Complejidad del Aprendizaje y Muestreo de Ejemplos en Sistemas Clasificadores

Ester Bernadó Mansilla

Resumen— **Este trabajo se centra en la aplicación de los Sistemas Clasificadores basados en Algoritmos Genéticos a problemas de aprendizaje supervisado o aprendizaje a partir de ejemplos. Concretamente, se estudia la dependencia entre la complejidad del aprendizaje del sistema clasificador y la representatividad de los ejemplos en el conjunto de entrenamiento. A partir de ahí, se propone un método de muestreo del conjunto de ejemplos de entrenamiento basado en los resultados parciales del sistema clasificador, de forma similar al método de *boosting*. El objetivo es intensificar el aprendizaje en las regiones de clasificación más difíciles para aumentar la velocidad de convergencia y a la vez, el porcentaje de aciertos.**

Palabras clave— **Sistemas Clasificadores, Algoritmos Genéticos, Aprendizaje, Complejidad.**

I. Introducción

El aprendizaje supervisado o aprendizaje a partir de ejemplos consiste en entrenar un sistema para que aprenda la función inherente a una colección de ejemplos. Dentro de este contexto, la *clasificación* consiste en aprender la función que relaciona los atributos de los ejemplos con la clasificación asociada.

Actualmente, los sistemas clasificadores basados en Algoritmos Genéticos (AG) están experimentando un notable auge entre la comunidad científica de aprendizaje artificial. Gran parte de este éxito se debe a trabajos de investigación recientes que demuestran el buen rendimiento de estos sistemas en problemas de clasificación, y su competitividad con respecto a otros algoritmos de aprendizaje tradicionales. En este marco, el sistema clasificador XCS [1], [2] es uno de los máximos exponentes. XCS es un sistema basado en el enfoque de Michigan, enfoque también denominado como *Learning Classifier Systems* (LCS), propuesto inicialmente por Holland [3], [4].

XCS se caracteriza por evolucionar una población de individuos donde cada individuo representa una regla de clasificación. Mediante la interacción con el entorno, bajo un esquema de aprendizaje por refuerzo [5], el sistema evalúa y actualiza el sistema de reglas. El algoritmo genético interviene como sistema de búsqueda clave para mejorar el conocimiento adquirido en forma de reglas y converger hacia un conjunto mínimo de reglas que representen el conocimiento de forma precisa y general.

Aunque el aprendizaje por refuerzo no impide que XCS se pueda usar para problemas de clasificación, se ha demostrado que el esquema de aprendizaje supervisado da lugar a resultados más eficientes [6], [7]. Así surge el sistema clasificador UCS [6], [7], propuesto como una variante del sistema XCS diseñada específicamente para problemas de aprendizaje supervisado.

El proceso de aprendizaje de estos sistemas es incremental. El sistema muestrea uniformemente el conjunto de ejemplos de entrenamiento. Por cada ejemplo, el sistema clasificador actualiza incrementalmente el conjunto de reglas de acuerdo con su clasificación. Estudios recientes han demostrado que la complejidad del aprendizaje depende de la distribución de los ejemplos representativos en el conjunto de entrenamiento [6], [7]. Es decir, el sistema tiende a aprender más rápido los ejemplos de las clases mayoritarias y necesita muchas iteraciones para poder clasificar correctamente las clases que tienen menos ejemplos representativos. Consecuentemente, es posible que para conseguir un aprendizaje completo el sistema tenga que realizar iteraciones excesivas. O bien, si el sistema se detiene después de un tiempo razonable pero no suficiente, puede haber aprendido tan sólo las clases más representativas. A primera vista, podría parecer que las clases más representativas son las más importantes y por tanto, hasta cierto punto este comportamiento sería deseable. Sin embargo, muchos problemas reales demuestran lo contrario. Por ejemplo, un problema de clasificación médico suele contar con más ejemplos representativos de pacientes que no padecen la enfermedad que los que sí la padecen. Éste es un caso en que interesa que el sistema también aprenda las clases minoritarias.

Este trabajo propone muestrear convenientemente el conjunto de ejemplos de entrenamiento para reforzar el aprendizaje en aquellas zonas del espacio de búsqueda más difíciles. El objetivo es aumentar la velocidad del aprendizaje e incluso mejorar el porcentaje de clasificación del sistema.

Departamento de Infórmatica. Enginyeria i Arquitectura La Salle. Universitat Ramon Llull. Passeig Bonanova, 8. 08022 Barcelona. E-mail: esterb@salleURL.edu

De hecho, la idea de muestrear adaptativamente el conjunto de ejemplos de entrenamiento se inspira en trabajos previos realizados en el campo de multiclasificadores como el método *boosting* [8], [9]. Boosting realiza un muestreo adaptativo del conjunto de ejemplos de entrenamiento. En cada iteración, se entrena un sistema clasificador y según su clasificación altera el muestreo del conjunto de entrenamiento para reforzar la clasificación de los ejemplos peor clasificados en futuras iteraciones. La diferencia de nuestro método con el de *boosting* es que aquí no entrenamos múltiples sistemas clasificadores, sino tan sólo uno, pero alterando la frecuencia de muestreo de los ejemplos de acuerdo con las clasificaciones parciales del sistema.

