Combinando múltiples discretizadores para aprendizaje de reglas evolutivo con enfoque de Pittsburgh

Jaume Bacardit, Josep Maria Garrell y Pere Miralles

Resumen— **Una de las posibles maneras de resolver problemas de clasificación con atributos reales es el uso de algoritmos de discretización, que nos permiten usar las representaciones del conocimiento clásicas para atributos simbólicos. El problema principal de los discretizadores suele ser que es muy difícil encontrar un único algoritmo que permita resolver de forma satisfactoria un gran número de problemas. Para solucionar la elección del discretizador ideal para cada problema, en trabajos previos hemos desarrollado una representación del conocimiento llamada** *Adaptive Discretization Intervals rule representation (ADI)***. Esta representación permite usar múltiples discretizadores al mismo tiempo y permitir al proceso evolutivo la tarea de escoger el discretizador/es ideal/es para cada problema. Hasta el momento se había trabajado mayoritariamente con discretizaciones uniformes de diversos tamaños. En este artículo se experimenta con un espectro amplio de discretizadores, uniformes y no uniformes, combinándolos con diversos criterios. Los resultados de las pruebas realizadas con diversos dominios muestran que se consigue obtener un sistema clasificador robusto y preciso.**

Palabras clave— **aprendizaje evolutivo, representaciones para atributos reales, algoritmos de discretización**

I. Introducción

La aplicación de los Algoritmos Genéticos [1], [2] para solucionar problemas de clasificación se suele identificar como *Genetic Based Machine Learning (GBML)*, y tradicionalmente se han usado dos enfoques diferentes: el de Pittsburgh y el de Michigan, representativos por los sistemas *GABIL* [3] y XCS [4] respectivamente.

Para tratar con problemas con atributos reales una de las posibles alternativas es el uso de un algoritmo de discretización, ya que nos permite reutilizar las representaciones del conocimiento clásicas de este tipo de sistemas. Un buen algoritmo de discretización tiene que buscar el equilibrio entre la pérdida de información inevitable de la discretización y tener un número razonable de puntos de corte. Por otro lado, es muy difícil encontrar un único algoritmo de discretización que dé resultados satisfactorios para un amplio número de problemas.

Para solucionar este tipo de problemas, en trabajos anteriores hemos desarrollado una representación del conocimiento llamada *Adaptive Discretization Intervals Rule Representation (ADI)* [5], [6]. Esta representación, descendiente de la usada en el sistema *GABIL* [3], realiza dos tareas: permite combinar múltiples algoritmos de discretización y, para acotar el espacio de búsqueda, permite fusionar los intervalos propuestos por los discretizadores.

En su versión inicial [5] se combinaban discretizadores uniformes de diversos tamaños, y posteriormente en su segunda revisión (*ADI2*) [6] se realizó una mezcla esquemas de discretización uniformes y no uniformes. Los resultados obtenidos indicaban que se consigue un sistema clasificador muy robusto y que obtiene resultados bastante satisfactorios en un espectro amplio de problemas.

En este artículo se continua la línea de mezclar discretizadores de muy diverso tipo y origen. En concreto se han probado 8 esquemas de discretización que con diversas parametrizaciones han dado lugar a 24 variantes. Primero se realizan pruebas individuales cuyos resultados sirven para proponer 10 posibles combinaciones de esquemas de discretización que se prueban posteriormente.

Este artículo esta estructurado en los siguientes apatados: La sección II presenta el marco de trabajo de nuestro sistema clasificador. A continuación se describe la representación *ADI* y los esquemas de discretización implementados en la sección III. Posteriormente, se detallan las pruebas realizadas en la sección IV, los resultados de las pruebas en la sección V y las conclusiones y lineas de futuro en la sección VI. Finalmente, la sección VII presenta el trabajo más relacionado con el artículo que aquí se presenta.

II. Marco de Trabajo

En esta sección describimos las características principales de nuestro sistema clasificador, que es un descendiente del sistema *GABIL* [3] de De Jong y Spears. Directamente del GABIL hemos

Intelligent Systems Research Group, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Psg. Bonanova 8, 08022-Barcelona, {jbacardit,josepmg,is08388}@salleURL.edu

Si |precisión$_a$ - precisión$_b$ | < umbral **Entonces**
  Si tamaño$_a$ < tamaño$_b$ **Entonces** a es mejor que b
  Si tamaño$_a$ > tamaño$_b$ **Entonces** b es mejor que a
  Si tamaño$_a$ = tamaño$_b$ **Entonces** usamos el caso general
Sino Caso general: usamos el individuo con mayor precisión

Fig. 1. Método de comparación de la selección jerárquica

utilizado el operador de cruce semánticamente correcto y la función de evaluación (porcentaje de acierto elevado al cuadrado).

