Índice General

Introducción **9**

1 Preliminares **13**

1.1 El aprendizaje inductivo de conceptos 13

1.1.1 Tipos de aprendizaje inductivo 16

1.1.1.1 La adquisición de conceptos frente a la generalización descriptiva 16

1.1.1.2 Descripción característica frente a la descripción discriminante 18

1.1.1.3 Aprendizaje de un único concepto frente al aprendizaje de múltiples conceptos 18

1.1.2 El conocimiento subyacente 19

1.2 Adquisición de conocimiento difuso 22

1.3 Descripción de algunos algoritmos de aprendizaje 24

1.3.1 Basados en árboles de decisión 24

1.3.2 Basados en la metodología STAR 26

1.4 La aplicación de la computación evolutiva al aprendizaje automático 29

1.4.1 Algoritmos genéticos 31

1.4.1.1 Estructura de un AG 32

1.4.1.2 Representación 33

1.4.1.3 Mecanismo de Selección 33

1.4.1.4 Operadores Genéticos 34

1.4.2 Algoritmos genéticos con codificación real 36

1.4.2.1 Operadores de Cruce 37

1.4.2.2 Operadores de Mutación 38

1.4.3 Modelos de aprendizaje basados en algoritmos genéticos 38

1.4.4 Sistemas clasificadores (SCs) 40

1.4.5 El modelo Pittsburgh en el aprendizaje automático de conceptos . . 42

1.4.5.1 El sistema de aprendizaje GABIL 44

1.4.5.2 El sistema de aprendizaje GIL 45

2 Una metodología genética para el aprendizaje de reglas difusas 47

2.1 Un enfoque iterativo para el aprendizaje de conceptos 47

2.2 SLAVE: un algoritmo de aprendizaje inductivo basado en el modelo iterativo genético 51

2.2.1 Representación del conocimiento 53

2.2.2 Concepto de adaptación 55

2.2.3 Ejemplos positivos y negativos 59

2.2.4 Selección de reglas 61

2.2.4.1 Grado de completitud 65

2.2.4.2 Combinación de consistencia y completitud 66

2.2.4.3 Simplicidad y comprensibilidad 67

2.2.5 El algoritmo genético de SLAVE 71

2.2.5.1 Representación de un elemento de la población 71

2.2.5.2 Generación de la población inicial 73

2.2.5.3 Función de evaluación (FITNESS) 74

2.2.5.4 Los operadores genéticos 74

2.2.5.5 Condición de terminación 77

2.2.6 Penalización de reglas 77

2.3 Mecanismo de resolución de conflictos 80

2.4 Pruebas experimentales 81

2.4.1 Propuesta original 82

2.4.2 Propuesta original con los nuevos operadores genéticos 87

3 Colaboración entre las soluciones parciales 91

3.1 Inconvenientes del modelo iterativo 91
3.2 Algoritmo de refinamiento de teorías difusas 95
3.2.1 Refinamiento de reglas difusas con consecuente crisp 97
3.2.2 Refinamiento para reglas con consecuente difuso 101
3.2.2.1 Un ejemplo del funcionamiento del algoritmo de refinamiento 104
3.2.3 Estudios experimentales 106
3.2.3.1 Comportamiento del algoritmo de refinamiento 107
3.3 Incorporación de colaboración/competición entre reglas en SLAVE 111
3.3.1 Colaboración/competición en el caso crisp 112
3.3.1.1 Pruebas experimentales 121
3.3.2 Información necesaria para consecuente continuo 123
3.3.2.1 Aciertos y fallos en el caso difuso 125
3.3.2.2 Pruebas experimentales 128

4 El problema de la selección de características 129
4.1 La selección de características 129
4.2 Modelos de selección de subconjuntos de características 132
4.3 Algoritmo genético de dos niveles 134
4.3.1 Representación de un elemento de la población 136
4.3.2 La generación de la población inicial 137
4.3.3 Operadores genéticos 138
4.4 Pruebas experimentales 138
4.5 Algoritmo genético de dos niveles con información 139
4.5.1 Medidas de información 141
4.5.2 Añadir las medidas de información a SLAVE 143
4.6 Pruebas experimentales 146

5 Aprendizaje de reglas con modificadores semánticos 153
5.1 Modelos difusos descriptivos y aproximativos 153
5.2 Modificadores semánticos 159

5.3 Redefinir SLAVE como un modelo difuso híbrido 164
5.3.1 Los modificadores semánticos de SLAVE 165
5.3.1.1 Modificación de la forma de una etiqueta 165
5.3.1.2 Modificación de la anchura de una etiqueta 166
5.3.1.3 Modificación por desplazamiento de una etiqueta 169
5.3.2 Composición de los modificadores semánticos 170
5.3.3 Aproximación genética a la alteración de las etiquetas 171
5.4 Pruebas experimentales 180

Conclusiones y trabajos futuros **185**

A Bases de datos con variable de clasificación crisp **191**

B Bases de datos con variable consecuente difuso **197**

C Algoritmos de aprendizaje utilizados para la confección de la parte experimental **203**

Bibliografía **204**

Índice de Figuras

1 Evolución de SLAVE por capítulos 11

1.1 Proceso de transmisión de conocimiento 13
1.2 Diagrama de un modelo de aprendizaje inductivo 14
1.3 Mecanismo de selección 34
1.4 Cruce simple 35
1.5 Mutación 35
1.6 Organización general de un sistema clasificador. 40

2.1 Envolvente convexa de etiquetas difusas 56
2.2 Estructura del espacio de soluciones parciales 62
2.3 Dominio de las variables X_1, X_2, X_3 69
2.4 Operador de cruce sobre dos puntos 75
2.5 Operador de rotación 75
2.6 Operador de Generalización 76
2.7 Comparativa del acierto en los conjuntos de prueba (1) 85
2.8 Comparativa del acierto en los conjuntos de prueba (2) 88

3.1 Estructura del algoritmo de aprendizaje SLAVE+R 97
3.2 Algoritmo de refinamiento 98
3.3 Funcionamiento del algoritmo de refinamiento 105
3.4 Conservación de la simplicidad en el algoritmo de refinamiento 106
3.5 Comparativa del acierto en los conjuntos de prueba (3) 108
3.6 Superficies obtenidas para el problema del péndulo 111

3.7 Comparativa del acierto en los conjuntos de prueba (4) 122
3.8 Dominio intervalar con solapamientos 124

4.1 Esquema de un modelo por filtro 132
4.2 Esquema de un modelo por envoltura 133
4.3 Esquema de SLAVE con dos niveles 136
4.4 Comparativa del acierto en los conjuntos de prueba (5) 140
4.5 Comparativa del acierto en los conjuntos de prueba (6) 149

5.1 Dominio de la variable X 155
5.2 Reglas obtenidas por SLAVE 156
5.3 Reglas obtenidas por SLAVE para los dos problemas 157
5.4 Etiqueta lingüística *aproximadamente 3* 160
5.5 Operador $\mu_L(x)^a$ 167
5.6 Operador L_b 169
5.7 Operador $\beta_3(L, c)$ 170
5.8 Solución al problema 2 usando el operador $\beta_2(label2, -0.6)$ 171
5.9 Representación del conjunto de soluciones en el AG 172
5.10 Coevolución sobre cinco algoritmos genéticos 173
5.11 Dominio de la variable X 174
5.12 Representación del valor asignado a la variable X 175
5.13 Distintas aproximación obtenidas para la base de datos MONTE 182
5.14 Evolución del algoritmo de aprendizaje SLAVE 186

B.1 Representación de la base de datos SIN 198
B.2 Representación de la base de datos PARABOLA 199
B.3 Representación de la base de datos 3DPARABOLA 200
B.4 Representación de la base de datos MONTE 200
B.5 El problema del péndulo 201
B.6 Superficie original del problema del péndulo 201

Índice de Tablas

2.1 Tamaño del espacio de búsqueda para los distintos modelos de aprendizaje con genéticos. 52

2.2 Parámetros fijados para la ejecución de SLAVE 82

2.3 Resultados de la propuesta original para caso crisp 82

2.4 Resultados obtenidos con C4.5, CN2 y LVQ 83

2.5 Resultados de SLAVE para el caso difuso 86

2.6 Resultados obtenidos con el algoritmo WM 86

2.7 Parámetros adicionales fijados para la ejecución de SLAVE 87

2.8 Resultados de SLAVE caso crisp incluyendo los nuevos operadores genéticos 87

2.9 Resultados de SLAVE para el caso difuso incluyendo los nuevos operadores genéticos 89

3.1 Resultados de SLAVE+R en el caso crisp 107

3.2 Resultados usando el refinamiento como un algoritmo de aprendizaje . . . 109

3.3 Resultados de SLAVE+R para el caso difuso 110

3.4 Conjunto de reglas obtenido por SLAVE+R para el problema del péndulo . 110

3.5 Grados de recubrimiento y peso de las reglas asociados a cada ejemplo de E dado el conjunto de reglas REGLAS 118

3.6 Compatibilidad de cada ejemplo con la regla $R_{B_2}(A)$. 118

3.7 Modificación de la tabla de recubrimientos 119

3.8 Resultados con la nueva versión de SLAVE 121

3.9 Resultados de SLAVE para el caso difuso con la nueva función de evaluación de reglas 128

4.1 Resultados con la nueva versión de SLAVE 139

4.2 Resultados de SLAVE dos niveles con copia 147
4.3 Resultados de SLAVE dos niveles con umbral de activación 147
4.4 Resultados obtenidos sobre las nuevas bases de datos con consecuente crisp 150
4.5 Resultados obtenidos sobre la base de datos MONTE+ 152

5.1 Características de los algoritmos genéticos implicados en la coevolución . . 177
5.2 Parámetros fijados para la ejecución de SLAVE con modificadores semánticos 180
5.3 Resultados de SLAVE con modificadores semánticos para el caso crisp . . . 181
5.4 Resultados de SLAVE con modificadores semánticos para el caso difuso . . 181

A.1 Base de datos IRIS 192
A.2 Base de datos INFARTO 193
A.3 Base de datos MONK 193
A.4 Base de datos WINE 194
A.5 Base de datos PIMA 195

Introducción

La capacidad de aprender es una de las características centrales de la inteligencia, y esto hace que el aprendizaje haya formado parte de la investigación en Inteligencia Artificial desde prácticamente los orígenes de la misma.

Las aplicaciones del aprendizaje son hoy en día muy numerosas. En concreto, es importante destacar la gran aportación que los algoritmos de aprendizaje tienen en el desarrollo práctico de algunos sistemas basados en el conocimiento, ya que como es conocido, las técnicas de elicitación son de difícil aplicación en muchos casos. El aprendizaje permite, en estos casos, una extracción del conocimiento mucho más eficaz. Este conocimiento podrá ser posteriomente validado por un experto.

La utilización de la lógica difusa, por otra parte, ha demostrado ser una herramienta extraordinariamente útil en la descripción y modelación de muchos sistemas. La utilización de dicha lógica ha permitido la modelización cualitativa de sistemas. Esta modelización es en muchos más comprensible y simple que otras alternativas. Sin embargo, el éxito comercial de muchos sistemas basados en el conocimiento descrito mediante la lógica difusa hace necesario el desarrollo de nuevos modelos de aprendizaje que permitan adquirir tal conocimiento. Por supuesto, ya se ha iniciado el trabajo, y desde distintos puntos de vista se han aportado modelos de aprendizaje que permiten identificar sistemas difusos. En concreto, en este trabajo, nos proponemos como objetivo el desarrollar un modelo de aprendizaje de reglas difusas basado en la utilización de algoritmos genéticos.

Objetivos

El objetivo de esta memoria es el desarrollo de un algoritmo de aprendizaje inductivo con las siguientes características:

- sea capaz de obtener un conocimiento que, además de proporcionar una buena descripción del problema, sea fácilmente interpretable desde un punto de vista humano. En este sentido, las reglas difusas son una forma de representación que permite dotar a dicho conocimiento de un alto grado de comprensibilidad,

- obtenga un buen comportamiento ante sistemas complejos, es decir, ante sistemas con un número elevado de variables o ante sistemas fuertemente afectados por la imprecisión de los datos o por el ruido,
- sea aplicable a problemas cuya variable consecuente toma valores tanto sobre un referencial nominal o discreto como sobre un referencial continuo. En este sentido, el algoritmo que se desea desarrollar pasa a ocupar un vacío existente en el campo de los algoritmos de clasificación, ya que los algoritmos que se pueden encontrar en la bibliografía se han desarrollodo para abordar uno de estos dos problemas, presentando, en general, un mal comportamiento sobre el otro. Por consiguiente, se desea desarrollar una nueva metodología de aprendizaje que sea válida para ambos tipos de problemas de clasificación.

Además, en el proceso de búsqueda asociado al algoritmo, queremos explorar el uso de algoritmos genético en el desarrollo de sistemas de aprendizaje. En este sentido, exploraremos un nueva vía de aplicación de los algoritmos genéticos al problema de la clasificación, que sea una alternativa a los modelos Michigan y Pittsburgh, que permita reducir el espacio de búsqueda, y por consiguiente el tiempo de respuesta del sistema.

Descripción por capítulos

En el primer capítulo, situamos el entorno en el que se ha desarrollado este trabajo, definiendo los conceptos básicos que se utilizarán en el desarrollo del algoritmo. Además, se describen las metodologías más utilizadas en el desarrollo de algoritmos de clasificación, incluyendo aquellas que tienen como componente fundamental los algoritmos genéticos.

En la figura 1 se muestra la evolución que a lo largo de esta memoria se realizará del algoritmo de aprendizaje. Así, en el capítulo 2 se propone la metodología básica del algoritmo de clasificación, que llamaremos SLAVE, describiendo los elementos básicos que componen el sistema de aprendizaje.

