posteriormente, los siguientes ficheros: ***data-utilities.lsp***, ***aq.lsp, AQquinlan1.lsp AQquinlan2.lsp, dna-standard.lsp***, ***labor-neg.lsp***, ***golf.data***, ***golf.names y golf.test***. Algunos de los demás ficheros también pueden resultar útiles.

3º) Una vez situados en el entorno Lisp disponible en *centauro*, se cargarán (*load*) en ese mismo orden los ficheros ***data-utilities.lsp*** y ***aq.lsp***. En este momento estará listo el sistema y bastará ejecutarlo con los ejemplos que se deseen: Para ello se deberá cargar (*load*) un fichero de ejemplos; por ejemplo, ***AQquinlan1.lsp*** y realizar el entrenamiento mediante la evaluación de la expresión:

(train-aq *raw-examples*)

o, si la salida fuese muy larga, para un resultado más detallado:

(print (train-aq *raw-examples*)),

(que duplica el resultado por pantalla).

Si las clases son dos, denominadas + y - , como en ***AQquinlan1.lsp*** el resultado es una familia de *complex* (conjuntos que recubren los ejemplos expresados como conjunciones de atributos) cubriendo a la clase + . Si. las clases son más, o dos con denominaciones diferentes a las anteriores, como en ***AQquinlan2.lsp*** -en este archivo las clases se denominaron *mas* y *menos*, siendo en lo demás idéntico al archivo *AQquinlan1.lsp*-, el resultado son tantas familias de *complex* como clases; también indica el número de ejemplos cubierto por cada familia. En el último caso, otra posibilidad es evaluar la expresión:

(train-aqr *raw-examples)**,

o para un resultado más detallado: **(print (train-aqr *raw-examples**))**,

que devolverán (de forma repetida en el caso de utilizar la última expresión) las familias de *complex* para cada clase en forma de reglas con la sintaxis siguiente:

($clase_i$ ($clase_i$ (<- $clase_i$+ $atributo_k$ $valor_{kr}$/? ... $atributo_n$ $valor_{ns}$/?) ...) ($clase_j$ (<-$clase_j$ ...))..))

que se leerá:

[1] Copyright (c) 1993 by Raymond Joseph Mooney. This program may be freely copied, used, or modified provided that this copyright notice is included in each copy of this code and parts thereof.

Si $\text{Atributo}_k = \text{valor}_{kr}$ ***y*** ... ***y*** $\text{Atributo}_h = \text{valor}_{hs}$ ***entonces*** clase_i

.....

Si ***... y*** ***.....*** ***y*** ***....*** ***entonces*** clase_j

.....

Se aconseja guardar las reglas resultantes, como testigos para futuras comparaciones, en un archivo cuyo nombre recuerde el algoritmo de aprendizaje utilizado; p. e. aqquinlanreg.txt

4º) Con los ficheros de ejemplos disponibles en el formato adecuado: ***dna-standard.lsp, labor-neg.lsp*** el sistema AQ produce resultados que en el primero de ellos son muy pobres e incomprensibles con la documentación de que se dispone. Compruébese. Como curiosidad, el primero de ellos hace referencia a secuencias de aminoácidos esenciales que intervienen en el ADN. Unas son secuencias reconocidas y clasificadas como de la clase *PROMOTER*[1] y las otras son contraejemplos de ese tipo de secuencias, clase *NEGATIVE*. Con este conjunto de entrenamiento se trata de obtener las reglas que caractericen a las secuencias identificadas como *PROMOTER* frente al resto. Si interesa la explicación del contenido del otro fichero consúltese el archivo *labor-neg.names* en la dirección **/profesores/alguero/examples** de centauro.

5º) La función *translate-c4.5-data-file* permite trasformar los ficheros de ejemplos del sistema C4.5 a la estructura de ficheros de ejemplos admitida por la implementación del AQ de Mooney. Para su aplicación, y a la vista del encabezamiento de su definición,

(defun translate-c4.5-data-file (file-stem &optional output-file) ...)

que aparece en ***data-utilities.lsp***, se evaluará la expresión:

(translate-c4.5-data-file " ...\\ ...*file-stem*" *["...\\...\\fichero-salida "]***)**

Por ejemplo, para *golf.data* y *golf.names* y *golf.test* el **file-stem** es ***golf*** y la expresión a evaluar sería:

(translate-c4.5-data-file " ...\\ ...*golf*" *["...\\...\\golf"]***)**

que devolvería el fichero golf.lisp en la dirección que se le haya indicado y contendría los ejemplos de entrenamiento en le formato de entrada de *aq.lsp.*