Este artículo se estructura de la siguiente forma. A continuación, se describe brevemente qué es el método *boosting* y sus similitudes con el presente trabajo. Posteriormente, se describe el sistema UCS, el cual usamos como base de nuestro estudio, con el objetivo de exportar los resultados a otros sistemas clasificadores similares como XCS. La sección IV analiza cómo el aprendizaje del sistema depende de la representatividad de los ejemplos. La sección V propone un método de muestreo adaptativo y presenta resultados comparativos con el muestreo uniforme. Finalmente, se presentan las conclusiones y trabajo futuro.

II. Boosting

El método *boosting* [8], [9] está diseñado para mejorar la precisión de un sistema de aprendizaje. El algoritmo entrena repetidamente un clasificador, cambiando la distribución de los ejemplos de entrenamiento de acuerdo con las clasificaciones de los sistemas que ha entrenado previamente. La idea clave es que a pesar de que el clasificador base sea muy sencillo, el multiclasificador combinado que resulta de las múltiples repeticiones es muy potente y llega a niveles de clasificación muy superiores a los clasificadores individuales.

El funcionamiento concreto del método *boosting* es el siguiente. Se mantiene un conjunto de pesos sobre el conjunto de entrenamiento. Cada ejemplo de entrenamiento tiene asociado un peso. Inicialmente, todos los pesos tienen el mismo valor. El método *boosting* entrena un clasificador repetidamente durante T iteraciones. En cada iteración del algoritmo, se entrena un clasificador y se evalúan los ejemplos clasificados correctamente e incorrectamente. Aquellos ejemplos que se clasifican incorrectamente ven incrementado su peso. De esta forma, se fuerza que el clasificador aprenda o refuerce su aprendizaje en los ejemplos más difíciles del conjunto de entrenamiento.

El clasificador puede usar directamente los pesos de los ejemplos de entrenamiento para realizar su clasificación. O alternativamente puede muestrear un subconjunto de ejemplos (de acuerdo con la distribución de pesos) y entrenar el clasificador con este subconjunto de ejemplos sin pesos. El resultado del clasificador combinado se obtiene a partir de la votación ponderada de los clasificadores individuales.

Este trabajo parte de la idea del método *boosting* para mejorar el aprendizaje de los sistemas clasificadores LCS. Los LCS evolucionan un conjunto de reglas incrementalmente, a partir de un muestreo uniforme de los ejemplos del conjunto de entrenamiento. Se ha demostrado que la complejidad del aprendizaje de cada regla depende del número de ejemplos representativos de la misma [6], [7]. Es decir, el sistema aprende de forma más rápida las regiones de clasificación que cuentan con más ejemplos representativos. Debido al muestreo uniforme, el sistema clasificador puede estar recibiendo repetitivamente ejemplos que ya están clasificados correctamente, y sin embargo apenas recibir suficientes muestras de las regiones de clasificación más difíciles y que por tanto están peor clasificadas. El método *boosting* proporciona un marco para cambiar esta tendencia. La idea es muestrear el conjunto de ejemplos de entrenamiento con una distribución que dependa del porcentaje de clasificación del sistema. Como se ha comentado, la diferencia de nuestro enfoque con el de *boosting* es que nosotros entrenamos un único sistema clasificador. El muestreo se realiza de forma adaptativa, en función de los resultados parciales del sistema. *Boosting*, en cambio, entrena múltiples clasificadores, con una distribución de los ejemplos de entrenamiento que depende del resultado final de los clasificadores previos.

III. Sistema Clasificador UCS

En esta sección se describe brevemente el sistema UCS. UCS se basa en el sistema XCS, pero a diferencia de éste, UCS está diseñado específicamente para problemas de aprendizaje supervisado. Para más detalles sobre UCS y XCS, el lector puede dirigirse a [6], [7] y [1], [2], [10] respectivamente.