La metodología propuesta está basada en la inclusión de un algoritmo genético como componente principal de un modelo de aprendizaje iterativo. Este modelo aprende de forma aislada cada una de las reglas necesarias para describir el sistema. Este hecho conlleva que las reglas obtenidas sean buenas reglas individuales, pero no contempla cómo es su funcionamiento con respecto al resto de las reglas que describen el conocimiento. Por esta razón, en el capítulo 3 sugerimos dos posibles vías para mejorar el comportamiento del algoritmo mediante la inclusión de mecanismos que permitan la colaboración en unos casos, y la competición en otros, de las reglas aprendidas. La primera de las vías propuestas consiste en la inclusión de un algoritmo de refinamiento de teorías como fase final del algoritmo de aprendizaje. Este algoritmo de refinamiento será el encargado de establecer

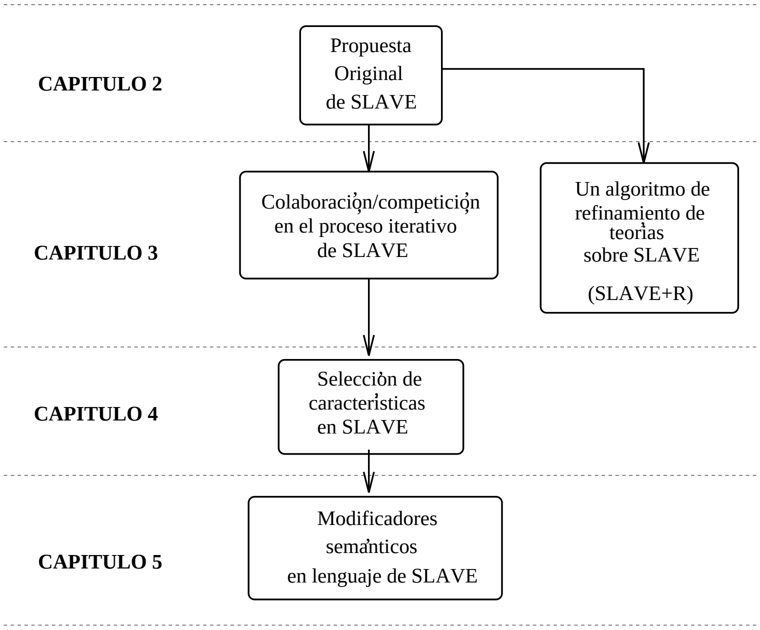


Figura 1: Evolución de SLAVE por capítulos

de forma conveniente la colaboración o competición entre las reglas, teniendo como objetivo no sólo mejorar la capacidad de predicción de las mismas, sino también de mejorar en la medida de lo posible la comprensibilidad del conocimiento extraído. Denominamos SLAVE+R a la inclusión del algoritmo de refinamiento de teorías propuesto como fase final del proceso de aprendizaje de SLAVE.

Como segunda vía, se contempla la posibilidad de que sea el propio algoritmo el que incorpore la colaboración/competición entre las reglas durante el proceso de aprendizaje. Con tal fin, se propone un nuevo mecanismo de evaluación de reglas, que contempla el conocimiento previamente extraído para los conceptos ya aprendidos, para determinar en qué medida las nuevas reglas que se proponen afectarán a la clasificación de los conceptos. La idea es en definitiva añadir más información al modelo iterativo en el que se basa SLAVE.

En el capítulo 4, se describe una nueva modificación de SLAVE con el fin de aumentar su capacidad de detección de variables irrelevantes. Así, en el módulo de selección de reglas se establecen dos procesos de búsqueda diferentes, uno que trata de encontrar el subconjunto de variables relevantes que formarán parte de la descripción de la regla, y otro que intenta encontrar la mejor asignación de valores a estas variables. Estas dos búsquedas se llevan a cabo de forma simultánea mediante la ejecución de forma paralela de dos algoritmos genéticos que mediante coevolución tratan de encontrar la mejor descripción para un concepto. Así, se consigue no sólo agilizar la selección, sino que además, se aumenta el nivel de exploración del espacio de los subconjuntos de variables relevantes. En este capítulo se proponen para su estudio, tres formas posibles de llevar a cabo esta modificación.

En el capítulo 5 nos planteamos finalmente la posibilidad de enriquecer el lenguaje de representación de SLAVE incluyendo modificadores semánticos. Con esta modificación intentaremos dotar al algoritmo de la capacidad de alterar la semántica de sus etiquetas iniciales, de manera que estas respondan lo más posible a la información que contienen los ejemplos.

La parte experimental de esta memoria se ha hecho seleccionando un conjunto de bases de datos y una serie de algoritmos de aprendizaje para comprobar el comportamiento de SLAVE. Las bases de datos estudiadas y los algoritmos de aprendizaje utilizados se describen en los apéndices A, B y C. En cada capítulo se recoge una parte experimental con el fin de probar cada nuevo sistema diseñado y de justificar la siguiente modificación del algoritmo.

Capítulo 1

Preliminares

1.1 El aprendizaje inductivo de conceptos

La adquisición de conocimiento es uno de los mayores problemas que se ha encontrado en el desarrollo de los sistemas basados en el conocimiento. Los elementos básicos que aparecen en los métodos tradicionales de elicitación son: **el experto**, que normalmente es una persona o conjunto de personas de las cuales se desea elicitar el conocimiento, y **el elicitador**, que es la persona o conjunto de personas que mediante técnicas de entrevista tratan de adquirir dicho conocimiento. Otros elementos importantes son **el modelo de comunicación**, que especifica la estructura en la cual el conocimiento se transmite entre experto y elicitador, y **la representación del conocimiento**, que indica la forma en la que el conocimiento adquirido se almacena. La figura 1.1 ([39]) muestra el proceso de transmisión del conocimiento:

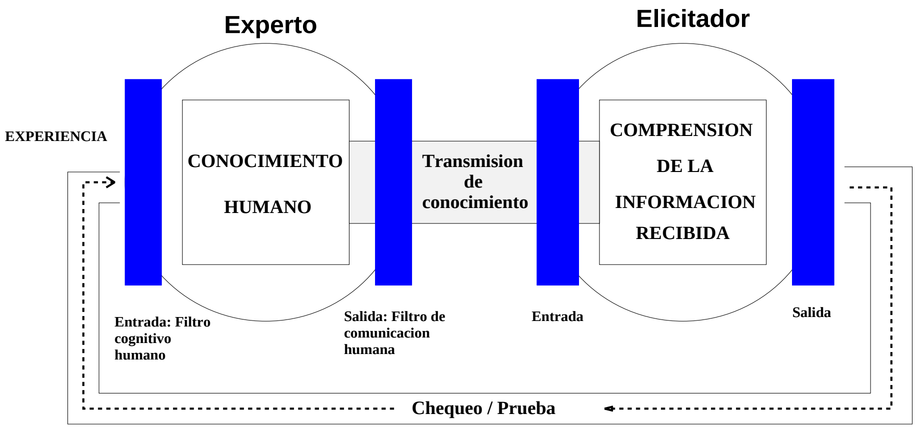


Figura 1.1: Proceso de transmisión de conocimiento

La elicitación de este conocimiento utilizando técnicas de entrevista [39] es una tarea compleja que consume mucho tiempo. Una alternativa a estos métodos tradicionales, que ha tenido recientemente un amplio desarrollo, consiste en la utilización de técnicas de aprendizaje automático [114, 115, 118, 125].

Uno de los modelos más estudiados es el aprendizaje inductivo, que consiste en un proceso de adquisición de conocimiento mediante la aplicación de inferencias inductivas sobre un conjunto de hechos. El conjunto de hechos representa el modelo de comunicación del conocimiento que, o bien es propuesto basándose en la experiencia de un experto, o bien se extrae directamente del sistema que se desea aprender. El algoritmo de aprendizaje toma el papel del elicitador usando para ello un modelo de representación del conocimiento. Un diagrama general de estos modelos se describe en la figura 1.2.

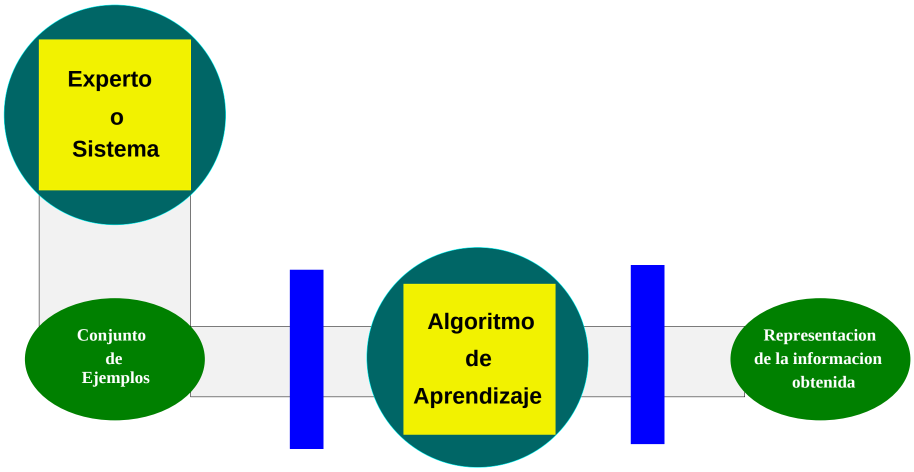


Figura 1.2: Diagrama de un modelo de aprendizaje inductivo

La inducción, que es una de las formas más comunes de aprendizaje, se basa en un fundamento bastante débil, ya que en principio, la adquisición de conocimiento no se puede validar completamente. Las aserciones obtenidas mediante inducción producen un número potencialmente infinito de hipótesis, sobre un número finito de comprobaciones que se pueden realizar.

La inducción tradicional trata de establecer cual es el mejor criterio para guiar la solución de las aserciones inductivas y cómo éstas se pueden confirmar. La búsqueda de respuestas a tales preguntas se ha convertido en un campo de discusión entre filósofos y lógicos. En este sentido, muchos filósofos tales como Karl Popper, Beltrand Russell y George Polya afirmaron que es imposible que una máquina aprenda correctamente haciendo uso de la inducción. Debido al gran desarrollo producido en los ordenadores modernos y a las últimas investigaciones en el campo de la inteligencia artificial, las visiones pesimitas mencionadas anteriormente se estan revisando. En la actualidad se ha comprobado que los ordenadores no sólo son capaces de almacenar una gran cantidad de conocimiento sino que también son capaces de trabajar con él.

Actualmente, el problema del aprendizaje inductivo se ha simplificado en el siguiente sentido: hay que dotar a las máquinas de la capacidad de poder proporcionar hipótesis plausibles, mientras que son los seres humanos los encargados de verificar su validez.

Frente a la deducción, la inferencia inductiva parte de un conjunto de hechos o ejemplos más que de un conjunto de axiomas. Su objetivo consiste en formular hipótesis que expliquen los hechos dados y sean capaces de predecir nuevos hechos. En otras palabras, la inferencia inductiva trata de obtener una descripción completa y correcta de un fenómeno dado, a partir de un conjunto de observaciones específicas de ese fenómeno o parte de él.

De los dos aspectos de la inferencia inductiva, la generación de hipótesis plausibles y su validación, sólo el primero de ellos es el cuerpo de estudio de los investigadores en el campo del aprendizaje inductivo.

En base a lo descrito anteriormente, podemos realizar la siguiente formulación del paradigma de la inducción propuesta por Michalski R.S. en [114]:

Dados

- *Un conjunto de hechos* que representan el conocimiento específico sobre objetos, situaciones o procesos,
- *una hipótesis inductiva inicial* que puede ser nula,
- *y un conocimiento sobre el problema o conocimiento subyacente (background knowledge)* que define las restricciones y afinidades impuestas sobre las observaciones dadas, y permite generar hipótesis inductivas candidatas, y que además, puede incluir alguna información sobre la teoría del dominio que permite a su vez definir criterios de preferencia para caracterizar las propiedades deseables de la hipótesis inducida.

Encontrar

- *Una hipótesis* que tautológicamente o débilmente implique las observaciones dadas y satisfaga las restricciones y preferencias impuestas por el conocimiento subyacente.

Definición 1.1 *Una hipótesis H implica tautológicamente a un conjunto de hechos F si y sólo si F es una consecuencia lógica de H, es decir, H=>F.*

La consecuencia lógica, H=>F se puede expresar de la siguiente manera:

- H especifica a F
- F generaliza a H

Derivando el conjunto de hechos F de la hipótesis H (inferencia deductiva), la fórmula H=>F preserva la veracidad, es decir, si H es cierta, F es una consecuencia lógica de H. Si por el contrario, H se ha derivado de F (inferencia inductiva), entonces la fórmula H=>F no preserva la veracidad, aunque sí preserva la falsedad, es decir, si algunos hechos hacen falso a F, entonces estos hechos hacen falso también a H.

Para algunos problemas puede no existir ninguna hipótesis que implique tautológicamente al conjunto de hechos. Para solventar estas situaciones, aparece la noción de *implicación débil*.

Definición 1.2 *Diremos que H implica débilmente a F, si no todos los hechos de F son una consecuencia lógica de H, pero sí al menos un subconjunto de F no vacío verifica la implicación tautológica.*

Admitiendo la implicación débil, este paradigma incluye métodos para generar hipótesis "blandas", que permiten explicar con un cierto grado de incertidumbre o de forma parcial un determinado fenómeno.

Dado un conjunto de hechos, existe un número potencialmente infinito de hipótesis que implican estos hechos. Por tanto, se hace imprescindible un conocimiento subyacente que permita restringir el espacio de búsqueda, así como criterios de preferencia para reducir el número infinito de posibles elecciones.

Una forma típica para definir tales criterios, consiste en especificar las propiedades deseables de la hipótesis, tales como:

a) Requerir que la hipótesis sea la descripción consistente más corta o más económica que se deduce de los hechos. Este tipo de restricción se conoce por *sesgo en la elección* y es utilizado en lenguajes de descripción completos tales como el propuesto en [108].

b) Otra opción es usar, para expresar las hipótesis, lenguajes de descripción restringidos [117]. Esta segunda opción se conoce como *sesgo en el lenguaje*.