Otros ficheros de ejemplos con el formato de entrada para C4.5 se encuentran en la dirección de centauro //**profesores/alguero/examples** y el *file-stem* consiste en el título común a tres ficheros con las extensiones respectivas *.names, .data* y *.test*. Para el *filestem* que se elija deberán de existir obligatoriamente los tres ficheros con las extensiones indicadas. Por experiencia -aunque no del todo confirmada- parece que es necesario crear un *filestem.test* vacío porque en otro caso la función *translate* incorpora los datos de los dos ficheros, *filestem.data* y *filestem.test*, en el fichero de salida (y posteriormente de entrenamiento) *filestem.lisp*. Si existiese inicialmente un fichero *filestem.test* no vacío, con el que se pretendiese evaluar el conocimiento adquirido por el sistema AQ, habría que renombrarlo cambiándole el *filestem* -se recomienda denomiarlo *nuevofielstem.data*- antes de construir el vacío. Posteriormente habría que traducir *nuevofielstem.data* al formato *aq.lsp* mediante la función *translate*, para lo que se haría una copia del antiguo archivo *filestem.names* denominándola *nuevofilestem.names*, y se crearía un archivo vacío denominado *nuevofilestem.test*. Con los tres ficheros creados para el *nuevo.filestem*, se estaría en condiciones de aplicarle a este nuevo *filestem* la función *translate*. El resultado de la traducción de formatos será guardará en el archivo *nuevofilestem.lisp* que contendrá los ejemplos test y que se podrán utilizar en la evaluación del conocimiento aprendido como indica el siguiente apartado.

6º) El fichero *data-utilities.lsp*, contiene una serie de funciones con poca o ninguna documentación entre las que se destaca por su utilidad la función *test-system* que permite comprobar el rendimiento del aprendizaje de cualquiera de los sistemas implementados por Mooney, en particular el AQ, sobre un conjunto test de ejemplos (*filestem.test*). La definición de esta función comienza por:

(defun test-system (system training-result test-examples &optional (print-results t) ....)

[1] The site on an operon that must bind with messenger RNA polymerase before transcription of the operon occurs. (Collins English Dictionary)

por lo que hay que proporcionarle el resultado de un entrenamiento y el conjunto test de ejemplos -al menos.

Si se ejecuta previamente:

(setq aq-result (train-aq *raw-examples*)) [1]

se tendrá en *aq-result* el resultado del entrenamiento para los últimos ejemplos cargados (*load)* y almacenados en ese momento en **raw-examples**, supóngase que fuesen los del fichero *golf.lsp*. Como este último conjunto de ejemplos no dispone de un conjunto test, mediante la función *test-system* comprobaremos los errores de *reescritura*; es decir, los errores cometidos por el conocimiento aprendido sobre los propios ejemplos de entrenamiento, que dicho sea de paso no tiene por qué ser cero. Para ello evalúese la expresión:

(test-system 'aq aq-result *raw-examples*) [2].

Si se dispusiese de un conjunto de ejemplos test bastaría con cargarlo (*load*), tras haber entrenado previamente al sistema mediante la evaluación de la expresión [1], y seguidamente evaluar la expresión anterior [2] tal y como está escrita, ya que en ese caso **raw-examples** contendría en ese momento a los ejemplos test y *aq-result* el conocimiento que se obtuvo ene el entrenamiento.

7º) El fichero *universal-tester.lsp* contiene más funciones para estudiar el resultado de los sistemas implementados por Mooney de la que destacamos *run-cv-tests* que ejecuta un *cross-validation* pero que no se tiene probada y no discutimos aquí, dejando su utilización a elección del usuario.

8º) A continuación se muestran los ejemplos de los ficheros AQquinlan1.lsp y AQquinlan2.lsp; ficheros que solamente se diferencian en el nombre de las clases: + y - para el primero, y *mas* y *menos* para el segundo. Sin embargo el tratamiento que reciben por el programa *aq.lsp* es ligeramente distinto, como ya se habrá podido comprobar.

Ejemplos	***Estatura***	***Pelo***	***Ojos***	***Clase***
$\mathbf{e_1}$	Baja	Rubio	Azules	+
$\mathbf{e_2}$	Alta	Rubio	Castaños	-
$\mathbf{e_3}$	Alta	Pelirrojo	Azules	+
$\mathbf{e_4}$	Baja	Moreno	Azules	-
$\mathbf{e_5}$	Alta	Moreno	Azules	-
$\mathbf{e_6}$	Alta	Rubio	Azules	+
$\mathbf{e_7}$	Alta	Moreno	Castaños	-
$\mathbf{e_8}$	Baja	Rubio	Castaños	-

GIJÓN, ABRIL DE 2001