A. Representación

UCS codifica el conocimiento en una población [P] de individuos. Cada individuo se denomina clasificador y consta de una regla de clasificación y un conjunto de parámetros asociados que estiman la calidad de la regla. La regla es del tipo *condición* $\rightarrow$ *clase*. La condición repre-

```
t ← 0
inicializar [P]
MIENTRAS no se cumpla condición fin
    t ← t+1
    obtener x del conjunto de entrenamiento
    calcular [M] con condición que satisface x
    SI  modo=aprendizaje
        obtener la clase C correspondiente a x
        calcular [C] y [!C]
        actualizar parámetros de [C] y [!C]
        SI activarAG()
            aplicar el Algoritmo Genético
        FSI
    SINO    /*modo = test*/
        predecir la clase C con votación de [M]
    FSI
FMIENTRAS
```

Fig. 1. Algoritmo del sistema clasificador UCS.

senta el conjunto de estados de entrada que se clasifican de acuerdo con la clase especificada. Si los atributos de los ejemplos de entrada son binarios, la condición se representa mediante el alfabeto ternario: $\{0, 1, \#\}$, donde $\#$ representa el símbolo comodín. Dada una entrada $\mathbf{x} = (x_l, .., x_2, x_1)$ y una regla $r = (c_l, ..., c_2, c_1) \rightarrow a$, la condición se activa si $\forall_i \; c_i = \# \vee x_i = c_i$.

Los principales parámetros asociados con la calidad de la regla son:

- *acc*: la precisión de la regla.
- *F*: el *fitness*.

Existen otros parámetros necesarios para el aprendizaje y explotación del sistema. Éstos son:

- *exp*: la experiencia de la regla o número de veces que se actualiza.
- *ns*: *niche size*, el tamaño promedio de los conjuntos [C] donde participa, tal como se explica más adelante.
- *ts*: *time stamp*, la iteración donde se aplicó el AG por última vez.
- *num*: numerosidad, o número de copias de esta regla que se codifican en la población.

A continuación, se detallan cómo se calculan y usan estos parámetros.

B. Componente de rendimiento

En la figura 1 se esquematiza el algoritmo del sistema UCS. En cada iteración del algoritmo, se suministra al sistema un ejemplo de entrada seleccionado del conjunto de ejemplos de entrenamiento $\mathbf{x} = (x_l, ..., x_2, x_1)$. El sistema busca las reglas cuyas condiciones satisfacen los atributos del ejemplo. A este subconjunto se le denomina *match set* [M]. A continuación el sistema opera de forma diferente según si está en *modo aprendizaje* o en *modo test*.

En modo aprendizaje, el ejemplo de entrada proviene con su clasificación correspondiente $\mathcal{C}$. Se comparan las clases propuestas por los clasificadores y la clase $\mathcal{C}$ correspondiente al ejemplo. Los clasificadores pertenecientes a [M] se dividen en dos subconjuntos: [C], formado por los clasificadores que proponen la clasificación correcta, y [!C], formado por el resto de clasificadores. En caso de que [C] esté vacío, se activa el operador de *covering* el cual se encarga de crear nuevos clasificadores con una condición generalizada acorde con el ejemplo actual y la clase $\mathcal{C}$ correspondiente al ejemplo. A continuación, se actualizan los parámetros de los clasificadores y eventualmente, se aplica el AG.

Si el sistema opera en modo test, se suministra un ejemplo $\mathbf{x} = (x_l, ..., x_2, x_1)$ y el sistema tiene que predecir su clasificación asociada. Para ello, se forma el conjunto [M] y la clasificación se determina a partir de una votación ponderada por fitness de los clasificadores presentes en [M].

C. Actualización de parámetros

En modo aprendizaje, se actualizan los parámetros de los clasificadores de [M], según si han propuesto la clase correcta o no. En primer lugar, se actualiza la precisión de cada regla:

$$acc = \frac{\text{número de clasificaciones correctas}}{\text{número de ejemplos cubiertos}}$$

Según esta fórmula, la precisión de una regla es el acumulado de los ejemplos que ha clasificado correctamente dividido por el número de ejemplos que ha cubierto. El fitness se calcula según:

$$F = (acc)^{\nu}$$

donde ν es una constante. La experiencia de cada clasificador *exp* representa el número de veces que un clasificador actualiza sus parámetros. Se incrementa cada vez que el clasificador participa en [M]. El parámetro ns se actualiza para todos los clasificadores de [C]. Almacena el número promedio de clasificadores que pertenecen a [C].

D. Algoritmo Genético

El AG tiene una tarea multimodal, ya que debe mantener un conjunto diverso de reglas que globalmente representen el conocimiento de la base de datos. Por tanto, el AG cuenta con algoritmos que favorecen la especiación o *niching* [11].

El AG se aplica localmente sobre [C], en lugar de hacerlo sobre toda la población [P]. Esto da lugar a un cruce restringido a clasificadores que tienen condiciones muy similares, favoreciendo así la especiación. El AG opera de la siguiente forma. Se seleccionan dos padres de [C] con probabilidad proporcional al fitness y se aplican

los operadores de cruce y mutación. Los descendientes se reintroducen en la población y compiten con los padres. Si la población está llena, se aplica un operador de borrado. La probabilidad de que un clasificador sea eliminado depende proporcionalmente del tamaño promedio de los conjuntos [C] donde participa. Esta medida se guarda en el parámetro ns (*niche size*) de cada clasificador. También depende inversamente del fitness del clasificador. Mediante el operador de borrado, se intentan equilibrar los recursos entre las distintas especies de clasificadores (donde las especies se asocian con los conjuntos [C]) y a la vez, borrar aquellos clasificadores que no representen correctamente el conocimiento.

