PRÁCTICA Nº 2

LOS SISTEMAS **AQ y ARCO**

PARTE I: AQ

1º) ***AQ*** es un sistema por recubrimiento de Mitchalski que originó una saga conocida como la familia *AQn* y que se basan en la metodología *star*, en la que también se basan la familia *CLUSTER/n* para aprendizaje no supervisado. Se trabajará aquí con una implementación del algoritmo AQ cuya autoría se debe a Mooney de la Universidad de Texas en Aston. Se carece de documentación precisa por lo que la información disponible es básicamente el código, con pequeñas indicaciones al margen y un pequeño fichero explicativo. La implementación está realizada en *Common Lisp* y todos los ficheros *lisp* que se utilizarán 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 al directorio propio, desde donde 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.

2º) Una vez 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***. Para realizar el entrenamiento se evaluará 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* cubriendo a la clase + . Si. las clases son dos o más 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 es dos o más familias de *complex* -tantas familias como clases- cubriendo los ejemplos por clases; también se indica el número de ejemplos cubierto por cada familia. En este 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:

Si $Atributo_k$= $valor_{kr}$ ***y*** ... ***y*** $Atributo_h$ = $valor_{hs}$ ***entonces*** $clase_i$

.....

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

.....

Guárdense las reglas resultantes como testigo para futuras comparaciones en un archivos con un nombre que recuerde el algoritmo de aprendizaje utilizado; p. e. Aqquinlanreg.

3º) Con los ficheros de ejemplos disponibles en el formato adecuado: ***dna-standard.lsp, labor-neg.lsp*** el sistema AQ produce resultados que en el segundo de ellos son muy pobres e incomprensibles Compruébese. (Los resultados incomprensibles seguramente se obtienen porque el sistema no está preparado para admitir valores desconocidos como aparecen en los ejemplos de ese caso.)

4º) 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 d e entrenamiento en le formato de entrada de *aq.lsp*

Ficheros de ejemplos con el formato de entrada para C4.5 se encuentran en la dirección **/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 existir los tres ficheros con las extensiones indicadas (si no existiese el fichero filestem.test siempre cabría construirlo igual al filestem.data, como se construyó el fichero golf.test, y entonces la validación sería sobre los propios ejemplos de entrenamiento.

5º) 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. La definición de esta función comienza por *(defun test-system (system training-result test-examples &optional (print-results t) ....)* 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 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.

6º) 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 discutimos aquí y dejamos su utilización a elección del usuario.

PARTE II: ARCO

7º) Los algoritmos **ARCO** y su derivado **DARCO,** estructurados como una herramienta de aprendizaje denominada APRENDE e implementada por Mª del Rosario Suárez Fernández en su Proyecto Fin de Carrera, se ejecutan desde el directorio C:/***ArchivosdePrograms/ETSIIII/Aprende*** o en de todos los ordenadores de la Sala 5, o en ***Hermes*** en la dirección ***C:/Asignaturas/ftp/EnseyApren/Recubrimiento/ARCO/Aprende***; bastará activar allí el icono titulado *Aprende*. El entorno es intuitivo como corresponde a un entorno tipo windows. ARCO ordena y depura

redundancias en una familia de reglas con costes asociados y con un conjunto de ejemplos de referencia. DARCO depura redundancias en una familia de atributos en cuanto a la intersección que determina dicha familia; es decir, extrae una subfamilia de atributos de una familia de partida tal que las intersección de los miembros de ambas familias coincide; también necesita de un conjunto de ejemplos como referencia.

8º) En el subdirectorio de ***Hermes***:

C://Asignaturas/ftp/EnseñanzayAprendizaje/Recubrimiento/ARCO/Datos

se tienen los archivos ARCquinlan.ej y ARCquinlan.rg, en los que aparece el conjunto de ejemplos de entrenamiento tomado en las clase de teoría como referencia, codificados conforme a las exigencias de estos algoritmos. También aparecen los ficheros ARCgolf.ej y ARCgolf.reg. Cópiense al directorio temporal con permiso de escritura del ordenador en uso. En un archivo aparecen dichos ejemplos, considerados como ejemplos de entrenamiento que son, y en el otro como reglas -reglas muy específicas y que se tratarán de generalizar en el proceso de aprendizaje. La codificación de los valores de los atributos es numérica y no se permite utilizar el 0 excepto en al codificación de las clases. En el caso citado ha sido

Ejemplos	*Estatura*	*Pelo*	*Ojos*	*Clase*
1= e_1	1= Baja	1= Rubio	1= Azules	1= +
2= e_2	2= Alta	1= Rubio	2= Castaños	0= -
3= e_3	2= Alta	2= Pelirrojo	1= Azules	1= +
4= e_4	1= Baja	3= Moreno	1= Azules	0= -
5= e_5	2= Alta	3= Moreno	1= Azules	0= -
6= e_6	2= Alta	1= Rubio	1= Azules	1= +
7= e_7	2= Alta	3= Moreno	2= Castaños	0= -
8= e_8	1= Baja	1= Rubio	2= Castaños	0= -

9º) Para realizar cualquiera de los procesos relacionados con el aprendizaje se debe de tener cargados, desde el directorio temporal con permiso de escritura, un conjunto de reglas y otro de ejemplos, aunque para el proceso ***Aprendizaje*** bastará con cargar el conjunto de ejemplos. Los posibles **procesos** son:

1. Ordenar y depurar una familia de reglas dada para obtener una subfamilia que optimice el coste de aplicación: ***Ordenar Reglas***.
2. Para cada ejemplo eliminar atributos redundantes de forma que se obtenga una regla con una cierta precisión en su cometido de clasificar: ***Simplificar Atributos***
3. Realizar el proceso ***Aprendizaje*** que supone realizar las dos tareas anteriores correlativamente y en el orden *Simplificar Atributos-Ordenar Reglas* sin detenerse. El resultado es una familia de reglas que clasifica a los ejemplos de entrenamiento con un 0% de errores. Para obtener una familia de reglas que sea no redundante elíjase en *Configuración*, antes de realizar el proceso de aprendizaje, la opción *eliminación inversa para reglas* -y manténgase la eliminación inversa para atributos, que está elegida por defecto.

10º) En la *Configuración* se pueden elegir diferentes opciones que no especificaremos aquí. En principio se trabajará con las opciones de configuración que aparecen por defecto. El proceso de *discretización* y sus correspondientes configuraciones *-paso de discretización y suavizado-* están en fase de prueba y en esta versión del sistema no funcionan correctamente.

GIJÓN, ABRIL DE 2000