1.1.1 Tipos de aprendizaje inductivo

1.1.1.1 La adquisición de conceptos frente a la generalización descriptiva

Los máximos exponentes del aprendizaje inductivo son: el aprendizaje por ejemplos, también llamado *adquisición de conocimiento* y el aprendizaje por observación, también conocido por *generalización descriptiva*. Para el primero, las observaciones son caracterizaciones de algunos objetos clasificados previamente por un profesor en una o varias clases (conceptos). La hipótesis de inducción se puede ver como una regla de reconocimiento de un concepto. El segundo de ellos consiste en determinar una descripción general, ley o teoría que caracterize a un conjunto de objetos.

Como ejemplo, presentamos diferentes problemas pertenecientes a cada una de estas dos categorias:

1. **Adquisición de conocimiento**:
   - Aprender una *descripción característica* de una clase de objetos que especifique todas las propiedades comunes de los objetos conocidos de la clase. Algunos trabajos sobre este problema son [9, 20, 67, 110, 117, 158].
   - Aprender una *descripción discriminante* de una clase de objetos que permita distinguir la clase dada de un número finito de otras clases. Este problema se aborda en [108, 110, 124].
   - Inferir secuencias de reglas de extrapolación que permiten predecir el siguiente elemento en una secuencia dada por ejemplo [32, 142].
2. **Generalización descriptiva**:
   - Formular una teoría caracterizando una colección de entidades. Por ejemplo, una teoría de números, como en [98].
   - Descubrir patrones en datos obtenidos mediante observación, como por ejemplo en los trabajos [65, 146].
   - Determinar una descripción taxonómica de una colección de objetos, como por ejemplo en [112, 113].

En esta memoria nos centraremos en el problema de la adquisición de conocimiento, donde el conjunto de hechos (o ejemplos) que describen un fenómeno, se pueden ver como un conjunto de implicaciones del tipo:

$$F : \{e_{ik} \rightarrow C_i\},\ i \in I$$

donde e_{ik} es la descripción del k-ésimo ejemplo de la clase C_i, y I es un conjunto de índices para las clase C_i.

Una hipótesis inductiva H se puede caracterizar como un conjunto de reglas de reconocimiento:

$$H : \{D_i \rightarrow C_i\},\ i \in I$$

donde D_i es una descripción de la clase C_i, es decir, una expresión de condiciones de tal forma que cuando se satisfaga esta expresión para un objeto, podemos considerar a dicho objeto como una instancia de la clase C_i.

De acuerdo con la definición de hipótesis inductiva, H especifica a F, y sustituyendo H y F por las anteriores expresiones y haciendo algunas transformaciones apropiadas, obtenemos las dos siguientes condiciones que se deben satisfacer para que se verifique la definición de la hipótesis inductiva:

$$(a) \qquad \forall i \in I,\ (E_i \ \ implica \ \ D_i)$$

$$(b) \qquad \forall i, j \in I,\ (D_i \ \ implica \ \ \overline{E_j}),\ si\ j \neq i$$

donde E_i es una descripción que verifican todos los ejemplos del conjunto de entrenamiento de la clase K_i y sólo estos ejemplos.

La expresión (a) se conoce como **condición de completitud** y la expresión (b) como **condición de consistencia**. Para que una regla de reconocimiento de conceptos sea aceptable, se requiere que la hipótesis inductiva verifique estas dos condiciones. Las condiciones de completitud y consistencia proporcionan el fundamento lógico de los algoritmos para el aprendizaje de conceptos mediante ejemplos.

1.1.1.2 Descripción característica frente a la descripción discriminante

La condición de completitud y la condición de consistencia permiten explicar claramente la diferencia entre estos dos tipos de descripciones. Una descripción característica para una clase de objetos es una expresión que verifica la condición de completitud. Normalmente, esta descripción es una conjunción de algunas propiedades simples, comunes a todos los objetos de la clase. Desde un punto de vista práctico, las más interesantes son *las descripciones características maximales* (las más largas) que son las más especificas. Tales descripciones intentan discriminar la clase dada del resto de las clases.

Una descripción discriminante satisface tanto la condición de completitud como la condición de consistencia y especifica uno o más caminos para distinguir la clase dada de un número fijo de otras clases. Las más interesantes son *las descripciones discriminantes minimales* (las más cortas) que son las más generales. Estas descripciones intentan minimizar la información necesaria para identificar la clase dada entre un número fijo de otras clases.

1.1.1.3 Aprendizaje de un único concepto frente al aprendizaje de múltiples conceptos

Respecto a los ejemplos contenidos en el conjunto de entrenamiento podemos distinguir dos tipos básicos de aprendizaje, el aprendizaje de un único concepto y el aprendizaje de una colección de conceptos. En el primer caso, podemos distinguir las dos siguientes situaciones:

a) El conjunto de entrenamiento sólo contiene instancias del concepto que se debe aprender, es decir, sólo hay ejemplos positivos. En este caso no es aplicable la condición de consistencia por existir una única clase. Por consiguiente, tampoco existe un límite que permita detener la generalización de una descripción. Los algoritmos que abordan este tipo de problemas, imponen ciertas restricciones sobre las descripciones. Las más comunes son:

   – imponer que sean descripciones características maximales, como en [67, 158].

   – imponer que la descripción no exceda un cierto grado de generalidad, como en [147].

b) En el conjunto de entrenamiento hay ejemplos y contraejemplos de un determinado fenómeno. En este caso los contraejemplos se pueden tratar como ejemplos pertenecientes a una clase diferente, por lo que es aplicable la condición de consistencia y por tanto, existe un límite para generalizar las descripciones [167, 169].

En el problema del aprendizaje de múltiples conceptos, también podemos distinguir dos subtipos en función de la naturaleza de las descripciones deseables:

a) En un primer caso, se impone como restricción que todas las descripciones D_i sean disjuntas entre sí, es decir,

$$\bigcap_{i=1}^{n} D_i = \emptyset.$$

b) En un segundo caso, se permite el solapamiento entre las descripciones. En este caso, puede ocurrir que un ejemplo verifique más de una descripción. Este hecho, puede ser deseable para algunos problemas, por ejemplo, supongamos que tenemos un paciente que tiene dos enfermedades distintas, por tanto, debe ser miembro de dos clases diferentes. Cuando esto ocurre, la condición de consistencia no es aplicable. Esta situación especial requiere una reformulación de la regla de inducción, para la cual se han propuesto diferentes soluciones [26, 108, 109].

1.1.2 El conocimiento subyacente

Como mencionamos anteriormente, dado un conjunto de observaciones se puede construir un número potencialmente infinito de hipótesis inductivas que implican a dichas observaciones. Por consiguiente, se hace necesaria la incorporación de información adicional, *conocimiento sobre el problema*, para restringir el espacio de posibles descripciones y localizar las más deseables.

Ahora, describiremos los componentes más usuales introducidos como conocimiento subyacente y detallaremos algunos de ellos. Estos componentes son:

- Información sobre los descriptores.
- Criterios de preferencia que especifican las propiedades deseables de las descripciones.
- Incorporación de heurísticas y procedimientos especiales que permitan al sistema de aprendizaje generar nuevos conceptos.

Un problema subyacente, fundamental para el desarrollo de sistemas automáticos de aprendizaje inductivo, es qué información recibe la máquina y qué información se espera que la máquina aprenda. El conjunto de datos de entrada al sistema es la componente más importante del mismo, como propone el propio paradigma de inducción. Los descriptores utilizados en los datos son características observables y medidas disponibles de los objetos. Los descriptores que formarán parte de las descripciones finales pueden ser seleccionados por un profesor siguiendo algún criterio de relevancia sobre el problema específico, o bien, dados todos los descriptores disponibles, dejar que el algoritmo de aprendizaje seleccione aquellos que son relevantes para el problema concreto. En este segundo caso, el papel fundamental del aprendizaje inductivo consiste en obtener las propiedades esenciales de los objetos, también conocido como **selección de características**.

Con respecto a la capacidad de selección de características, se propone en [115] la siguiente clasificación de los algoritmos de aprendizaje:

- **Relevancia completa:** En este caso todos los descriptores proporcionados al sistema se consideran como relevantes para la tarea de aprender.
- **Relevancia parcial:** Las observaciones pueden contener un número elevado de descriptores irrelevantes o redundantes, siendo algunos de ellos relevantes. La tarea de aprender consiste en seleccionar los más relevantes y construir a partir de ellos descripciones apropiadas. Este tipo de sistemas de aprendizaje son llamados *sistemas de aprendizaje inductivo selectivo.* Algunos sistemas de esta clase son AQ11 [109] e ID3 [125].
- **Relevancia indirecta:** Las observaciones pueden contener descriptores que no son relevantes de forma directa, pero que a partir de algunos de los descriptores iniciales pueden construirse nuevos descriptores que sean relevantes de forma directa. En este caso, la tarea de aprender consiste en construir nuevos descriptores a partir de los iniciales y formular en base a ellos descripciones apropiadas. Estos sistemas son también llamados *sistemas de aprendizaje inductivo constructivo.* Algunos sistemas de esta clase son BACON [94], INDUCE [97] y AM [98].

Estos tres casos representan situaciones que progresivamente van restando importancia a la relevancia de los descriptores iniciales, (es decir, menos trabajo para la persona que define el problema) y más demanda al sistema de aprendizaje.

Además, los descriptores almacenan información del problema que se debe considerar durante el proceso de aprendizaje. Esta información puede ser:

(1) Una especificación del dominio y del tipo del descriptor.

(2) Una especificación de los operadores que se le pueden aplicar.

(3) Una especificación de las restricciones y las relaciones entre este descriptor y otros descriptores.

(4) Para descriptores numéricos, el significado, la varianza o una distribución completa de probabilidad sobre los valores que puede tomar.

(5) Una caracterización de los objetos a los cuales se les puede aplicar este descriptor.

(6) Sinónimos que pueden usarse para denotar este descriptor.

(7) Una definición de este descriptor en base a otros descriptores.

De entre todas estas informaciones que pueden incluirse como parte de un descriptor, a continuación detallaremos brevemente la (1) y la (3).

Para un problema concreto, normalmente es posible determinar el conjunto de valores que puede tomar cada descriptor. Al conjunto de estos valores se le conoce con el nombre de *dominio*. El dominio se usa para restringir el nivel de generalización que admite un descriptor.

Otra información importante para restringir el proceso de generalización está relacionado con la estructura de su dominio, es decir, con las relaciones existentes entre los elementos que lo constituyen. Atendiendo a la estructura interna de un dominio, podemos distinguir tres tipos de descriptores:

- **Nominales:** El conjunto de valores que toman los descriptores está formado por símbolos o nombres independientes.
- **Lineales:** El conjunto de valores de los descriptores lineales está completamente ordenado. Variables medidas sobre ordinales, intervalos o escalas son casos particulares de este tipo de descriptores.
- **Estructurados:** El dominio de los descriptores posee una estructuración jerarquizada que refleja las relaciones existentes entre sus valores.

Algunas otras informaciones importantes vienen impuestas en forma de restricciones. Para un problema concreto, algunas descripciones no se pueden aceptar como admisibles debido a propiedades específicas de los descriptores y a las relaciones que se establecen entre ellos. Algunos ejemplos de estas restricciones son:

- *Interdependencia entre valores.* En problemas prácticos, algunas variables especifican un estado de un objeto, mientras otras variables especifican características sobre ese estado. Dependiendo del valor de las variables de estado, el valor de una variable que caracteriza el estado, puede ser o no necesario.

- *Propiedades de los descriptores.* Descriptores que especifican relaciones entre objetos pueden poseer ciertas propiedades generales: simetría, transitividad, reflexividad, etc. Este tipo de información se puede incorporar en forma de regla para la obtención de nuevos hechos.

- *Interrelación entre descriptores.* En algunos problemas pueden existir relaciones entre descriptores que restringen sus valores.

En [114] se ofrecen normas generales para la incorporación de otra información como conocimiento subyacente de un problema.

1.2 Adquisición de conocimiento difuso

Uno de los objetivos principales que se plantean en este trabajo, es el desarrollo de un algoritmo que sea capaz de expresar el comportamiento de un sistema en términos difusos. Fundamentalmente, dos razones motivan este objetivo: la primera de ellas es que los modelos difusos han tenido un exito evidente en la modelización de sistemas. Tanto es así, que hoy día podemos ver la aparición de productos que incluyen en su diseño este tipo de modelos. La segunda razón, es que a menudo, la forma en que los humanos transmiten su conocimiento adopta una forma muy parecida a la forma en la cual representa su conocimiento los sistemas difusos. Por consiguiente, la representación mediante difusos de la información de un sistema aumenta la compresibilidad del mismo desde el punto de vista humano.

Por ejemplo, términos como *grande, alto, pequeño* y *mediano* son corrientes en una conversación para describir el tamaño de un objeto. Estos términos imprecisos son conocidos como *etiquetas lingüísticas* si están asociados a una semántica que los dota de significado. Las variables que toman estos términos son conocidas como *variables lingüísticas*. Por tanto, es interesante para muchos problemas la utilización de descripciones expresadas en términos de variables lingüísticas, sobre todo si el fin último del aprendizaje es que la información extraída sea fácilmente interpretable por un ser humano.

Formalmente, la definición de variable lingüística fué propuesta por Zadeh en [172], apoyándose en las propiedades de la teoría de subconjuntos difusos [171].

La teoría de subconjuntos difusos se basa en la relajación del concepto de pertenencia de un elemento a un conjunto. En la teoría tradicional, un elemento pertenece o no a un conjunto, mientras que en la teoría de subconjuntos difusos, un elemento pertenece a un conjunto en un cierto grado.