La activación del AG también está diseñada para favorecer la especiación. Por ello, la activación del AG en un cierto conjunto [C] depende del tiempo promedio que ha transcurrido desde la última aplicación del AG en los clasificadores de [C]. Para ello, cada clasificador mantiene un parámetro ts (*time stamp*) que almacena la iteración donde se aplicó por última vez el AG. Si el valor promedio de ts en [C] supera un cierto umbral θ_{GA}, entonces se activa el AG. Así se favorece que todas las especies tengan las mismas oportunidades de reproducción y se evita que una especie dominante acabe imponiéndose sobre el resto, uniformizando la población.

E. Subsunción

La subsunción es un operador adicional diseñado para favorecer la máxima generalización de las reglas. Habitualmente se aplica sobre los descendientes del AG. Antes de reintroducir los descendientes en la población, éstos son chequeados para ver si pueden ser subsumidos por sus padres. Esto sucede si los padres son versiones más generales que los hijos y además son suficientemente fiables (elevada experiencia y precisión). En este caso, los hijos no se reintroducen en la población y en su lugar, se incrementa la numerosidad de los padres.

IV. UCS con muestreo uniforme

En esta sección se presenta un conjunto de experimentos realizados con el sistema UCS bajo muestreo uniforme. El objetivo es demostrar empíricamente que el aprendizaje de UCS depende de la representatividad de los ejemplos de entrenamiento. Los experimentos se realizan con un problema de clasificación artificial diseñado para resaltar esta dependencia. El problema cuenta con diversas clases, y cada una de ellas cuenta con un número distinto de ejemplos representativos. En concreto, el problema, que denominamos *posición*, se define de la forma siguiente. Dada una entrada de l atributos $(x_l, ..., x_2, x_1)$ la clase asociada corresponde a la posición del bit de más peso con valor 1. En caso que no haya ningún bit con valor 1, entonces la clase asociada es 0. La tabla I muestra el conjunto óptimo de reglas que resuelve el problema posición con 12 atributos, en notación ternaria, tal como se describe en la sección III. Obsérvese que las clases 0 y 1 están representadas con tan sólo un ejemplo, mientras que la clase 12 está representada por 2048 ejemplos, es decir, la mitad de los ejemplos de entrenamiento. El objetivo es demostrar que las reglas más específicas, es decir las que cuentan con menos ejemplos representativos, son las más difíciles de aprender, mientras que las reglas generales que cuentan con más ejemplos representativos suelen obtenerse más rápidamente.

TABLA I

Conjunto óptimo de reglas de clasificación del problema posición con 12 atributos. La columna N muestra el número de ejemplos de entrenamiento que cubre cada regla. La columna % muestra el porcentaje de ejemplos que cubre cada regla con respecto al total de ejemplos de entrenamiento.

Regla de clasificación	N	%
000000000000:0	1	0.02
000000000001:1	1	0.02
00000000001#:2	2	0.05
0000000001##:3	4	0.10
000000001###:4	8	0.20
00000001####:5	16	0.39
0000001#####:6	32	0.78
000001######:7	64	1.56
00001#######:8	128	3.13
0001########:9	256	6.25
001#########:10	512	12.5
01##########:11	1024	25
1###########:12	2048	50

Se ha realizado un conjunto de experimentos con el problema posición desde 8 hasta 15 atributos. Para cada problema, se lanzan 10 ejecuciones con el sistema UCS y se promedian los resultados. Los valores de los parámetros de configuración del sistema UCS son los siguientes (ver [7] para consultar la notación de cada parámetro): $P_{\#} = 0.33$, $\nu = 10$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $doGASubsumption = yes$, $doASSubsumption = no$, $\theta_{sub} = 20$, $specify = yes$, $N_{sp} = 20$, $P_{sp} = 0.5$, $acc_0 = 0.99$. La figura 2 muestra el rendimiento en aprendizaje de UCS a lo largo de las iteraciones para el problema posición desde 8 hasta 15 atributos. Cada curva representa el porcentaje de población

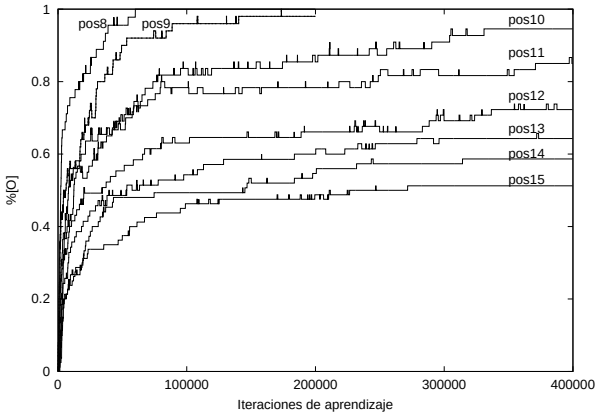

Fig. 2. Porcentaje de la población óptima obtenida por UCS en el problema posición con muestreo uniforme.