Estrategia de clasificación: Nuestro conjunto de reglas es ordenado, es decir, que la primera regla cuyo predicado lógico sea cierto para el ejemplo a clasificar será la seleccionada, usando la estructura llamada *Lista de decisión* [7].

Operador de Mutación: Definimos p_{mut} como la probabilidad de mutar un individuo. Cuando un individuo ha sido seleccionado para mutarse, aleatoriamente se escoge uno de sus genes para ser mutado.

Control del tamaño de los individuos: Trabajar con individuos de tamaño variable implica tener que añadir un cierto control sobre éste, para impedir un crecimiento descontrolado de los individuos (el efecto *Bloat* [8]) y para aplicar presión de generalización, que es necesaria para obtener un buen rendimiento en la fase de test. Este control se realiza mediante la combinación de dos operadores:

- *Eliminación de reglas:* Este operador borra las reglas de los individuos que no clasifican ningún ejemplo de entrenamiento. El operador se aplica con un par de restricciones: (a) el proceso se activa después de un cierto número de iteraciones, para evitar una pérdida masiva de diversidad y (b) el operador no se aplica si el número de reglas del individuo es inferior a un cierto umbral.
- *Selección jerárquica [9]:* Usamos un algoritmo de selección basado en torneo con una función de comparación a dos niveles (precisión y tamaño) definida en la figura 1.

III. La representación ADI y los algoritmos de discretización

A. La representación ADI

La estructura semántica de regla es heredada de *GABIL*: Cada regla consiste en una condición y una clase asociada, entendiendo que predecimos la clase asociada para los ejemplos que hagan cierta la condición. La condición es un predicado lógico definido como:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

Donde A_i es el atributo i del problema y V_i^j es el valor j que puede tomar el atributo i.

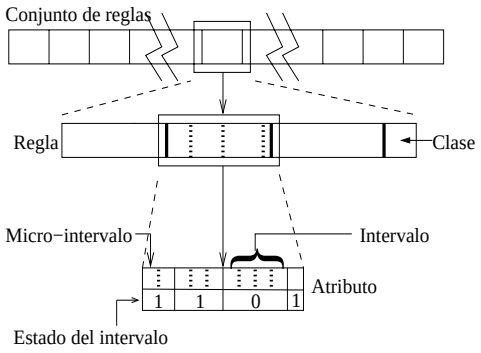

Fig. 2. La representación ADI

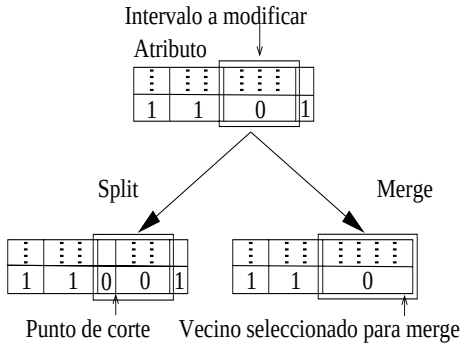

Fig. 3. Los operadores de *split* y *merge*

En *GABIL* este tipo de predicados se pueden codificar en una cadena binaria de la siguiente forma: Supongamos que tenemos un problema con dos atributos, donde cada atributo puede tomar tres valores diferentes {1,2,3}. Una regla de la forma "Si el primer atributo tiene el valor 1 o 2 y el segundo tiene el valor 3, luego predecimos la clase 1" se puede codificar con la cadena binaria 110|001|1. Para atributos reales, cada bit (excepto el de la clase) seria asociado a un intervalo de discretización.

Los intervalos usados en la representación ADI no son estáticos, sino que evolucionan fusionándose (*merge*) o partiéndose (*split*) (teniendo un tamaño mínimo llamado *micro-intervalo*. Es necesario extender la codificación binaria de *GABIL*, y se hace tal como esta representado en la figura 2. La figura 3 muestra las operaciones de *split* y *merge*.