Una forma de representar las descripciones en términos lingüísticos lo constituyen las reglas difusas. Una regla difusa consta de un parte antecedente, descripción o condición y una parte consecuente o de clasificación. La estructura básica de un regla difusa es la siguiente:

$$\text{IF } X_1 \text{ is } A_1 \text{ and } \ldots \text{ and } X_n \text{ is } A_n \text{ THEN } Y \text{ is } B$$

donde cada variable X_i tiene un conjunto referencial U_i y toma valores en un dominio difuso D_i para $i \in \{1, \ldots, n\}$. El conjunto referencial para Y es V y su dominio es F. El valor de la variable Y es B, donde $B \in F$ y el valor de la variable X_i es A_i tal que $A_i \in D_i$.

Las reglas difusas, por tanto, se presentan desde el punto de vista humano, como una herramiento útil y confortable para la incorporación de conocimiento en modelos computacionales. Por consiguiente, como modelos válidos para la construcción de sistemas expertos.

Frente a la estructura básica de regla propuesta anteriormente, en la literatura podemos encontrar otras propuestas alternativas tal como el modelo de regla TSK [23] que se aplicado fundamentalmente al control de procesos.

En este trabajo, se propondrá un modelo de regla que generaliza el modelo básico propuesto anteriormente, que resulta ser más útil que otros modelos por dos razones: el número de reglas necesarias para expresar el comportamiento de un sistema es menor, y aumenta significativamente la comprensibilidad del conocimiento extraído.

La estructura de regla que aquí se propone, es la extensión a la utilización de etiquetas lingüísticas de un modelo de regla ampliamente utilizado en diferentes problemas de clasificación conocido como modelo DNF modificado. Además, en el capítulo 5 de esta memoria, generalizamos aún más este modelo de regla para incluir modificadores semánticos. Esta extensión esta basada en la definición de variable lingü'istica booleana propuesto en [172]. La inclusión de los modificadores semánticos dota de un mayor grado de flexibilidad al proceso de aprendizaje, y por tanto de una mejor aproximación entre el comportamiento del sistema y el conocimiento obtenido.

En el desarrollo de esta memoria, se propondrán dos modelos de razonamiento [34, 55] asociados a este modelo de regla que se adaptan a los dos diferentes tipos problemas de clasificación, que se pueden presentar en función del tipo de dominio asociado a la variable de clasificación. En este sentido, distinguimos problemas cuya variable de clasificación tiene asociado un referencial nominal o discreto, que denominaremos caso crisp, y problemas cuya variable de clasificación tiene un referencial continuo que discretizaremos utilizando conjuntos difusos. En el primer caso, a diferencia del segundo, el dominio asociado a la variable de clasificación toma valores no difusos. Este hecho, obligará a modificar la conducta de los algoritmos de aprendizaje en función del tipo de dominio asociado a la variable de clasificación. La justificación de este cambio de conducta se expondrá detenidamente en el siguiente capítulo de esta memoria.

1.3 Descripción de algunos algoritmos de aprendizaje

De entre los distintos tipos de aprendizaje inductivo existentes en la bibliografía, destacan por su importancia dos grupos de algoritmos: aquellos que utilizan árboles de decisión para representar el conocimiento adquirido y que normalmente utilizan una estrategia de divide y vencerás descendente y aquellos que utilizan una estrategia ascendente basada en operaciones de generalización y especificación. A continuación describimos las características más relevantes de estos grupos.

1.3.1 Basados en árboles de decisión

Dentro de los algoritmos que usan árboles de decisión como forma de representación del conocimiento adquirido, podemos considerar las tres siguientes familias:

- La familia ID3, cuyos máximos exponentes son el propio programa ID3 desarrollado por Quinlan [125] y el programa CLS de Hunt [81] que se puede considerar como el preculsor de esta filosofía de aprendizaje.
- La familia de árboles de regresión, cuyo máximo exponente es CART, desarrollado por Breiman et al. [13].
- La familia compuesta por otros métodos que utilizan criterios estadísticos ([140]).

Todos estos sistemas se caracterizan por emplear una estrategia de divide y vencerás en forma descendente que particiona un conjunto de ejemplos dado, en subconjuntos cada vez más pequeños a medida que el árbol crece. La diferencia fundamental entre las tres familias es el criterio de selección utilizado para subdividir el conjunto de ejemplos en cada paso del algoritmo.

Estos algoritmos de aprendizaje utilizan criterios de selección de características para elegir el atributo por el cual dividir el conjunto de ejemplos. El conjunto de ejemplos se divide progresivamente usando los atributos seleccionados hasta que los ejemplos de cada subconjunto generado pertenecen a una única clase. A los subconjuntos que verifican esta condición se les conoce como *hojas* del árbol, mientras aquellos que no lo verifican se les conoce como *nodos*. Cada hoja del árbol está asociada a un concepto, que corresponde a la clase de los ejemplos que pertenecen a dicho subconjunto.

De las tres familias anteriores, detallaremos a continuación la primera que es la más referenciada en el campo de la inteligencia artificial y sobre la que actualmente hay una gran cantidad de investigadores trabajando sobre esta filosofía de aprendizaje. Para más información sobre la segunda y tercera familia ver [140].

El algoritmo ID3 (Induction of Decision Trees) de Quinlan [125, 126] es el descendiente más fiel del originario algoritmo CLS (Concept Learning System) de Hunt [81] y el más conocido de los algoritmos de aprendizaje que usan árboles de decisión.

ID3 construye un árbol de decisión de forma descendente (es decir, partiendo de los descriptores hacia los ejemplos) siendo guiado por una medida de información que se establece sobre los ejemplos. La estructura básica del algoritmo es iterativa. El objetivo consiste en construir un árbol que sea capaz de clasificar correctamente todos los ejemplos del conjunto de entrenamiento. Para ello, selecciona un atributo y en base a él divide el conjunto de ejemplos. Si este atributo es suficiente para clasificar todos los ejemplos (es decir, todos los subconjuntos generados por este atributo son hojas) el proceso termina. En otro caso, se selecciona otro atributo y se repite el proceso.

La selección del siguiente atributo por el cual se debe seguir dividiendo el conjunto de ejemplos es la componente fundamental de este algoritmo. Con este fin, se define un criterio de selección basado en una medida de información. Este criterio nos dice la ganancia de información obtenida si se selecciona un determinado atributo. El atributo más relevante será aquel que mayor información proporcione.

Las ventajas principales de este algoritmo son su simplicidad y su bajo costo computacional. Por contra, este algoritmo presenta también algunos inconvenientes importantes como:

- Tiene grandes problemas para aprender coherentemente ante la presencia de ruido, es decir, es muy vulnerable al ruido.
- Tampoco trabaja correctamente cuando aparecen valores desconocidos en los ejemplos.
- Tiene poca capacidad de generalización, lo que ocasiona la aparición de las llamadas hojas vacías (*null leaf*). Las hojas vacías son hojas que representan a subconjuntos vacíos de ejemplos, y por tanto, no tienen ninguna clase asociada. Esto provoca que no se puedan clasificar ejemplos que pertenecen a estos subconjuntos de ejemplos, por no aparecer en el conjunto de entrenamiento.
- El hecho de no soportar bien el ruido y la inconsistencia de algunos datos en el conjunto de ejemplos provoca la aparición de las llamadas hojas muertas (*dead-end leaf*). Este tipo de hojas aparece cuando se han explorado todos los atributos y alguno de los subconjuntos finales contiene ejemplos de más de una clase.
- La necesidad de precisión total del algoritmo obliga a que todos los ejemplos tienen

que estar correctamente clasificados, lo que puede provocar árboles de decisión demasiado grandes que complican la comprensión de los mismos.

- El lenguaje utilizado para el aprendizaje es demasiado pobre, y no contempla las posibles relaciones existentes entre distintos atributos.
- Es incapaz de trabajar con atributos continuo-valuados. Sólo trabaja con atributos o variables cuyo dominio es nominal.

El algoritmo ID3 supuso un punto de partida válido e interesante para el aprendizaje automático de conceptos y en base a él han aparecido en la literatura nuevas propuestas que intentan corregir sus deficiencias. Entre estas nuevas propuestas, podemos destacar el sistema de aprendizaje ACLS (Analogue Concept Learning System) que es una generalización de ID3 que admite como valores para los atributos un número ilimitado de valores enteros y en una filosofía semejante de generalización [18, 154]. El sistema ASSISTANT [17] es el primer descendiente de ID3 que trabaja con atributos continuo-valuados. Otra cualidad que introduce este sistema es la posibilidad de determinar y construir clases jerárquicas, es decir, que una clase puede ser una división más fina de una clase superior.

La teoría de subconjuntos difusos se ha utilizado como método para resolver el problema que supone trabajar con atributos continuo-valuados para la construcción de árboles de decisión. Los sistemas Fuzzy-ID3 [160] y FIDMV [86] constituyen dos exponentes de esta forma de resolver el problema, donde los atributos continuos son discretizados usando variables lingüísticas. Esta vía para dar respuesta al problema de los atributos continuo-valuados ha motivado un creciente interés de los investigadores de este campo que ha ocasionado la aparición de multitud de mecanismos para mejorar el comportamiento de estos sistemas. Entre estos trabajos, destacamos los métodos de poda definidos para este tipo de árboles que se proponen en [6].

Para resolver algunos problemas de ID3, el propio Quinlan propone el sistema C4 [127] y más tarde el C4.5 [128], donde además de incorporar métodos para trabajar con variables continuas, introduce criterios de poda para aumentar tanto la comprensibilidad del árbol como su precisión. Este sistema lo usaremos en la parte experimental para comparar su comportamiento con el algoritmo que se desarrollará a lo largo de esta memoria.

Otras propuestas destacables son los sistemas ID4 e ID5 [148] que transforman a ID3 en un sistema de aprendizaje incremental, así como el trabajo de Uthurusamy [154] que introduce mecanismos para eliminar la sensibilidad al ruido de los árboles de decisión.

1.3.2 Basados en la metodología STAR

Frente a los sistemas que utilizan una estrategia divide y vencerás descendente donde el conocimiento se representa mediante árboles de decisión, están los sistemas de aprendizaje que utilizan la estrategia de generalización y especificación de forma ascendente. La mayoría de estos sistemas representan el conocimiento mediante reglas de producción. Uno de los máximos exponentes de este tipo de sistemas es AQ11 [109] diseñado por Michalski y que siguen la filosofía STAR. Los antecedentes de esta metodología se encuentran en los trabajos de Winston [167] con su programa ARCH y el algoritmo de eliminación de candidatos propuesto por Mitchell [116].

El programa ARCH describe un mecanismo para aprender conceptos basado en la búsqueda de relaciones entre las representaciones mediante redes semánticas en el mundo de bloques. En este programa son importantes dos procesos:

(i) encontrar y explorar las semejanzas existentes entre las descripciones estructurales para un mismo tipo de configuración,

(ii) encontrar diferencias significativas entre los ejemplos positivos y los ejemplos negativos.

Estos dos procesos corresponden en la terminología anteriormente descrita en este capítulo a la generalización y a la especificación. El programa ARCH generaliza las descripciones para cubrir todos los ejemplos positivos y las especifica para eliminar todos los ejemplos negativos. Si al sistema se le presenta un nuevo ejemplo, si este es positivo se generalizan las descripciones para cubrirlo, si por contra es negativo, se especifican las descripciones existentes.

El algoritmo de eliminación de candidatos de Mitchell tiene una estrategia semejante a la propuesta por Winston en el sentido que ambos usan procesos de generalización y especificación, pero se diferencian en la forma de explorar el espacio de soluciones. Mientras en el programa ARCH las hipótesis son generadas y verificadas una a una, en el método de Mitchell las hipótesis son sistemáticamente eliminadas del espacio de todas las posibles hipótesis cuando estas no son satisfactorias. Lo novedoso de este método es el uso de un sistema muy eficiente para representar el conjunto de todas las hipótesis como un espacio de versiones.

Frente a estos dos sistemas de aprendizaje iniciales, AQ11 propone una variante más sofisticada del uso de la generalización y la especificación, donde la principal ventaja de AQ11 y de todos los algoritmos que utilizan la metodología STAR es el hecho de trabajar con un espacio de versiones que aseguran una correcta representación de las descripciones y un mecanismo muy apropiado para el uso de estos procesos de generalización. Además, con esta representación se construyen descripciones tan generales como permitan los ejemplos negativos.

El algoritmo AQ11 presenta también algunos problemas:

- El hecho de utilizar el algoritmo de eliminación de candidatos conlleva arrastrar todos los problemas de este algoritmo. Un problema típico de este algoritmo es que las entradas para un concepto no estén en zonas adyacentes del espacio de soluciones.

- El conjunto de reglas que AQ11 obtiene depende del orden en el cual se le presentan los ejemplos del conjunto de entrenamiento al sistema. En general, la primera regla inducida será más general que las que se obtengan posteriormente.
- El costo computacional de este algoritmo es mucho mayor que el necesario en sistemas que utilizan árboles de decisión tales como ID3, aunque por contra, es mucho mejor la comprensibilidad de los resultados.
- Es un algoritmo muy sensible al ruido.

Así como ID3 se puede considerar como el punto de partida para el desarrollo de nuevos sistemas de aprendizaje que usan árboles de decisión, a la familia de algoritmos AQ los podemos considerar en el mismo sentido para estrategias de aprendizaje ascendentes.

De los anteriores problemas de AQ11, el más importante es su alta sensibilidad al ruido, para el que muchos investigadores han propuesto soluciones. El propio Michalski en [115] propone una primera solución, introduciendo el uso de procedimiento de adaptación parcial a la hora de ejecutar las reglas. Una segunda aproximación es propuesta por Clark con su algoritmo CN2 [19] que introduce un algoritmo descendente que produce las mismas reglas (del tipo If-then-else) pero controlando el nivel de ruido.