óptima conseguida por el sistema en cada iteración, que se denota como %[O]. Es decir, se muestra qué porcentaje de reglas óptimas (como las ilustradas en la tabla I) ha obtenido el sistema clasificador. Esta métrica es útil para tener una medida más aproximada del rendimiento en todas las regiones de clasificación. Si usáramos exclusivamente el porcentaje de aciertos, las regiones de clasificación con pocos ejemplos representativos (como las pertenecientes a las clases 0 y 1) tendrían muy poca influencia en esta medida. Si el sistema clasificador no es capaz de predecir correctamente estas clases, prácticamente no se notaría en el porcentaje de aciertos, aparentando un aprendizaje prácticamente completo[1].

Los resultados que se presentan en la figura 2 demuestran que el aprendizaje es más lento a medida que aumenta la complejidad del problema. A primera vista podría parecer que la complejidad del problema depende exclusivamente del número de atributos. Pero de hecho, varios experimentos aportados en la bibliografía demuestran que la complejidad depende más directamente de la representatividad de los ejemplos para cada regla óptima que se tiene que evolucionar. De hecho, experimentos realizados con el problema del multiplexor con 11 atributos tienen una complejidad mucho inferior al problema posición con 11 atributos [7], [6]. Asimismo, experimentos realizados con el problema de la paridad demostraban la escasa influencia del número de atributos en la complejidad del aprendizaje [12], [6]. A partir de estos estudios, se ha propuesto que la complejidad del aprendizaje en sistemas clasificadores sigue la fórmula [7]:

$$t_{learning} = max_{o \in [O]} \frac{c_o}{f_o} \quad (1)$$

donde $t_{learning}$ modela la complejidad del sistema en número de iteraciones, o es cada regla de la población óptima [O] que se tiene que evolucionar para obtener un aprendizaje completo, c_o es una constante de complejidad que depende de la regla y del problema y f_o es la representatividad de los ejemplos para esta regla. Es decir, f_o es el número de ejemplos representativos de la regla con respecto al total de ejemplos de entrenamiento. Esta fórmula explica que cada región de clasificación tiene una complejidad distinta que depende del porcentaje de ejemplos representativos. El aprendizaje completo se consigue cuando se obtienen todas las reglas y por tanto, depende de la región de clasificación más difícil.

En la tabla II se muestra la población final obtenida por UCS en una ejecución del problema pos12. La población consta de 11 reglas precisas y el resto que pertenecen a reglas que se están explorando recientemente. Estas últimas reglas no se han mostrado en su totalidad por brevedad. Esta población contiene 9 de las 13 reglas óptimas. Se observa que el sistema ha desarrollado las reglas más generales, es decir las que cuentan con más ejemplos. En concreto: ha obtenido las reglas para las clases: 12-5 y 3. Para la clase 4 existen dos versiones precisas que no están todavía suficientemente generalizadas. Para el resto de clases (3,2,1,0) todavía no se ha encontrado ninguna aproximación precisa. La columna *tb* de la tabla muestra la iteración de entrenamiento en que se descubre la regla. Mediante esta medida, se puede observar que existe una correlación directa entre la iteración en que se descubre una regla y el número de ejemplos representativos de la misma: a más ejemplos representativos, la regla se descubre antes. Este resultado confirmaría la hipótesis formulada en (1). El problema que esto conlleva es que mediante muestreo uniforme seguiremos mostrando al sistema clasificador entradas pertenecientes a las reglas más generales. Por ejemplo, la clase 12 está representada por un 50% de ejemplos sobre el total. Estas entradas no serían estrictamente necesarias si el sistema ya ha aprendido la regla asociada. Si aumentamos la frecuencia de muestreo a los ejemplos peor clasificados (como los pertenecientes a las clases 0,1,2) estaremos favoreciendo que el clasificador acelere el aprendizaje en estas regiones. La próxima sección lo analiza.

[1] Otra solución alternativa a la métrica %[O] sería usar el porcentaje de aciertos ponderado, el cual calcula el porcentaje de aciertos por cada clase y posteriormente, calcula el promedio. Se ha optado por la métrica %[O] puesto que además da idea del número de reglas que se han aprendido con respecto al total esperado. Para problemas reales donde no se conoce la población óptima, debería usarse el porcentaje de aciertos ponderado.