Los detalles de la representación ADI se definen a continuación:

1. Un conjunto de intervalos de discretización estáticos (llamados *micro-intervalos*) se asigna en la fase de inicialización del *GA* para cada atributo de cada regla de cada individuo.
2. Los intervalos de las reglas de forman uniendo *micro-intervalos* contiguos.
3. En la población coexistirán diversos conjun-

tos de intervalos de discretización. La evolución escogerá el discretizador correcto para cada atributo.
4. Por razones de coste computacional, definiremos un límite superior sobre el número de intervalos permitido por atributo, que en la mayoría de los casos sera inferior al número de *micro-intervalos* asignado para cada atributo.
5. Cuando aplicamos una operación de *split* a un intervalo, seleccionamos aleatoriamente un punto en sus *micro-intervalos* para aplicar la operación.
6. Cuando aplicamos una operación de *merge* a dos intervalos, el estado (0 o 1) del intervalo resultante será el del intervalo que contenga más *micro-intervalos*. Si tienen igual número, el estado se escoge aleatoriamente.
7. La discretización asignada en la inicialización para cada atributo se escoge de un conjunto predefinido.
8. El número y tamaño (número de *micro-intervalos* adyacentes) se seleccionan al azar.
9. Los puntos de corte del operador de cruce solo podrán estar en las fronteras entre atributos, para garantizar la integridad semántica de la reglas.

Para integrar los operadores de *split* y *merge* en el proceso evolutivo, hemos ampliado el ciclo de funcionamiento del *GA* con dos fases adicionales que se aplican a la población de hijos después del operador de mutación. La figura 4 contiene el pseudocódigo del operador de *merge*. El operador de *split* funciona de forma equivalente.

B. Algoritmos de discretización integrados en la representación ADI

A continuación se describe de forma breve los 8 tipos de algoritmos de discretización con los que se ha experimentado para este artículo:

- Discretizador de tamaño uniforme [10]. Es el clásico esquema de discretización donde el dominio de cada atributo se divide en n partes de tamaño igual.
- Discretizador de frecuencia uniforme [10]. Suponiendo que el atributo a discretizar tiene m valores distintos, este discretizador divide el dominio de cada variable en n partes, donde cada parte tiene m/n valores continuos del atributo.
- One Rule (1R) [11]. Este discretizador, de forma *greedy*, intenta crear intervalos donde las muestras contenidas tengan asociada la misma clase, con algunas restricciones simples.
- ID3 [12]. Este discretizador utiliza los puntos de corte usados en el algoritmo de inducción de árboles de decisión *ID3*, basándose en la minimización de la entropía de información.
- Fayyad & Irani [13]. Este discretizador esta inspirado en el ID3, pero añadiendo un criterio de parada para no generar demasiados puntos de corte. Este criterio esta basado en el *Minimum Description Length Principle (MDLP)* [14].
- Discretizador de Mántaras [15]. Otro discretizador basado en la minimización de la entropía, pero con una formulación diferente, llamado distancia de Mántaras. También utiliza el criterio *MDLP*.
- Zeta [16]. Este discretizador genera puntos de corte de forma recursiva escogiendo el punto que maximice la presencia de la clase mayoritaria en los intervalos generados. El criterio de parada es llegar al número de puntos de corte especificados por el usuario.
- ChiMerge [17]. Discretizador que parte de tantos intervalos como valores tiene el atributo a discretizar y los fusiona mientras se cumpla un criterio basado en el testo estadístico χ^2. Este discretizador tiene como parámetro el nivel de significación de χ^2.

IV. Diseño de las pruebas

Esta sección describe el diseño de la experimentación realizada para este artículo. En primer lugar se describen los problemas usados para la experimentación y posteriormente se describen las configuraciones de las pruebas realizadas.

A. Problemas de prueba

Para facilitar la comparativa de los resultados de este artículo con los resultados anteriores que se tenían de la representación *ADI*, se han usado los mismos problemas de prueba. En primer lugar tenemos un problema sintético (*tao* [18]) que tiene fronteras entre clases no ortogonales, y es un reto para una representación del tipo que usamos aquí. También hemos usado diversos problemas del repositorio de la Universidad de California en Irvine (UCI) [19]. Los problemas seleccionados son Pima (*pim*), Iris (*irs*), Glass (*gls*) y Winsconsin Breast Cancer (*bre*). Finalmente, hemos usado también tres problemas de nuestro repositorio privado. Los dos primeros tratan sobre diagnóstico de cáncer de mama basado en biopsias (*bps* [20]) y en mamografías (*mmg* [21]). El tercero trata sobre la predicción de notas de alumnos (*lrn* [22]). Las características de los problemas están detalladas en la tabla I. La partición de los ejemplos en conjuntos de entrenamiento y test se ha hecho siguiendo la técnica de validación cruzada estratificada con 10 estratos [23].

```
ParaCada Individuo i de la Población
    ParaCada Regla j del Individuo i
        ParaCada Atributo k de la Regla j del Individuo i
            Si valor aleatorio [0.,1] < p_merge
                Seleccionar un intervalo al azar del atributo
                Aplicar la operación de merge al intervalo
            FinSi
        FinParaCada
    FinParaCada
FinParaCada
```