Otras mejoras sustanciales de este sistema pueden verse en las siguientes generaciones de la familia AQ, como son AQ15, AQ15-GA y AQ17. En AQ15 [115] Michalski introduce una nueva herramienta muy poderosa y flexible que permite la construcción de nuevos descriptores en base a los descriptores iniciales [114]. Esto supone que AQ entra en el mundo de la relevancia indirecta de los descriptores iniciales. Este nuevo sistema demuestra ser mejor que AQ11 en cuanto a la comprensibilidad de las descripciones obtenidas y en cuanto al porcentaje de acierto. AQ15-GA es una implementación basada en el uso de algoritmos genéticos sobre AQ15 propuesta por Thrun [148].

En los últimos años han aparecido nuevas versiones más poderosas de la familia AQ tales como:

- AQ17-DCI [148], donde además de la generación de nuevos descriptores se define una *función de calidad* para los atributos que sobrepasan una determinada cota prefijada para determinar la relevancia de cada uno de ellos en las descripciones.
- AQ17-FCLS (Flexible Concept Learning), [148] que es capaz de trabajar con representaciones numéricas y simbólicas para describir los conceptos y además las reglas se aprenden en dos fases: una primera fase semejante a la metodología STAR que permite una preclasificación de los ejemplos, y una segunda fase de especificación de las reglas para mejorar su porcentaje de aciertos.
- AQ17-HCI (Hypothesis-driven Constructive Induction) [148], donde se generan nuevos atributos en base a las descripciones obtenidas en iteraciones anteriores.

1.4 La aplicación de la computación evolutiva al aprendizaje automático

Los organismos vivos poseen una consumada destreza en la resolución de problemas, esta habilidad la obtienen a través de la *evolución.* En casi todos los organismos, la evolución se produce mediante dos procesos elementales: 1) la *selección natural*, que determina qué miembros de la población sobrevivirán hasta reproducirse y 2) la *reproducción*, que garantiza la mezcla y recombinación de sus genes entre la descendencia. El efecto conjunto de estos dos procesos marca el éxito de 3 billones de años de vida sobre el planeta: cualquier organismo, en sucesivas generaciones, es capaz de *adaptarse* a cambios en su forma de vida, e incluso en su estructura, para combatir cualquier agente que hiciera peligrar su supervivencia.

A través de la historia, la naturaleza ha sido una fuente inagotable de inspiración para el desarrollo técnico y científico, el cual, a cambio, ha contribuido a una mejor comprensión de la propia naturaleza. J. Holland estudia el fenómeno de la adaptación tal y como ocurre en la naturaleza y desarrolla mecanismos para incorporarla en un sistema computerizado. En 1975, en su libro *Adaptación en sistemas naturales y artificiales* ([72]) propone una clase de algoritmos adaptativos, ideados como una abstracción de la evolución para resolver problemas prácticos y para servir de modelos computacionales de sistemas naturales evolutivos, algoritmos que posteriormente recibieron el nombre de *algoritmos genéticos.*

En la naturaleza, *los procesos evolutivos* ocurren cuando se satisfacen las siguientes condiciones:

- Una entidad o individuo tiene la habilidad de reproducirse.
- Hay una población de tales individuos que son capaces de reproducirse.
- Existe alguna variedad, diferencia, entre los individuos que se reproducen.
- Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad.

Esta diversidad se manifiesta en cambios en los cromosomas de los individuos de la población, y se traslada a la variación de la estructura y el comportamiento de los individuos en su entorno. Estos cambios en la estructura y comportamiento se reflejan en el grado de supervivencia, adaptación, y en el nivel de reproducción. Los individuos que mejor se adaptan a su entorno son los que más sobreviven y más se reproducen. En un periodo de tiempo y con muchas generaciones de individuos, la población llega a contener más individuos cuyos cromosomas se trasladan a estructuras y a comportamientos adecuados a su entorno, sobreviviendo y reproduciéndose en un alto índice, de forma que con el tiempo la estructura de individuos en la población cambia debido a la selección natural ([24]).

J. Holland fue uno de los pioneros en la aplicación de los principios en los que se basan los procesos de la evolución natural como reglas en el diseño de sistemas artificiales para la resolución de problemas. En [72], Holland proporciona un marco general en el que se muestra cómo aplicar los procesos evolutivos a los sistemas artificiales diseñados a partir de los sistemas naturales. Cualquier problema de adaptación puede formularse generalmente en términos genéticos. Una vez formulados en estos términos, tales problemas pueden resolverse mediante lo que hoy se conoce como *algoritmos genéticos* (AGs).

Paralelamente a los estudios realizados por Holland, la idea de aplicar los principios en los que se basan los procesos de la evolución natural como reglas en los procedimientos de búsqueda fue investigada con otros enfoques ([41]). Actualmente, todas estas aportaciones iniciales han cristalizado en lo que se denomina *computación evolutiva*, y recoge las distintas vertientes en el diseño de los *algoritmos de evolución* como herramientas para el análisis de sistemas no lineales complejos por medio de simulación computacional y basados en los principios en los que se basan los procesos de evolución natural. La idea general de estos algoritmos es manejar, durante cierto tiempo, una población de soluciones candidatas para un problema particular, a través de operadores inspirados en los mecanismos naturales que producen variación genética y en la selección natural. Las diferentes variantes de algoritmos de evolución existentes actualmente son:

- *Algoritmos genéticos* ([48, 72]).
- *Estrategias de evolución* ([138]).
- *Programación evolutiva* ([40, 41]).
- *Programación genética* ([91, 92]).

En los años sesenta y setenta, se introducen las estrategias de evolución como métodos para optimizar parámetros de dispositivos aerodinámicos. El modelo era simple; a partir de un vector de parámetros reales, solución factible para un problema dado, se generaba otro vector mediante cambios aleatorios sobre el primero, y se seleccionaba el mejor de ellos para reiterar de nuevo el proceso. Posteriormente, Schwefel desarrolló esta idea con más profundidad ([137, 138]). El estudio de las estrategias de evolución ha sido un área de investigación muy activa, independiente del área de los AGs (aunque recientemente se han comenzado los intercambios entre ambos campos ([4, 5])).

Fogel, Owens y Walsh desarrollaron en los años sesenta la programación evolutiva, una técnica en la que las soluciones candidatas para un problema dado se representan como máquinas de estados finitos, las cuales evolucionan mediante las mutaciones aleatorias producidas sobre sus diagramas estado-transición y la selección de los mejores ([40, 41]).

Koza propone en 1991 ([91]) un modelo de algoritmo de evolución específico para obtener programas de ordenador que resuelven problemas dados. Para un problema concreto, un algoritmo de este tipo maneja un conjunto de árboles que representan expresiones en lenguaje lisp que resuelven el problema a cierto nivel. La aplicación de operadores especializados y de un mecanismo de selección produce árboles de expresiones que resuelven el problema concreto cada vez con mayor precisión. Tras el trabajo de Koza, se han presentado bastantes modelos basados en la misma idea pero que usan otros lenguajes de programación, tales como el C y C++. Esto ha dado lugar a la creación de un área dentro de la computación evolutiva con nombre propio: la programación genética ([92]).

En los últimos años, se ha dado una amplia interacción entre los investigadores de las distintas vertientes de la computación evolutiva. En muchos casos, esto ha permitido romper las fronteras entre los AGs, las estrategias de evolución, la programación evolutiva y la programación genética.

1.4.1 Algoritmos genéticos

Los AGs ([48, 72]) son procedimientos adaptativos para la búsqueda de soluciones en espacios complejos, inspirados en los procesos genéticos de los organismos naturales y en los principios de la evolución natural de poblaciones. Su idea básica es mantener una *población* de *cromosomas*, los cuales representan soluciones candidatas a un problema concreto, que evolucionan con el tiempo a través de un proceso de competición y variación controlada. Cada cromosoma tiene una bondad o adaptación asociada, que describe la adecuación de la solución a la que representa. El proceso de competición, denominado *mecanismo de selección*, utiliza estas adaptaciones para determinar los cromosomas que se usan para crear otros nuevos. Los nuevos cromosomas se generan a través de los operadores genéticos denominados *cruce* y *mutación*.

Los AGs se han aplicado con éxito en problemas de búsqueda y optimización. Gran parte de este éxito se debe a su capacidad para explotar la información acumulada sobre un espacio de búsqueda, y así dirigir las siguientes búsquedas hacia los mejores subespacios. Esta es su principal ventaja, sobre todo en espacios grandes, complejos y parcialmente definidos donde las técnicas clásicas de búsqueda no son apropiadas. Los AGs ofrecen una aproximación válida a problemas que requieren técnicas de búsqueda efectivas y eficientes.

Bajo la formulación inicial de los AGs, las soluciones se han representado mediante cadenas binarias de longitud fija, debido principalmente a que existen razones teóricas que recomiendan el uso de esta representación ([50]) y además, porque ésta permite un diseño simple de los operadores genéticos. Sin embargo, las buenas propiedades de los AGs no radican en el uso de cadenas de ceros y unos ([3, 130]). Por esta razón, frecuentemente se han utilizado representaciones no binarias más adecuadas para cada uno de los problemas particulares de aplicación. Una de las representaciones alternativas más importantes es la *representación real*. Esta representación parece particularmente natural cuando se tratan problemas con variables continuas. El uso de nuevas representaciones y la construcción de nuevos operadores para manipularlas ha propiciado que la concepción actual de AG sea bastante diferente y más general que la idea original de Holland ([72]).

A continuación ofrecemos una descripción más detallada de los AGs, considerando los siguientes aspectos: la estructura de un AG, la representación, el mecanismo de selección y los operadores genéticos de cruce y mutación.

1.4.1.1 Estructura de un AG

Un AG comienza con una población de cromosomas generados aleatoriamente y va obteniendo mejores cromosomas gracias a la aplicación de los operadores genéticos, los cuales están basados en los procesos genéticos que ocurren en la naturaleza. La evolución se produce sobre la población en forma de selección natural. Durante sucesivas iteraciones, llamadas *generaciones*, se evalúa la adecuación (adaptación) de los cromosomas como soluciones, y en base a esta evaluación, se forma una nueva población de cromosomas usando un mecanismo de selección y operadores genéticos específicos, tales como el cruce y la mutación. Para cada problema a resolver, se ha de proporcionar una *función de evaluación o adaptación*, f. Dado un cromosoma, la función de evaluación devuelve un valor numérico que se supone proporcional a la utilidad o adaptación de la solución que representa.

Aunque existen muchas posibles variantes de AGs, los mecanismos fundamentales bajo los cuales funcionan consisten en operar sobre una población de individuos, que inicialmente se genera de forma aleatoria, y cambiar en cada generación la población atendiendo a los tres pasos siguientes:

1. Evaluación de los individuos de la población.
2. Formación de una población intermedia a través del mecanismo de selección.
3. Formación de una nueva población a partir de los operadores genéticos de cruce y mutación.

Estos tres pasos se repiten hasta que el sistema deja de mejorar, o se alcanza un número máximo, T, de generaciones especificado por el usuario. El esquema 1.1 muestra la estructura de un AG.

Debido a la naturaleza aleatoria de los AGs, normalmente todo el proceso se repite más de una vez. Los investigadores de los AGs suelen obtener estadísticas tales como el mejor elemento encontrado durante una ejecución, la media de la función de evaluación de todos los cromosomas muestreados, generación en la que se encontró el mejor elemento, etc, todas ellas, promediadas sobre el número total de ejecuciones.

Esquema 1.1 *Estructura de un algoritmo genético*

Procedimiento Algoritmo Genético
Inicio
$t = 0$;
Inicializar $P(t)$;
Evaluar $P(t)$;
Mientras *(No se cumpla la condición de parada)* **Hacer**
Inicio
$t = t + 1$;
Seleccionar $P(t)$ desde $P(t-1)$;
Aplicar Cruce y Mutación sobre $P(t)$;
Evaluar $P(t)$;
Final
Final

1.4.1.2 Representación

La representación es uno de los elementos clave en la operatividad de un AG, debido a dos razones: 1) los AGs manipulan una representación codificada del problema y 2) la representación puede limitar en gran medida la visión que tiene un sistema sobre su entorno ([92]).

Con frecuencia, las soluciones se representan usando cadenas binarias. Para representar un espacio de búsqueda $S = S_1 \times .... \times S_n$ por medio del alfabeto binario se usa un conjunto de funciones $cod_i : S_i \rightarrow \{0,1\}^{L_i}$ con $L_i \in N$ y $i = 1,...,n$, tal que cod_i codifica cada elemento de S_i usando una cadena de ceros y unos. Un elemento $x = (x_1,...,x_n) \in S$ ($x_i \in S_i$) queda representado a través de la concatenación de la codificación de cada una de sus componentes: $cod(x) = cod_1(x_1)...cod_n(x_n)$. Los AGs basados en este tipo de representación serán denominados AGs con codificación binaria (AGCBs)

En muchos casos, se han aplicado representaciones no binarias más naturales para los problemas particulares de aplicación, tales como en los trabajos [28, 69, 82, 101].

1.4.1.3 Mecanismo de Selección

Consideremos un población P con los cromosomas $C_1,...,C_N$. El mecanismo de selección produce una población intermedia, P', con copias de cromosomas en P. El número de copias recibidas por cada cromosoma depende de su función de evaluación; aquellos cromosomas con función de evaluación alta tienen usualmente mayor probabilidad de contribuir con copias para P'. El mecanismo de selección se realiza siguiendo dos pasos:

1. Para cada cromosoma C_i en P se calcula la probabilidad, $p_s(C_i)$, de incluir una copia suya en P'. El modelo más utilizado para calcular esta probabilidad es el modelo *proporcional* ([48, 72]), donde $p_s(C_i)$, $i = 1, ..., N$ se calcula como

$$p_s(C_i) = \frac{f(C_i)}{\sum_{j=1}^{N} f(C_j)}.$$

   De esta forma, los cromosomas con función de evaluación por encima de la media reciben más copias que aquéllos con función de evaluación por debajo de la media.