TABLA II

POBLACIÓN OBTENIDA POR UCS EN EL PROBLEMA POS12 EN LA ITERACIÓN 400,000 BAJO MUESTREO UNIFORME. LAS REGLAS ESTÁN ORDENADAS POR NUMEROSIDAD. POR CADA REGLA SE MUESTRA: LA CONDICIÓN Y CLASE, LA PRECISIÓN (ACC), EL FITNESS (F), EL TAMAÑO DE LA ESPECIE (NS), LA EXPERIENCIA (EXP), LA ITERACIÓN DONDE SE CREA LA REGLA (TB) Y LA NUMEROSIDAD (N).

regla	acc	F	ns	exp	tb	n
1##########:12	1.00	1.00	51.00	199180	1468	51
01##########:11	1.00	1.00	47.04	99588	1827	47
001##########:10	1.00	1.00	42.00	49602	2253	42
0001#########: 9	1.00	1.00	41.20	24788	5153	41
00001########: 8	1.00	1.00	36.24	12331	8447	35
000001#######: 7	1.00	1.00	25.12	5835	19696	25
0000001######: 6	1.00	1.00	24.29	2981	20033	23
00000001#####: 5	1.00	1.00	17.82	1283	67322	17
000000001##0: 4	1.00	1.00	10.48	176	252345	10
000000001##1: 4	1.00	1.00	6.41	197	195694	6
0000000001##: 3	1.00	1.00	6.99	87	323838	5
#0#0#0000000: 0	0.20	0.00	14.39	16	393154	3
0#000001####: 5	0.70	0.03	19.31	11	399120	1
			...			
1001########: 9	0.00	0.00	36.83	17	399773	1

TABLA III

POBLACIÓN OBTENIDA POR UCS EN EL PROBLEMA POS12 EN LA ITERACIÓN 400,000 CON MUESTREO ADAPTATIVO. LAS REGLAS ESTÁN ORDENADAS POR NUMEROSIDAD. POR CADA REGLA SE MUESTRA: LA CONDICIÓN Y CLASE, LA PRECISIÓN (ACC), EL FITNESS (F), EL TAMAÑO DE LA ESPECIE (NS), LA EXPERIENCIA (EXP), LA ITERACIÓN DONDE SE CREA LA REGLA (TB) Y LA NUMEROSIDAD (N).

regla	acc	F	ns	exp	tb	n
1##########:12	1.00	1.000	58.000	198690	927	58
01##########:11	1.00	1.000	52.550	99008	1070	52
001##########:10	1.00	1.000	48.000	49607	2235	48
0001#########: 9	1.00	1.000	33.025	24709	4009	33
00001########: 8	1.00	1.000	29.934	12425	10850	29
000001#######: 7	1.00	1.000	26.060	6053	11628	25
0000001######: 6	1.00	1.000	19.062	3046	26427	19
00000001#####: 5	1.00	1.000	18.641	1567	21469	17
00000000001#: 2	1.00	1.000	9.034	71	316377	9
000000001###: 4	1.00	1.000	9.029	191	303046	9
0000000001##: 3	1.00	1.000	3.772	356	54341	3
#00000000000: 0	0.78	0.081	2.079	46	349007	2
000000000001: 1	1.00	1.000	2.100	70	115567	2
			...			
00#0#1###0##: 7	0.22	0.000	34.011	10	399820	1

V. UCS CON MUESTREO ADAPTATIVO

En esta sección, se altera la frecuencia de muestreo de los ejemplos de acuerdo con los resultados parciales del sistema UCS. Este cambio no implica ninguna modificación en el algoritmo de aprendizaje de UCS. Tan sólo se altera el algoritmo de muestreo de los ejemplos. Concretamente, el algoritmo de muestreo se basa en que cada ejemplo de entrenamiento $\mathbf{x}_i$ tenga asociado un peso p_i. Inicialmente, cada peso tiene el mismo valor. En cada iteración de aprendizaje:
1. Se suministra un ejemplo $\mathbf{x}_i$ al sistema UCS, con una probabilidad de muestreo proporcional al peso. Si todos los pesos son iguales, el muestreo es uniforme.
2. UCS propone una clasificación y de acuerdo con la precisión de la misma, se actualiza el peso del ejemplo:

$$p_i = p_i * (1 \pm \alpha)$$

Si la clasificación es incorrecta, el peso aumenta su valor. Si por el contrario es correcta, el peso disminuye su valor. De esta forma, los futuros muestreos tenderán a potenciar los ejemplos con peso más elevado, es decir, aquellos que han sido clasificados incorrectamente.

Para comparar los resultados del muestreo adaptativo con los del muestreo uniforme, se han ejecutado las mismas pruebas que en la sección anterior. Para ello, se ha dimensionado el parámetro $\alpha = 0.1$ para todos los experimentos. Un valor para este parámetro muy elevado puede llevar a oscilaciones e inestabilidades en el valor de los pesos. Con el valor escogido se espera conseguir una variación moderada de los pesos de una iteración a la siguiente, pero reflejando a la larga la dificultad promedio para clasificar el ejemplo de entrenamiento correspondiente.