Fig. 4. Pseudocódigo del operador de *merge*

TABLA I
CARACTERÍSTICAS DE LOS PROBLEMAS DE PRUEBA

ID.	Instancias	Atributos	Clases
tao	1888	2	2
pim	768	8	2
irs	150	4	3
gls	214	9	6
bre	699	9	2
bps	1027	24	2
mmg	216	21	2
lrn	648	6	5

B. Configuración de las pruebas

El objetivo de este artículo es buscar una combinación de esquemas de discretización que usados de forma conjunta en la representación *ADI* tengan un buen rendimiento en un amplio abanico de problemas. Dado que probar todas las posibles combinaciones de esquemas de discretización con sus correspondientes parametrizaciones es un trabajo enorme, realizaremos pruebas individuales con todos los algoritmos de discretización por separado y, posteriormente, generaremos varias combinaciones basándonos en diversos criterios.

Dado que algunos de los discretizadores con los que se experimenta tienen parámetros, pasamos de 8 maneras de discretizar a 24 esquemas de discretización distintos que se detallan en la tabla II.

Los criterios para decidir las combinaciones de discretizadores son los siguientes:

1. Discretizadores con el mejor porcentaje de acierto (en test) para cada uno de los 8 problemas de pruebas
2. 8 primeros discretizadores en ranking de acierto. El ranking se calcula contando cuantas veces cada discretizador ha sido el mejor método en porcentaje de acierto, el segundo, ...
3. 8 primeros discretizadores en porcentaje de acierto
4. Discretizadores sin parámetros
5. 8 discretizadores que generan más puntos de corte
6. 8 discretizadores que generan menos puntos de corte
7. 8 discretizadores que generan soluciones con más reglas
8. 8 discretizadores que generan soluciones con menos reglas
9. 4 primeros discretizadores en ranking (subconjunto del grupo 2, intentando limitar el espacio de búsqueda un poco más)
10. 4 discretizadores que generan soluciones con más reglas (subconjunto del grupo 7, intentando limitar el espacio de búsqueda un poco más)

Finalmente, la configuración del algoritmo genético se detalla en la tabla III.

V. RESULTADOS

En esta sección mostramos los resultados obtenidos para este artículo. Al realizar esta experimentación no solo queremos saber el rendimiento de cada discretizador/combinación de discretizadores sino también como son las reglas generadas por cada configuración. Finalmente, también es interesante saber su coste computacional. Por todo esto, para cada configuración con la que se ha experimentado mostraremos las medias de porcentaje de acierto en test, reglas e intervalos por atributo y coste computacional. En los resultados de discretizadores individuales también informaremos del número medio de puntos de corte de cada discretización.

Los tests se han realizado sobre un Pentium4 a 2.4GHz con sistema operativo Linux y lenguaje de programación C++. Cada test se ha repetido con 15 semillas distintas del generador de números aleatorios, y los resultados se han promediado.

A. Resultados de las pruebas individuales

Por razones de espacio solo se reproducen las medias de resultados de los 8 problemas de prueba (en la tabla IV) y, para cada problema, la configuración que ha tenido mayor porcentaje de acierto en test (en la tabla V). En la tabla IV se ha marcado en negrita el mejor método para cada columna de resultado. Los rankings se han realizado según el porcentaje de acierto.

Podemos extraer algunas conclusiones globales de estos resultados. En primer lugar, si nos fijamos en el porcentaje de acierto podemos ver que

TABLA II

ESQUEMAS DE DISCRETIZACIÓN USADOS

Discretizador	Valores del parámetro	Significado del parámetro
Chimerge	0.01, 0.5, 0.9, 0.99	Nivel de significación
Tamaño uniforme	5,10,15,20,25	Número de intervalos
Frecuencia uniforme	5,10,15,20,25	Número de intervalos
Zeta	5,10,15,20,25	Número de intervalos
One Rule	3,6	Número mínimo de elementos por intervalo
ID3	—	—
Fayyad	—	—
Mántaras	—	—