2. Basándose en las probabilidades de selección calculadas en el paso anterior, se asigna, aleatoriamente, a cada elemento de P el número de copias suyas que aparecerán en P'. Este paso se realiza a través de un método de *muestreo*. El más simple, llamado *muestreo aleatorio simple* ([48, 72]), consiste en simular el comportamiento de una ruleta en la que existe para cada elemento de la población una región proporcional al valor de su probabilidad de selección. Cada vez que se juega, la ruleta determinará un elemento para la población intermedia. Jugando N veces, la población intermedia queda totalmente determinada. Los elementos con mayor probabilidad de selección tendrán, en la ruleta, una mayor proporción de espacio asignado, de esta forma la probabilidad de que la ruleta se detenga en las zonas asignadas a estos elementos es grande.

La figura 1.3 muestra un ejemplo de la aplicación del mecanismo de selección, mostrándose a la izquierda la ruleta asociada.

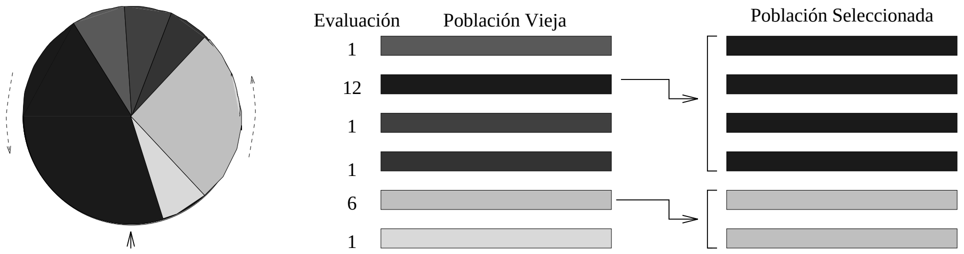


Figura 1.3: Mecanismo de selección

Una estrategia adicional, que se utiliza frecuentemente como complemento del mecanismo de selección, es el *elitismo* ([29]), el cual asegura que el mejor elemento de una generación t estará presente en la generación $t + 1$. Esto es conveniente ya que es posible que el mejor elemento de la generación t desaparezca debido a la aplicación del propio mecanismo de selección o por los operadores de cruce y mutación.

1.4.1.4 Operadores Genéticos

Tras llevar a cabo el mecanismo de selección, se aplican los operadores de cruce y mutación sobre los elementos de la población intermedia.

Operador de Cruce

El operador de cruce es un método para compartir información entre los cromosomas. Generalmente se combinan las características (genes) de dos cromosomas padres para generar dos cromosomas hijos. El operador de cruce juega un papel central en los AGs. De hecho puede considerarse como una de las características que definen a estos algoritmos y es uno de los componentes a tener en cuenta para mejorar su eficacia ([99]). A veces, al operador de cruce se le denomina operador de *recombinación*.

Las definiciones para los operadores de cruce (y mutación) dependen de la representación particular escogida. Para el caso particular de los AGCBs, el operador clásico de cruce es el *cruce simple* ([48, 72]), en el que dados dos cromosomas padres, $C_1 = (c_1^1 \ldots c_L^1)$ y $C_2 = (c_1^2 \ldots c_L^2)$, se generan los dos cromosomas hijos, $H_1 = (c_1^1, .., c_i^1, c_{i+1}^2, ..., c_L^2)$ y $H_2 = (c_1^2, .., c_i^2, c_{i+1}^1, ..., c_L^1)$, donde i es un número aleatorio perteneciente a $\{1, ..., L-1\}$. La figura 1.4 muestra un ejemplo de la aplicación de este operador.

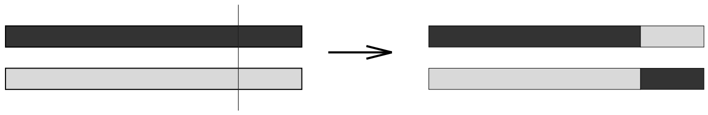


Figura 1.4: Cruce simple

Un parámetro de control del AG, denominado *probabilidad de cruce*, p_c, determina la probabilidad de aplicar el operador de cruce sobre parejas de cromosomas de la población intermedia.

Operador de Mutación

El operador de mutación altera arbitrariamente uno o más componentes (genes) de un cromosoma para aumentar la variabilidad estructural de la población. El papel de la mutación en los AGs es restaurar material genético perdido o no explorado en la población. De esta forma, se asegura que la probabilidad de alcanzar cualquier punto del espacio de búsqueda nunca es cero. Cada gen en cada cromosoma sufre una mutación de acuerdo con un parámetro de control, denominado probabilidad de mutación, p_m.

En los AGCBs, la mutación de un gen es fácil: se cambia su valor, "1" por "0" y viceversa. La figura 1.5 muestra un ejemplo de la aplicación del operador de mutación.

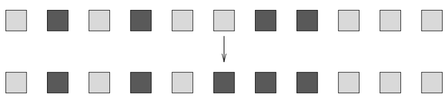


Figura 1.5: Mutación

1.4.2 Algoritmos genéticos con codificación real

Los AGCBs tienen ciertas dificultades cuando se aplican sobre espacios de búsqueda continuos con grandes dimensiones y donde se requiere una gran precisión numérica. También se plantea un problema cuando las variables que se manejan toman un número finito de valores discretos válidos, el cual no es potencia de 2. En este caso, algunos códigos binarios serán redundantes.

Una de las alternativas más importantes a la codificación binaria es la codificación real. Este tipo de codificación parece particularmente natural para resolver problemas con variables en espacios de búsqueda continuos. Un cromosoma es un vector de números reales cuyo tamaño es igual a la longitud del vector solución del problema; de esta forma, *cada gen representa una variable del problema.* Los valores de los genes están forzados a permanecer en el intervalo establecido para las variables a las cuales representan, de modo que los operadores genéticos tendrán que preservar este requerimiento.

La utilización de la codificación real aparece inicialmente en aplicaciones particulares, tales como en [101] para problemas de quimiometría. Posteriormente, los AGCRs se han usado principalmente en problemas de optimización numérica sobre dominios continuos ([28, 69]). Hasta 1991 no se realizó ningún estudio teórico sobre estos algoritmos; su utilización provocaba cierta controversia: los investigadores familiarizados con la teoría fundamental de los AGs no comprendían los buenos resultados empíricos mostrados por los AGCRs, ya que esta teoría sugería que los alfabetos de cardinal bajo debían ser más efectivos que aquellos con cardinales altos. Más tarde, se presentaron distintas herramientas para el tratamiento teórico de los AGCRs ([50, 129, 170]), lo que permitió corroborar su buen comportamiento.

El uso de parámetros reales posibilita manejar grandes dominios (incluso dominios desconocidos) para las variables, tarea difícil en las implementaciones binarias donde extender el dominio significaría sacrificar precisión, supuesta una longitud fija para los cromosomas. Otra ventaja es la capacidad para explotar la *gradualidad* de las funciones con variables continuas, donde el concepto gradualidad se refiere a que pequeños cambios en las variables se corresponden con pequeños cambios en la función. En esta línea, se destaca la habilidad de los AGCRs para *el ajuste local* de las soluciones, existiendo operadores de mutación (*mutación no uniforme* ([106])) que permiten que ésta se realice de forma más rápida y eficaz que en los AGCB.

Usando codificación real, la representación de las soluciones está muy cercana a la formulación natural de muchos problemas. Así se evita el proceso de codificación y decodificación binaria que se debe producir en los AGCBs, de manera que se aumenta la rapidez del AG. En [130], Radcliffe aduce que no es necesario que existan diferencias entre el genotipo y el fenotipo para que se produzca evolución. Por ello, no se justifica que los operadores genéticos deban definirse en razón a la representación escogida. Muy al contrario, el autor aboga por definir tanto los operadores genéticos como los conceptos de esquema directamente sobre el espacio de los fenotipos siempre y cuando esto sea posible. Los operadores genéticos y los conceptos de esquema presentados para los AGCRs en la literatura concuerdan con la idea de Radcliffe.

1.4.2.1 Operadores de Cruce

Supongamos que los cromosomas $C_1 = (c_1^1, ..., c_n^1)$ y $C_2 = (c_1^2, ..., c_n^2)$ se han seleccionado para aplicarles el operador de cruce. A continuación, se muestran los efectos de diferentes operadores de cruce para AGCRs sobre estos cromosomas.

Hay que notar que cada operador produce un número distinto de hijos por lo que a veces se hace necesaria una selección para obtener los cromosomas que formarán parte de la población. Esta selección se denominará *selección de descendientes*.

- *Cruce plano* ([129])

  Se genera el hijo, $H = (h_1, ..., h_i, ..., h_n)$, donde h_i es un valor escogido de forma aleatoria (uniformemente) del intervalo $[c_i^1, c_i^2]$.

- *Cruce simple* ([106, 170])

  Aleatoriamente se elige una posición $i \in \{1, 2, ..., n-1\}$ y se generan

  $$H_1 = (c_1^1, c_2^1, ..., c_i^1, c_{i+1}^2, ..., c_n^2), \text{ y } H_2 = (c_1^2, c_2^2, ..., c_i^2, c_{i+1}^1, ..., c_n^1).$$

- *Cruce aritmético* ([106])

  Se generan dos hijos, $H_k = (h_1^k, ..., h_i^k, ..., h_n^k)$ $k = 1, 2$, donde $h_i^1 = \lambda c_i^1 + (1-\lambda)c_i^2$ y $h_i^2 = \lambda c_i^2 + (1-\lambda)c_i^1$. λ es una constante (cruce aritmético uniforme) o varía en función del número de generaciones realizadas (cruce aritmético no uniforme).

- *Cruce BLX-α* ([35])

  Se genera un hijo, $H = (h_1, ..., h_i, ..., h_n)$, donde h_i es un número escogido de forma aleatoria (uniformemente) del intervalo $[c_{min} - I\alpha, c_{max} + I\alpha]$, donde $c_{max} = \max(c_i^1, c_i^2)$, $c_{min} = \min(c_i^1, c_i^2)$ y $I = c_{max} - c_{min}$. El cruce BLX-0.0 es igual al cruce plano.

- *Cruce lineal* ([170])

  Se generan tres hijos, $H_k = (h_1^k, ..., h_i^k, ..., h_n^k)$ $k = 1, 2, 3$, donde $h_i^1 = \frac{1}{2}c_i^1 + \frac{1}{2}c_i^2$, $h_i^2 = \frac{3}{2}c_i^1 - \frac{1}{2}c_i^2$ y $h_i^3 = -\frac{1}{2}c_i^1 + \frac{3}{2}c_i^2$.

  Con este tipo de cruce se realiza una *selección de descendientes* consistente en que los dos cromosomas más prometedores son los seleccionados para formar parte de la nueva población.

1.4.2.2 Operadores de Mutación

Supongamos que $C = (c_1, ..., c_i, ..., c_n)$ es un cromosoma y c_i es un elemento a mutar, a continuación se muestra el gen $c_i^{'}$ resultante de aplicar los dos operadores de mutación más habitualmente utilizados para AGCRs son:

- *Mutación aleatoria* ([129])

  $c_i^{'}$ se toma de forma aleatoria (uniformemente) del dominio $[a_i, b_i]$.

- *Mutación no uniforme* ([106])

  Si se supone que este operador se aplica durante una generación t y que T es el máximo número de generaciones del AGCR, entonces

  $$c_i^{'} = \begin{cases} c_i + \triangle(t, b_i - c_i), & \text{Si } \beta = 0 \\ c_i - \triangle(t, c_i - a_i), & \text{Si } \beta = 1 \end{cases}$$

  donde β es un número aleatorio que puede valer cero o uno, y

  $$\triangle(t, y) = y(1 - r^{(1-\frac{t}{T})^b}),$$

  donde r es un número aleatorio del intervalo $[0, 1]$ y b es un parámetro elegido por el usuario, que determina el grado de dependencia con el número de iteraciones. La función $\triangle(\cdot, \cdot)$ devuelve un valor en el rango $[0, y]$ de forma que la probabilidad de devolver un número cercano a cero crece conforme t es mayor. De esta forma, el intervalo de generación de genes es más pequeño conforme pasan las generaciones. Esta propiedad permite que este operador produzca una búsqueda uniforme del espacio de búsqueda cuando t es pequeño, y una búsqueda local en las etapas finales, favoreciendo el ajuste local.

1.4.3 Modelos de aprendizaje basados en algoritmos genéticos

A partir de los años 80 surgió un creciente interés en aplicar los métodos basados en los AGs a problemas de aprendizaje automático, en particular al aprendizaje de reglas de producción en base a conjuntos de ejemplos atributo-valorados. El principal problema en tales aplicaciones consiste en encontrar una representación "confortable" en el sentido de que sea capaz tanto de recoger las características del problema como de representar las soluciones potenciales. Cuando usamos una representación para aprender conceptos mediante reglas tenemos un problema añadido, el número de tales reglas es desconocido a priori, por tanto, la representación tradicional con longitud fija no es recomendable. Tradicionalmente se han propuesto dos modelos de aprendizaje :

- **Modelo de Michigan**, en donde la población consta de cromosomas de longitud fija y la solución del problema se representa por todos los cromosomas de la población. Este modelo es usualmente presentado en la literatura con el nombre de *Sistemas Clasificadores* (SCs). Su base es un mecanismo especial de asignación de pesos mediante algoritmos como el *Bucket-Brigade* y fué desarrollado originariamente por Holland [77].

- **Modelo de Pittsburgh**, que constituye una extensión del modelo tradicional de cromosomas de longitud fija. Los cromosomas ahora tienen longitud variable y se utilizan para representar soluciones individuales, es decir, bases de reglas completas. Este modelo fué inicialmente propuesto por Smith [145].

La diferencia fundamental entre estos dos modelos, es que mientras el primero busca como solución a una población completa, el segundo sólo busca un único individuo de la población.

Las dos modelos presentan algunos inconvenientes debido principalmente a que implican una readaptación de los tradicionales AGs.