La figura 3 compara los resultados obtenidos con los dos métodos de muestreo para el problema posición desde 8 hasta 15 atributos. Cada gráfica representa el problema posición con un número determinado de atributos. Las curvas que se muestran son el porcentaje de la población óptima obtenida durante el aprendizaje, usando muestreo uniforme y adaptativo. En todos los casos, se observa una mejoría en cuanto a velocidad de convergencia usando el muestreo adaptativo. Es decir, el muestreo adaptativo permite aprender con anterioridad todo el problema, debido a que se refuerza el aprendizaje para las reglas de clasificación más difíciles. Al igual que sucede con la figura 2, el aprendizaje es más lento a medida que aumentamos el número de atributos de entrada. El muestreo adaptativo no evita este efecto, pero sí permite aumentar la velocidad de convergencia en todos los casos en relación con el muestreo uniforme. En los problemas más difíciles (desde 13 a 15 atributos) la mejoría obtenida con muestreo adaptativo es moderada, debido probablemente a que la variación de los pesos es suave. En estos casos, se podrían estudiar esquemas con variaciones de peso más bruscas o bien esquemas alternativos de modificación de los pesos.

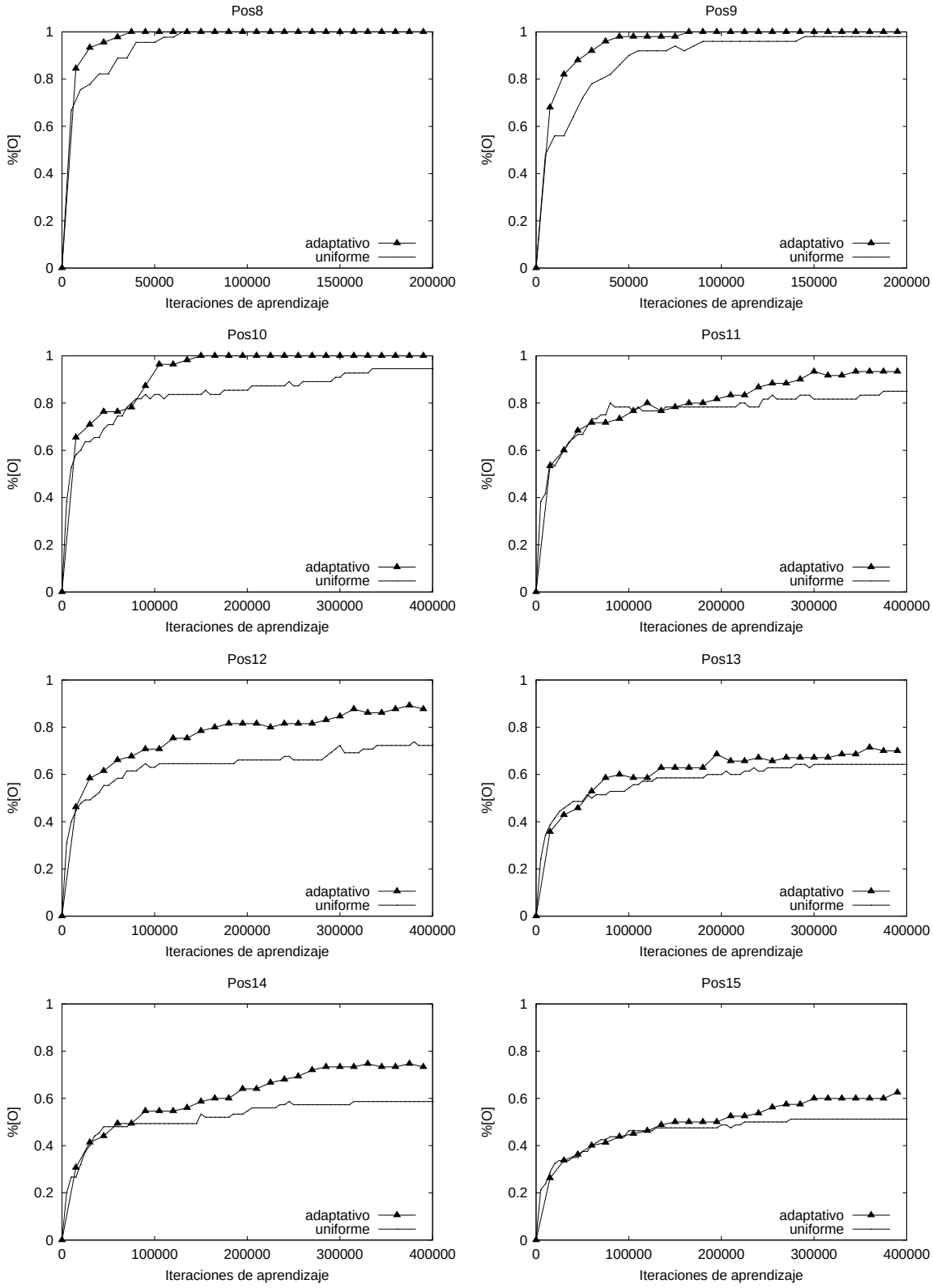

Fig. 3. Comparación entre los esquemas de muestreo uniforme y muestreo adaptativo para el sistema UCS con el problema posición desde 8 hasta 15 atributos.