TABLA III

PARÁMETROS DEL *GA*

Parámetro	Valor
	Parámetros generales
Probabilidad de cruce	0.6
Algoritmo de selección	Torneo de grado 3
Tamaño de la población	300
Probabilidad de mutación de individuo	0.6
Iteraciones	250 para *bps*, *bre*, *irs*, *mmg* y *pim*, 700 para *lrn* y 900 para *gls* y *tao*
	Eliminación de reglas
Iteración de activación	30
Número mínimo de reglas	Número de clases del problema + 3
	Selección jerárquica
Iteración de activación	30
Umbral	0.01 excepto 0.001 para el problema *tao*
	Representación ADI
Probabilidad de *split*	0.05 excepto 0.25 para el problema *tao*
Probabilidad de *merge*	0.05
Numero máximo de intervalos	10 excepto 15 para el problema *tao*

TABLA IV

RESULTADOS DE LAS PRUEBAS INDIVIDUALES

Discretizador	Acierto	Ranking	Reglas	Intervalos por Atributo	Tiempo (s)	Puntos de corte
Chimerge001	78.6 %	15	4.74	2.19	35.3	62.2
Chimerge050	79.3 %	14	5.57	2.86	40.3	94.6
Chimerge090	79.5 %	12	5.57	2.89	40.3	99.0
Chimerge099	79.3 %	11	5.59	2.88	40.3	99.1
FrecUnif05	78.5 %	17	4.83	2.17	30.1	40.5
FrecUnif10	79.5 %	10	5.83	2.84	36.2	91.1
FrecUnif15	79.9 %	2	5.99	3.22	37.2	141.8
FrecUnif20	80.0 %	**1**	6.40	3.50	39.0	192.4
FrecUnif25	78.4 %	7	5.47	3.47	38.5	243.0
Id3	**80.1 %**	3	6.95	3.72	37.8	1563.8
Mántaras	78.3 %	16	3.87	**1.58**	**25.0**	**14.3**
Fayyad	78.9 %	9	4.13	1.64	25.7	16.6
Onerule03	76.9 %	18	4.67	2.29	28.4	548.5
Onerule06	76.9 %	19	4.33	2.12	28.0	242.9
TamañoUnif05	76.6 %	8	4.37	2.14	28.0	40.5
TamañoUnif10	78.6 %	13	4.98	2.60	31.9	91.1
TamañoUnif15	79.8 %	4	5.68	3.18	34.1	141.8
TamañoUnif20	79.9 %	5	6.13	3.46	35.4	192.4
TamañoUnif25	79.6 %	6	6.23	3.52	35.7	243.0
Zeta05	72.7 %	23	**3.86**	1.97	26.6	40.5
Zeta10	76.1 %	20	4.23	2.54	30.3	91.1
Zeta15	76.0 %	24	4.49	2.97	32.8	141.8
Zeta20	76.5 %	22	4.73	3.21	33.6	192.4
Zeta25	77.1 %	21	4.89	3.30	34.0	243.0
Media	78.2 %	–	5.15	2.76	33.5	202.8

TABLA V
MEJOR DISCRETIZADOR PARA CADA PROBLEMA

Problema	Discretizador	Acierto	Reglas	Intervalos por Atributo	Tiempo (s)	Puntos de corte
bps	FrecUnif10	80.8 %	3.54	2.16	53.1	216
bre	TamañoUnif5	96.2 %	2.18	1.48	13.4	36
gls	FrecUnif15	68.6 %	8.59	3.01	38.0	126
irs	TamañoUnif15	96.9 %	3.77	1.48	3.6	56
lrn	Fayyad	69.1 %	6.95	3.13	31.8	10.8
mmg	FrecUnif5	67.8 %	5.26	2.14	12.5	84
pim	FrecUnif5	75.1 %	4.25	1.85	14.8	32
tao	ID3	95.6 %	19.72	9.87	122.1	74

hay dos tipos de discretizadores, el One Rule y el Zeta, cuyos resultados son sensiblemente inferiores al resto de configuraciones. El mejor discretizador en media de acierto es el ID3 y en ranking el de frecuencia uniforme de tamaño 20.

El coste computacional, como es lógico en este tipo de sistemas, está muy ligado al tamaño del individuo en número de reglas y, en menor medida, a intervalos por atributo. Aunque menos clara, parece que hay una correlación entre acierto y número de reglas, indicando que el rendimiento del sistema baja si la discretización hace que se genere un conjunto de reglas demasiado simple.