El modelo de Pittsburgh tiene dos inconvenientes fundamentales, el primero de ellos se debe a que los individuos en la población son de longitud variable, lo que ocasiona un tratamiento especial de dichos elementos y una redefinición de todos los operadores genéticos para adaptarse a estas circunstancias. La segunda está relacionada con el espacio de posibles soluciones. Supongamos que N es el número de reglas necesario para describir el problema, K es el número de clases, n es el número de atributos y V es el número de valores por atributo. El número de reglas posibles que constituyen el espacio de búsqueda viene definido por la siguiente fórmula [157]:

$$N^{K(V+1)^n}$$

es decir, el espacio de búsqueda es extremadamente grande, con lo que el tiempo requerido para encontrar una buena solución lo es también.

Frente al modelo de Pittsburgh, el modelo Michigan no necesita readaptar los operadores tradicionales de los AGs y además el número de posibles soluciones viene determinado por la siguientes expresión:

$$K(V+1)^n$$

que es sensiblemente menor que el del otro modelo, aunque aún sigue siendo un espacio de búsqueda bastante grande. Por otro lado, el hecho de que la solución al problema se represente por una subpoblación de individuos (que difiere de los tradicionales AGs que buscan el mejor individuo) provoca la introducción de una función de evaluación que debe tener en cuenta a todos los miembros de la población. Esto implica una función de evaluación algo más compleja, lo que ocasiona generalmente un costo computacional mayor frente a los AGs tradicionales.

Para la construcción de nuestro algoritmo exploraremos una vía alternativa de la aplicación de los algoritmos genéticos al problema del aprendizaje automático, que reduzca el espacio de las soluciones posibles.

En las siguientes secciones pasaremos a estudiar algunos algoritmos que usan estos modelos.

1.4.4 Sistemas clasificadores (SCs)

Las primeras ideas desarrolladas sobre los SCs fueron presentadas por Holland en [72, 73], y fué este mismo autor junto con Reitman en [74] quien desarrolló el primer SC. A partir de este momento se han propuesto en la literatura muchos tipos diferentes de SCs [12, 16, 48, 121, 155, 165, 166].

Los SCs se definen como sistemas paralelos basados en reglas, de paso de mensajes, que son capaces de interactuar con el entorno y de aprender mediante refuerzo (asignando recompensas) y descubrimiento de reglas.

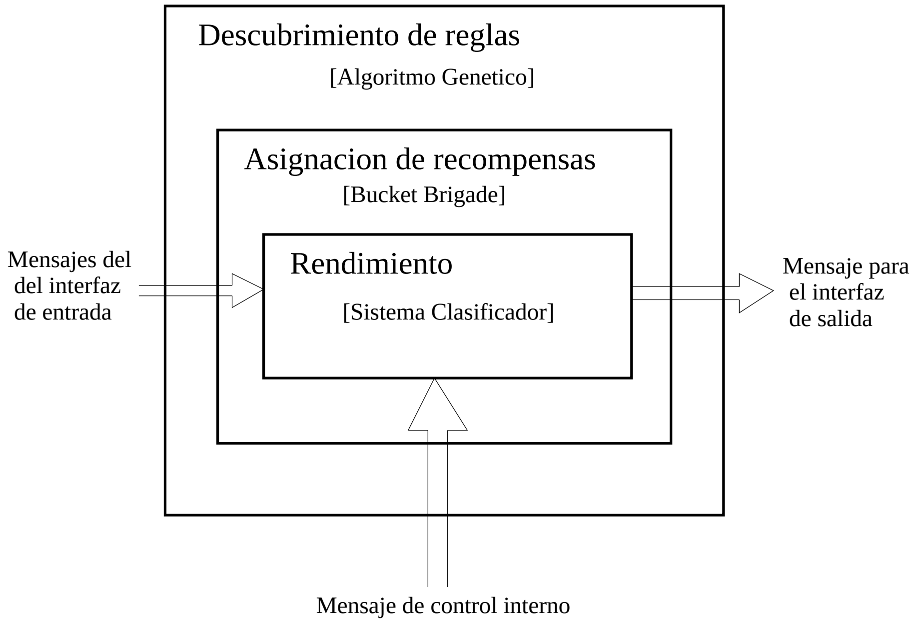


Figura 1.6: Organización general de un sistema clasificador.

En un SC podemos distinguir tres niveles de actividad (como se muestra en la fig. 1.6):

- **Sistema de rendimiento:** Es la actividad que se produce a más bajo nivel y es la encargada de interactuar con el entorno del SC. El sistema de rendimiento suele ser un sistema basado en reglas como la mayoría de los sistemas expertos. Este tipo de reglas son llamadas *clasificadores*. La comunicación con el medio que rodea al SC se realiza a través del paso de mensajes.

- **Sistema de asignación de recompensas:** El sistema de rendimiento debe determinar cuando las reglas son correctas para clasificar. Por consiguiente, es necesario tener un segundo nivel de actividad que

  a) determine la efectividad (bondad para clasificar) de cada uno de los clasificadores (reglas), y

  b) determine las reglas que son incorrectas y deben ser reemplazadas del conjunto de los clasificadores.

  La forma en la cual se determina la bondad para la clasificación de las reglas usadas en el sistema de rendimiento se realiza a través de un algoritmo de aprendizaje por refuerzo, en el cual, una vez que las reglas son lanzadas, el sistema premia a aquellas reglas que se han comportado de acuerdo a la situación existente y castiga a aquellas que han tenido un comportamiento erróneo. Tradicionalmente se vienen usando dos esquemas diferentes para la asignación de recompensas:

  1. **El Algoritmo del Bucket Brigade**. Este algoritmo trabaja con un esquema de aprendizaje local que precisa pocos requerimientos computacionales, tanto de memoria como de tiempo de CPU. Es un algoritmo de asignación de recompensas que se usa en muchos SCs como por ejemplo en [11, 46, 76].

     En un SC este algoritmo se enlaza con el sistema de asignación de recompensas formando un mecanismo que conduce a la competición entre los distintos clasificadores del sistema. Algunas variantes sobre este algoritmo se pueden encontrar en [33, 79].
  2. **El plan de Profit-Sharing**. Este es un esquema de aprendizaje global que presenta un comportamiento mejor en líneas generales que el algoritmo de Bucket Brigade. Básicamente el proceso de elección del clasificador a disparar es semejante al del algoritmo anterior. La diferencia entre ambos esquemas aparece en el método utilizado para la asignación de recompensas. En la literatura se puede encontrar más información sobre este esquema, como por ejemplo en los trabajos [63, 74] y algunas variantes en [63, 162]

- **Sistema de descubrimiento de reglas.** Es el encargado de determinar entre todas la reglas posibles, cuales de ellas pueden ser las más útiles, evaluando una pequeña porción de ellas. Por el hecho de explorar sólo una porción de reglas es posible que entre las examinadas no se encuentren reglas buenas para clasificar, por este motivo es necesario que el sistema permita la incorporación de nuevas reglas reemplazándolas por las no que son útiles.

  El conjunto inicial de reglas se puede obtener aleatoriamente y mediante pequeñas alteraciones y recombinando partes de distintas reglas intentar mejorar su bondad.

Esta forma progresiva de tratar de mejorar las reglas paso a paso combinando información de otras reglas y con alteraciones aleatorias encaja perfectamente con la filosofía de los algoritmos genéticos.

1.4.5 El modelo Pittsburgh en el aprendizaje automático de conceptos

Los algoritmos de adquisición automática de conceptos que utilizan los AGs con el modelo Pittsburgh son llamados Sistemas de Aprendizaje (SAs). Conceptualmente, estos algoritmos son simples de describir porque el método de aprendizaje es directamente la descripción del algoritmo genético. Por tanto, un SA queda determinado en función de las siguientes componentes del algoritmo genético:

- **El espacio de búsqueda y la representación de las reglas:** Básicamente, los SAs usan dos lenguajes para la representación de las reglas.
  - El primero de ellos, conocido como *forma normal disyuntiva modificada*, fué propuesto por Michalski [114] y supone un lenguaje útil para representar conceptos complejos en base a un conjunto disyuntivo de reglas de clasificación. La parte antecedente de la regla se compone de la conjunción de una o más variables, mientras la parte consecuente indica el concepto que debe ser asignado a un ejemplo cuando este se adapte a la parte condición. Además en la parte antecedente se permiten disyunciones internas, es decir, que una misma variable pueda contener una disyunción de valores de su dominio.

    Una regla usando este lenguaje permite construciones del tipo:

    Si (F1 = rojo) y (F2 = esférico o cúbico) entones es de la clase C1.

    suponiendo que el dominio de la variable F1 es {rojo, verde, blanco, azul} y el de la variable F2 es {esférico, cilíndrico, cúbico}.

    Usando este tipo de lenguaje es posible determinar si una variable es útil para describir una clase. Si una variable toma como valor todo su dominio de definición, esto indica que esta variable no es necesaria para describir una clase, por ejemplo, en la siguiente regla

    Si (F1 = rojo o verde o blanco o azul) y
    (F2 es cúbico) entonces es de la clase C1.

    la única variable relevante de la regla es F2, y la regla se puede reformular como:

    Si (F2 = cúbico) entonces es de la clase C1

Normalmente, en este lenguaje se suele utilizar una codificación binaria para representar las reglas, de manera que si una variable puede tomar k valores diferentes, su representación requiere k bits, donde un 0 en el bit i-ésimo indica que la variable no toma el valor i-ésimo de su dominio y un 1 indica que la variable toma el valor i-ésimo de su dominio. Así, el ejemplo anterior tendría la siguiente representación genética, suponiendo que los ejemplos se pueden clasificar en tres clases:

F1	F2	Clase
1111	001	100

- El segundo lenguaje es conocido por (VL_1) y fué propuesto por Michalski y otros en [115]. Este lenguaje es más poderoso que el descrito anteriormente en el sentido que permite descripciones más complejas para representar un concepto. Los elementos más destacables de este lenguaje son los siguientes:
  - (-) Los valores que pueden tomar los atributos (variables antecedente) pueden ser de tres tipos: nominales, intervalares o estructurados (los mismos tres tipos que se describieron anteriormente).
  - (-) Existen un conjunto de relaciones que pueden ser aplicadas para la definición de una regla. Las relaciones permitidas son $(=, <>, >, <, \leq, \geq)$.
  - (-) La parte antecedente de la regla está compuesta por un conjunto de selectores. Cada selector es de la siguiente forma

    [*variable relación valor*]
  - (-) Cuando la relación utilizada es el =, entonces se permite dentro de cada selector que una variable tome diferentes valores de su dominio que son interpretados como disyunciones internas, al igual que en el anterior lenguaje.

  Una regla expresada en este lenguaje es de la forma:

  Si [F1 = verde] [F2 <> cilíndrico] entonces es de la clase C1

  asumiendo que F1 y F2 tiene los mismos dominios que en el caso anterior.

  El problema fundamental cuando trabajamos con el lenguaje VL_1 es encontrar un buena representación genética que permita contemplar todas las posibilidades que ofrece el lenguaje, frente a la forma normal disyuntiva modificada que ofrece una representación binaria bastante confortable.

- **El mecanismo de evaluación:** La función de evaluación para estos sistemas, debido a que cada cromosoma en la población representa un conjunto de reglas, intenta recoger conceptos de la teoría clásica de aprendizaje tales como la consistencia y la completitud [114]. Normalmente, los SA usan los porcentajes de acierto de cada elemento de la población sobre el conjunto de ejemplos para determinar el grado en el que los anteriores conceptos se satisfacen. Además, se suelen incluir en la función de evaluación otros parámetros que son importantes para algunos problemas concretos. Parámetros tales como el costo asociado al uso de los valores de cada variable para construir la descripción, la longitud de las reglas o exigir un nivel de generalidad a las reglas.

- **El conjunto de operadores genéticos usados:** Los SAs necesitan redefinir los operadores genéticos tradicionales para adaptarlos al problema del aprendizaje estableciendo restricciones sobre su uso. Además, se introducen nuevos operadores que permitan evolucionar al sistema más rápidamente hacia soluciones deseadas. Existen diferentes propuestas de nuevos operadores para este modelo, por ejemplo, desde los operadores simples propuestos por De Jong [31] que consisten en la inclusión de dos operadores especiales de mutación, uno que permite la generalización de reglas y el otro permita la especificación de las mismas, hasta algunos más complejos propuestos por Janikow [84] o por Grefenstette [64] que proponen diferentes operadores para diferentes niveles en la evolución del sistema.

GABIL y GIL junto con SAMUEL son los tres SAs más conocidos y referenciados en la literatura. Mientras podemos considerar al sistema GABIL (propuesto por De Jong en [31]) como la versión más simple de un SA, tanto SAMUEL (propuesto por Grefenstette en [64]) y como GIL (propuesto por Janikow en [85]) representan el otro extremo de los SAs, donde incorporan muchos conceptos de los algoritmos de aprendizaje tradicionales.

En los dos siguientes apartados describimos el sistema GABIL como ejemplo de un SA simple y el sistema GIL como máximo exponente de los SAs más sofisticados.

1.4.5.1 El sistema de aprendizaje GABIL

GABIL (GA-based concept learner) es un SA desarrollado por De Jong para el problema del aprendizaje de conceptos simples, donde el conjunto de entrenamiento sólo contiene ejemplos del concepto que debe ser aprendido. GABIL utiliza la forma normal disyuntiva modificada para expresar las reglas y una representación binaria de las mismas, basada en la idea que se mostraba en el apartado anterior. Cada cromosoma tiene una longitud variable (formado por la concatenación de reglas simples) y representa un conjunto de reglas para clasificar el concepto descrito por el conjunto de ejemplos.