La tabla III muestra la población obtenida en una ejecución del problema pos12 con muestreo adaptativo. Esta ejecución es homóloga a la presentada en la tabla II para el muestreo uniforme. Se puede observar que bajo muestreo adaptativo se han evolucionado prácticamente todas las reglas de la población, a excepción de la regla perteneciente a la clase 0 que todavía necesita especializar el único bit que tiene con '#'. Comparando el campo *tb* de las dos tablas, que representa la iteración en que se descubre cada regla, se observa que el muestreo adaptativo permite acelerar el descubrimiento de las regiones de clasificación más difíciles. Se observa que las reglas más específicas (por ejemplo, desde la clase 0 hasta 5) se descubren con anterioridad.

VI. Conclusiones

Este trabajo presenta una primera aproximación al *boosting* aplicado y adaptado para sistemas clasificadores tipo Michigan. El método *boosting* entrena múltiples clasificadores, con una distribución de los ejemplos de entrenamiento que depende del porcentaje de clasificación de los sistemas clasificadores que entrena. El clasificador combinado que resulta de la múltiples iteraciones tiene una capacidad predictiva superior a los clasificadores individuales. En este trabajo, se aplica un método similar con dos objetivos: mejorar la clasificación global del sistema clasificador y aumentar la velocidad de aprendizaje. Se identifican las zonas del espacio de búsqueda más complejas como aquellas que están representadas con pocos ejemplos de entrenamiento. Se propone un método de muestreo que, aprovechando el aprendizaje incremental del sistema clasificador, usa los resultados parciales de clasificación y adapta el muestreo para reforzar el aprendizaje en las regiones de clasificación más difíciles.

Los resultados de este trabajo son una primera muestra de la viabilidad de la propuesta con respecto a los objetivos planteados. Los experimentos se han realizado con un problema diseñado artificialmente para potenciar la presencia de clases (o regiones de clasificación) poco representativas en el conjunto de entrenamiento. Los resultados demuestran que el sistema aprende mucho más rápido las regiones de clasificación más difíciles, acelerando por tanto el aprendizaje global del sistema. Esto repercute directamente en el porcentaje de clasificación, ya que a igual tiempo de entrenamiento, un método con muestreo adaptativo es capaz de obtener porcentajes de clasificación superiores a los del muestreo uniforme.

Como trabajo futuro se desea ampliar el trabajo a problemas de clasificación reales, extendiéndonos asimismo al estudio de la capacidad de generalización del sistema. Igualmente, el trabajo queda abierto al estudio de esquemas alternativos de muestreo adaptativo.

Agradecimientos

El autor agradece el soporte de *Enginyeria i Arquitectura La Salle* para la realización de este trabajo, así como el soporte del Ministerio de Ciencía y Tecnología bajo el proyecto TIC2002-04036-C05-03 con participación de fondos FEDER y el soporte del *Departament d'Universitats, Recerca i Societat de la Informació* para grupos de investigación consolidados de Catalunya 2002-SGR-00/55.

Referencias

[1] Stewart W. Wilson, "Classifier Fitness Based on Accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[2] Stewart W. Wilson, "Generalization in the XCS Classifier System," in *Genetic Programming: Proceedings of the Third Annual Conference*, J.R. Koza and et al., Eds. 1998, pp. 665–674, San Francisco, CA: Morgan Kaufmann.

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

[4] John H. Holland, "Adaptation," in *Progress in Theoretical Biology*, R. Rosen and F. Snell, Eds. 1976, vol. 4, pp. 263–293, New York: Academic Press.

[5] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: The MIT Press/Bradford Books, 1998.

[6] Ester Bernadó Mansilla, *Contributions to Genetic Based Classifier Systems*, Ph.D. thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona, Spain, 2002.

[7] Ester Bernadó Mansilla and Josep M. Garrell Guiu, "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks," *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.

[8] Robert E. Schapire, "A brief introduction to boosting," in *Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[9] Yoav Freund and Robert E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, August 1997.

[10] M.V. Butz and S.W. Wilson, "An algorithmic description of XCS," in *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds. 2001, vol. 1996 of *Lecture Notes in Artificial Intelligence*, pp. 253–272, Springer-Verlag Berlin Heidelberg.

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

[12] Tim Kovacs and Manfred Kerber, "What makes a problem hard for XCS?," in *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds. 2001, vol. 1996 of *Lecture Notes in Artificial Intelligence*, pp. 80–99, Springer-Verlag Berlin Heidelberg.