B. Combinaciones de discretizadores

Con los resultados de las pruebas individuales y los criterios descritos en la sección anterior, las combinaciones de discretizadores que probaremos son las siguientes:

1. TamañoUnif5, TamañoUnif10, TamañoUnif15, FrecUnif5, FrecUnif15, ID3 y Fayyad
2. TamañoUnif5, TamañoUnif15, TamañoUnif20, TamañoUnif25, FrecUnif15, FrecUnif20, FrecUnif25 y ID3
3. TamañoUnif15, TamañoUnif20, TamañoUnif25, FrecUnif10, FrecUnif15, FrecUnif20, ID3 y Chimerge0.9
4. ID3, Fayyad y Mántaras
5. TamañoUnif20, TamañoUnif25, FrecUnif20, FrecUnif25, ID3, OneRule3, OneRule6, Zeta25
6. TamañoUnif5, TamañoUnif10, FrecUnif5, FrecUnif10, Zeta5, Zeta10, ChiMerge001, Mántaras, y Fayyad (son 9 en vez de 8 debido a un empate)
7. TamañoUnif15, TamañoUnif20, TamañoUnif25, FrecUnif10, FrecUnif15, FrecUnif20, ID3 y Chimerge0.99
8. TamañoUnif5, Zeta5, Zeta10, Zeta15, OneRule3, OneRule6, Fayyad y Mántaras
9. TamañoUnif15, FrecUnif15, FrecUnif20 y ID3

Los resultados de estas pruebas con combinaciones de discretizadores están en la tabla VI. Se ha añadido a esta tabla dos filas con los resultados que teníamos de un artículo anterior [6]. La primera fila combina discretizadores de tamaño uniforme y 4,5,6,7,8,10,15,20 y 25 intervalos (llamada *ADI2*). La segunda fila añade a la primera el discretizador de Fayyad e Irani (llamada *ADI2+Fayyad*). El coste computacional no aparece al haberse realizado las pruebas en otra màquina. La tabla VII detalla el mejor grupo para cada problema.

Si agrupásemos la tabla VI con la IV podríamos decir que todos los grupos estarían en la parte alta del ranking de acierto, y la media global es sensiblemente superior a la de los discretizadores individuales. Estos resultados confirman nuestras conclusiones de trabajos anteriores sobre el beneficio de combinar discretizadores [6]. Aunque de forma ligera, se han superado los resultados del artículo anterior.

Respecto a las tablas de los mejores resultados para cada problema de prueba, vemos que los grupos de discretizadores nunca superan al mejor discretizador individual. Este hecho es bastante natural, ya que al tener un solo discretizador el espacio de búsqueda es más pequeño y no siempre el mejor método en entrenamiento tambien es el mejor en test. No obstante, las diferéncias sólo superan el 1 % en 2 de los 8 problemas, lo que muestra la robusteza de los grupos de discretizadores.

Comparando los resultados entre grupos, encontramos las mismas tendencias que teníamos de las pruebas individuales. Los mejores grupos son los mejores del ranking individual (tanto los 4 como los 8 primeros) y las que generaban más reglas (también 8 y 4) y los peores son los dos grupos donde teníamos los discretizadores con menos reglas y menos puntos de corte. La mejor media (la del grupo 9 de los 4 mejores discretizadores en ranking de acierto) es superior al mejor discretizador individual (el ID3) aunque la diferencia es muy pequeña.

VI. CONCLUSIONES Y TRABAJO FUTURO

En este artículo hemos continuado el trabajo anterior sobre una representación del conocimiento llamada *Adaptive Discretization Intervals*

TABLA VI

Resultados de las pruebas combinatorias

Discretizador	Acierto	Reglas	Intervalos por Atributo	Tiempo (s)
Grupo 1	79.4 %	5.91	2.69	34.9
Grupo 2	80.0 %	6.57	3.31	38.0
Grupo 3	79.7 %	6.60	3.32	44.6
Grupo 4	79.4 %	5.38	2.34	30.3
Grupo 5	79.7 %	6.18	3.21	39.5
Grupo 6	79.1 %	5.34	2.32	39.9
Grupo 7	79.9 %	6.70	3.32	44.7
Grupo 8	78.6 %	**5.07**	**2.17**	**34.4**
Grupo 9	**80.2 %**	6.74	3.47	38.8
Grupo 10	79.8 %	6.80	3.56	38.2
Media	79.6 %	6.13	2.97	38.3
ADI2	80.0 %	6.30	3.10	—
ADI2+Fayyad	80.0 %	6.00	2.90	—

TABLA VII

Mejor grupo de discretizadores para cada problema

Problema	Grupo	Acierto	Reglas	Intervalos por Atributo	Tiempo (s)
bps	Grupo 9	80.4 %	3.97	2.47	68.3
bre	Grupo 6	96.0 %	2.37	1.64	14.6
gls	Grupo 7	67.2 %	8.22	3.10	35.6
irs	Grupo 9	95.4 %	3.79	1.70	3.7
lrn	Grupo 4	68.9 %	6.92	3.39	33.2
mmg	Grupo 9	66.8 %	5.09	2.62	12.2
pim	Grupo 2	74.8 %	4.12	2.55	15.2
tao	Grupo 3	95.1 %	17.33	8.28	120.5

rule representation (ADI), usada en un sistema de aprendizaje evolutivo basado en el enfoque de Pittsburgh. Esta representación esta pensada para tratar problemas con atributos reales basándose en la combinación de múltiples discretizadores.

En este artículo nos hemos centrado en realizar pruebas exhaustivas sobre varios tipos de discretizadores, para encontrar una combinación que tenga un buen rendimiento sobre un espectro amplio de problemas. Se ha probado 8 algoritmos de discretización que combinados con sus posibles parametrizaciones han dado lugar a 24 posibles esquemas de discretización.

Las pruebas con cada uno de los discretizadores por separado ya han identificado varias familias de discretizadores cuyo rendimiento era sensiblemente inferior al resto y, en general, se identifica la tendencia que los discretizadores que funcionan mejor son los que generan soluciones más grandes y con mas intervalos por atributo. Creemos que este hecho se debe a que estos esquemas son los que no han perdido demasiada información al discretizar.

A partir de las pruebas individuales se ha propuesto 10 grupos diferentes basándose en diferentes criterios. Los resultados de las pruebas combinatorias muestran que todas las combinaciones de discretizadores tienen un rendimiento bastante aceptable, corroborando la experimentación iniciada en nuestros trabajos anteriores. En concreto, destacan los grupos de discretizadores que contienen los 8 o 4 primeros discretizadores en ranking de acierto, esta última superando al mejor discretizador individual, aunque por un escaso margen.

Como trabajo futuro habría que investigar las causas de este escaso margen entre las medias de acierto del mejor discretizador individual y la mejor combinación, como tambien las diferencias entre el mejor discretizador/grupo de discretizadores para cada problema. Una vez determinadas las causas habria que buscar si es posible aumentar su rendimiento. Por otro lado, quedan todavía muchos otros tipos de algoritmos de discretizacion a combinar con los actuales.

VII. Trabajo relacionado

En el campo de los *GBML* existen muchos enfoques sobre como tratar con atributos reales. De una forma muy simplista se pueden clasificar en dos grandes grupos: los sistemas que usan discretización y los que manejan directamente los valores reales.

En la bibliografía existen muchos algoritmos de discretización, en la sección III se detallaran todos los algoritmos con los que se ha experimentado para este artículo. Para citar un algoritmo aquí, podemos nombrar el de Fayyad e Irani [13], que es uno de los algoritmos de discretización más utilizados en la actualidad.

Como aplicación específica de las discretizaciones en el campo de los *GBML* encontramos la Codificación Natural de Aguilar y Riquelme [24], que es una propuesta cercana a nuestra representación ADI, pero usando un tipo de reglas semánticamente diferentes y unos operadores de cruce y mutación especializados. Usan su propio algoritmo de discretización, llamado *USD* [25].

Hay diversas alternativas de representaciones del conocimiento que trabajan directamente con valores reales. Encontramos reglas con intervalos reales (el sistema XCSR [26] de Wilson), o el sistema GALE [18] de Llorà y Garrell, donde coevolucionan varias representaciones del conocimiento como árboles de decisión o conjuntos de instancias usadas como núcleo de un clasificador de tipo vecino más cercano.

Agradecimientos

Este trabajo ha estado parcialmente financiado por el Programa Nacional de I+D+I (proyectos TIC2002-04036-C05-03 y TIC2002-04160-C02-02) del Ministerio de Ciencia y Tecnología (cofinanciado por el Fondo Europeo de Desarrollo Regional) y con la ayuda para grupos de investigación consolidados 2002SGR 00155 concedida por la Generalitat de Catalunya. El primer autor reconoce el soporte del *Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya* bajo la beca 2001FI 00514. Finalmente, los autores quieren agradecer el soporte constante de Ingeniería y Arquitectura La Salle al Grupo de Investigación en Sistemas Inteligentes.

Referencias

[1] John H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[2] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[3] Kenneth A. DeJong and William M. Spears, "Learning concept classification rules using genetic algorithms," *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 651–656, 1991.

[4] Steward W. Wilson, "Classifier fitness based on accuracy.," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[5] Jaume Bacardit and Josep M. Garrell, "Evolution of multi-adaptive discretization intervals for A rule-based genetic learning system," in *Proceedings of the Eighth Iberoamerican Conference on Artificial Intelligence*. 2002, pp. 350–360, LNAI vol. 2527, Springer.

[6] Jaume Bacardit and Josep M. Garrell, "Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system," in *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003*. 2003, pp. 1818–1831, LNCS 2724, Springer.

[7] Ronald L. Rivest, "Learning decision lists," *Machine Learning*, vol. 2, no. 3, pp. 229–246, 1987.

[8] Terence Soule and James A. Foster, "Effects of code growth and parsimony pressure on populations in genetic programming," *Evolutionary Computation*, vol. 6, no. 4, pp. 293–309, Winter 1998.

[9] Jaume Bacardit and Josep M. Garrell, "Métodos de generalización para sistemas clasificadores de Pittsburgh," in *Proceedings of the "Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)"*, 2002, pp. 486–493.

[10] H. Liu, F. Hussain, C. Lim Tam, and M. Dash, "Discretization: An enabling technique," *Data Mining and Knowledge Discovery*, vol. 6, no. 4, pp. 393–423, 2002.

[11] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, vol. 11, pp. 63–91, 1993.

[12] J. Ross Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81 – 106, 1986.

[13] Usama M. Fayyad and Keki B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *IJCAI*, 1993, pp. 1022–1029.

[14] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. vol. 14, pp. 465–471, 1978.

[15] R. López De Mántaras, "A distance-based attribute selection measure for decision tree induction," *Machine Learning*, vol. 6, no. 1, pp. 81–92, 1991.

[16] K. M. Ho and P. D. Scott, "Zeta: A global method for discretization of continuous variables," in *Knowledge Discovery and Data Mining*, 1997, pp. 191–194.

[17] R. Kerber, "Chimerge: Discretization of numeric attributes," in *Proc. of AAAI-92*, San Jose, CA, 1992, pp. 123–128.

[18] Xavier Llorà and Josep M. Garrell, "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *Proceedings of the Third Genetic and Evolutionary Computation Conference*. 2001, pp. 461–468, Morgan Kaufmann.

[19] C. Blake, E. Keogh, and C. Merz, "Uci repository of machine learning databases," 1998, (www.ics.uci.edu/mlearn/MLRepository.html).

[20] E. Martínez Marroquín, C. Vos, and et al., "Morphological analysis of mammary biopsy images," in *Proceedings of the IEEE International Conference on Image Processing*, 1996, pp. 943–947.

[21] J. Martí, X. Cufí, J. Regincós, and et al., "Shape-based feature selection for microcalcification evaluation," in *Imaging Conference on Image Processing, 3338:1215-1224*, 1998.

[22] Elisabet Golobardes, Xavier Llorà, Josep Maria Garrell, David Vernet, and Jaume Bacardit, "Genetic classifier system as a heuristic weighting method for a case-based classifier system," *Butlletí de l'Associació Catalana d'Intel.ligència Artificial*, vol. 22, pp. 132–141, 2000.

[23] Ian H. Witten and Eibe Frank, *Data Mining: practical machine learning tools and techniques with java implementations*, Morgan Kaufmann, 2000.

[24] Jesus S. Aguilar-Ruiz, José C. Riquelme, and Carmelo Del Valle, "Improving the evolutionary coding for machine learning tasks," in *Proceedings of the European Conference on Artificial Intelligence, ECAI'02*, Lyon, France, 2002, pp. pp. 173–177, IOS Press.

[25] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme, "Discretización supervisada no paramétrica orientada a la obtención de reglas de decisión," in *Proceedings of the CAEPIA2001*, 2001, pp. 53–62.

[26] Stewart W. Wilson, "Get real! XCS with continuous-valued inputs," in *Festschrift in Honor of John H. Holland*, L. Booker, Stephanie Forrest, M. Mitchell, and Rick L. Riolo, Eds. 1999, pp. 111–121, Center for the Study of Complex Systems.