Las características fundamentales del sistema vienen dadas por la función de evaluación utilizada y por los nuevos operadores genéticos que son introducidos. Con respecto a la función de evaluación, GABIL mide la bondad de un conjunto de reglas (un individuo de la población) en función del porcentaje de aciertos de este conjunto de reglas sobre el conjunto de ejemplos de entrenamiento de la siguiente forma:

$$fitness(individuo_i) = (porcentaje\ de\ acierto)^2$$

Esta forma de evaluar los individuos es equivalente a medir la consistencia y la completitud de las reglas sobre el conjunto de entrenamiento, en el sentido de que cuanto mayor sea el porcentaje de aciertos, mayor será la satisfacción tanto de la consistencia como de la completitud de las reglas.

Este sistema se caracteriza también por la inclusión de dos nuevos operadores genéticos de mutación llamados *operador de adición de nuevas alternativas* y *operador de eliminación de alternativas*. Estos dos operadores son adaptaciones de los propuestos por Michalski [114] que permiten generalizar y especificar el conjunto de reglas. El primero de los operadores añade una disyunción interna en un atributo de una regla, mientras que el segundo elimina una disyunción interna de un atributo.

Estos dos operadores se introducen con la siguiente intención: si una regla es buena, puede que una regla más general o una regla más especifica que ella sea mejor. Con la introducción de estos dos operadores se permiten movimientos aleatorios de generalización y especialización de reglas cuya permanencia en las sucesivas iteraciones dependerá de la bondad de dichas modificaciones.

Por último destacaremos una modificación que propone el propio De Jong sobre GABIL [31] para transformarlo en un SA adaptativo. La idea consiste en que los dos operadores anteriores no se usen indiscriminadamente sobre todos los individuos de la población, sino que sean los propios individuos, durante el proceso de evolución del sistema, los que determinen cual de los dos operadores se debe aplicar sobre ellos. Para llevar a cabo esta modificación, en la representación de los cromosomas se añaden dos nuevos bits que indicarán si el elemento deber ser generalizado o especificado. El sistema con esta modificación es bautizado con el nombre de GABIL adaptativo.

1.4.5.2 El sistema de aprendizaje GIL

GIL (Genetic-based Inductive Learning) es un sistema bastante más complejo que GABIL que fué desarrollado por Janikow [85]. Utiliza como lenguaje de representación de las reglas una versión restringida del lenguaje VL_1. Esta restricción consiste en que el único operador de relación permitido es el "=". GIL es un SA para el aprendizaje de múltiples conceptos, es decir, que el conjunto de entrenamiento contiene ejemplos de clases diferentes. Además, utiliza una representación genética binaria semejante a la descrita en [31] para GABIL.

La función de evaluación que utiliza GIL es algo más compleja, y tiene en cuenta tanto la consistencia como la completitud del conjunto de reglas sobre el conjunto de entrenamiento, así como el costo asociado a la obtención de los valores para los atributos. La combinación de estos tres factores se realiza de la siguiente forma: define una medida llamada corrección que es una suma ponderada de las medidas de consistencia y completitud

$$\text{corrección}(individuo_i) = \frac{w_1 \ completitud + w_2 \ consistencia}{w_1 + w_2}.$$

esta medida de corrección se combina con el costo asociado del conjunto de reglas dando lugar a la siguiente función de evaluación:

$$fitness(individuo_i) = correccion(individuo_i)(1 + w_3 * (1 - costo(individuo_i)))^f$$

donde w_1, w_2 y w_3 representan la importancia de cada uno de los factores que entran en juego y f es un valor que crece lentamente en el intervalo [0,1] a medida que el genético evoluciona.

Además, la parte más sofisticada del sistema GIL son los operadores genéticos que introduce, así como la forma en que estos se activan en el proceso evolutivo. Los operadores genéticos de GIL se dividen en tres niveles de actuación:

- **A nivel de conjuntos de reglas:** donde actuán sobre todos los individuos de la población, operan con uno o dos individuos al mismo tiempo.
- **A nivel de regla:** donde actuan únicamente sobre una única regla, es decir, sobre un segmento de un individuo de la población.
- **A nivel de la parte condición de una regla:** donde actuán únicamente sobre una condición de una regla, es decir, solo con un segmento de un individuo de la población que corresponda a la parte precondición de una regla.

Sobre cada uno de estos niveles se define un conjunto de operadores, donde a su vez estos operadores pueden clasificarse en tres tipos: de generalización, de especificación y de independencia. Los dos primeros siguen la misma filosofía que los operadores definidos en GABIL para permitir movimientos aleatorios de generalización y especificación, mientras que el último tipo introduce nuevos individuos en la población para aumentar la diversidad del proceso evolutivo. Para una descripción más detallada de todos los operadores genéticos utilizados en GIL ver [85].

Capítulo 2

Una metodología genética para el aprendizaje de reglas difusas

2.1 Un enfoque iterativo para el aprendizaje de conceptos

En la literatura podemos encontrar propuestas muy diferentes para el problema del aprendizaje de múltiples conceptos ([31, 48, 85, 86, 110, 126]). En un intento de clasificar las diferentes propuestas hemos considerado como un elemento fundamental la posible descomposición del problema. Así, consideramos la siguiente clasificación:

a) **Enfoques sin descomposición del problema.** En este primer caso, el aprendizaje de conceptos se considera como un problema indivisible, y el algoritmo trata de extraer un conocimiento que dé una respuesta global a todos ellos. En esta categoría podemos considerar los algoritmos de aprendizaje inductivo basados en redes neuronales, donde toda la información existente es suministrada a la red y esta trata de encontrar una disposición interna que sea capaz de dar una solución al problema.

b) **Enfoques con descomposición del problema.** En esta categoría, podemos encuadrar a aquellos algoritmos que asumen que el problema de la clasificación de los ejemplos se puede descomponer en un conjunto de subproblemas. En este caso, la solución al problema se construye en base a la agregación de las soluciones obtenidas en cada subproblema.

Dentro del enfoque por descomposición del problema, podemos encontrar en la literatura del aprendizaje automático diversas estrategias, tales como las que proporcionan los algoritmos basados en árboles de decisión ([125]). Sin embargo, nos centraremos en nuestro estudio en un tipo especial de estrategia, utilizada ampliamente en diversos algoritmos de aprendizaje, que denominaremos **estrategia iterativa**. Una estrategia iterativa es aquella donde el problema de aprender m conceptos se descompone en m subproblemas cada uno de ellos correspondiente al aprendizaje de un único concepto. Básicamente, esta estrategia fija la clase que se desea aprender y en base a los ejemplos del conjunto de entrenamiento y mediante procesos de generalización y especificación se va construyendo la descripción de la clase. Por tanto, una solución parcial, siguiendo esta estrategia, se compone de una o varias descripciones para un concepto. La solución global se construye como la agregación de las diferentes soluciones parciales.

Esquema 2.1 *Enfoque iterativo con un solo nivel*

0. Sean $X_1, X_2, \ldots, X_n$ las variables predictivas que son incluidas para aprender el comportamiento de un determinado sistema. Sea Y la variable de clasificación definida sobre un dominio discreto de m conceptos. Además, sea E un conjunto con k ejemplos que describen el comportamiento en el pasado de dicho sistema. Sean $GLOBAL$ y $PARCIAL$ dos conjuntos de reglas que inicialmente están vacíos.

1. Seleccionar un concepto B que aún no se ha aprendido.

2. Encontrar un conjunto de descripciones $PARCIAL$ que sean una solución al problema de clasificar los ejemplos del concepto B que se encuentran en el conjunto de entrenamiento E.

3. Añadir las nuevas descripciones al conjunto $GLOBAL$.

$$GLOBAL = GLOBAL \bigcup PARCIAL$$

4. Eliminar las descripciones de $PARCIAL$; $PARCIAL = \emptyset$.

5. Si existen más objetos que aprender, volver al paso 1., en otro caso devolver $GLOBAL$ como solución al problema de aprendizaje.

Normalmente, un algoritmo inductivo para el aprendizaje de reglas guiado por una estrategia iterativa responde a la estructura propuesta en el esquema 2.1. Las principales diferencias entre dos algoritmos inductivos que siguen esta estrategia se encuentran en la forma de definir el paso 2 del esquema anterior. Este paso implica dos características fundamentales en el algoritmo de aprendizaje:

a) un criterio para valorar si un determinado conjunto de descripciones se puede considerar como una solución para representar un determinado concepto, y

b) un algoritmo de búsqueda que permita encontrar dicho conjunto de descripciones.

En general, los algoritmos que siguen esta estrategia se apoyan en las condiciones de consistencia y completitud, propuestas en [114], como criterio para determinar si un conjunto de descripciones es solución al problema de aprender los conceptos. Estas dos condiciones, que ya se describieron en el capítulo anterior, proporcionan un fundamento lógico a los algoritmos de aprendizaje inductivo y aunque Michalski las propone como condición que debe satisfacer la solución global, se puede extender su aplicación de forma trivial a las soluciones parciales. Si todas las soluciones parciales satisfacen las condiciones de consistencia y completitud, entonces la solución global compuesta por la agregación de las soluciones parciales también las satisfacen.

En la literatura podemos encontrar trabajos, tales como [51, 157], que van un paso más alla en la descomposición del problema a aprender. Según el planteamiento de estos autores, el encontrar un conjunto de descripciones que sea una solución al problema de clasificar los ejemplos de un concepto, puede descomponerse a su vez, en un conjunto de subproblemas más simples. En este planteamiento, podemos considerar dos soluciones parciales de diferente orden: una solución parcial de primer orden, que corresponde a la solución parcial del esquema anterior y una solución parcial de segundo orden, que establece que una descripción es solución, si puede ser una descripción que pertenezca al conjunto de descripciones para un determinado concepto. Los algoritmos de aprendizaje que siguen esta filosofía mantienen básicamente la estructura que se propone en el esquema 2.2.

Esta nueva descomposición plantea dos problemas:

a) El número de descripciones necesarias para cada concepto no se conoce a priori.

b) Es necesario establecer un criterio para determinar si una descripción puede considerarse como una solución parcial de segundo orden.

Para dar respuesta a los anteriores problemas, en la mayoría de los casos, se vuelve a recurrir a la extensión de las condiciones de consistencia y completitud para determinar las soluciones de segundo orden. La condición de consistencia es fácilmente extensible, ya que si el conjunto de descripciones para un determinado concepto verifican esta condición, cada una de las descripciones particulares deben verificarla también. Por contra, la condición de completitud no es extensible en este sentido. Sin embargo, la agregación de las descripciones individuales si debe verificar la condición de completitud. Por consiguiente, la condición de completitud establece un criterio para determinar cuándo el conjunto de descripciones obtenidas son suficientes para una determinada clase.

Esquema 2.2 *Enfoque iterativo con dos niveles*

0. Sean $X_1, X_2, \ldots, X_n$ las variables predictivas que son incluidas para aprender el comportamiento de un determinado sistema. Sea Y la variable de clasificación definida sobre un dominio discreto de m conceptos. Además, sea E un conjunto con k ejemplos que describen el comportamiento en el pasado de dicho sistema. Sean $GLOBAL$ y $PARCIAL$ dos conjuntos de reglas que inicialmente estan vacíos.

1. Seleccionar un concepto B que aún no se ha aprendido.

2. Mientras $PARCIAL$ no sea una solución parcial de primer orden hacer:

   2.1. Encontrar una descripción d que sea solución parcial de segundo orden.

   2.2. Añadir d al conjunto de descripciones $PARCIAL$.

   $$PARCIAL = PARCIAL \bigcup d.$$

   2.3 Penalizar las descripciones de $PARCIAL$ para que no vuelvan a ser seleccionadas.

3. Añadir las nuevas descripciones al conjunto $GLOBAL$.

   $$GLOBAL = GLOBAL \bigcup PARCIAL$$

4. Eliminar las descripciones de $PARCIAL$; $PARCIAL = \emptyset$.

5. Si existen más objetos que aprender, volver al paso 1., en otro caso devolver $GLOBAL$ como solución al problema de aprendizaje.

Teniendo en cuenta lo anterior, tenemos criterios para determinar si una descripción es una solución parcial de segundo orden, y en base a ella, criterios para determinar si un conjunto de descripciones pueden considerarse como solución parcial de primer orden.

Definición 2.1 *Sea una descripción d para un determinado concepto T. Diremos que d es una descripción que proporciona una solución parcial de segundo orden al problema de describir T, si d verifica la condición de consistencia.*

Definición 2.2 *Sea un conjunto de descripciones $D = \{d_1, d_2, \ldots, d_m\}$ para un determinado concepto T. Diremos que D es una solución parcial de primer orden al problema de describir T, si $\forall d_i$, d_i es solución parcial de segundo orden y además D verifica la condición de completitud.*

El apartado 2.3. de la descripción de esta metodología es importante, ya que si las descripciones sobre un concepto que se han aprendido no son penalizadas, es posible que vuelvan a aparecer como solución. Una forma muy fácil de penalizar las reglas consiste en eliminar los ejemplos del conjunto de entrenamiento que son representados por alguna de las descripciones ya existentes en el conjunto $PARCIAL$. Esta forma de penalizar reglas es la más común en algoritmos que siguen esta metodología tales como los algoritmos de la familia AQ [115] o el algoritmo CN2 [19].

En esta metodología podemos encontrar soluciones parciales a dos niveles; en el primer nivel la solución esta formada por un conjunto de descripciones para un único concepto que satisface las condiciones de consistencia y completitud, mientras que en el segundo nivel la solución esta formada por una única descripción que satisface la condición de consistencia. En el siguiente apartado estudiamos más detalladamente algunas características que son deseables para las soluciones parciales de segundo nivel.

2.2 SLAVE: un algoritmo de aprendizaje inductivo basado en el modelo iterativo genético

En este capítulo se han descrito los elementos más importantes que constituyen un algoritmo inductivo de aprendizaje de conceptos basados en reglas atributo-valuadas. Además se han propuesto enfoques generales para la construcción de dichos algoritmos.

Un modelo iterativo genético de aprendizaje sigue el esquema 2.2, y usa un algoritmo genético para buscar las soluciones parciales de segundo orden del apartado 2.1.

El enfoque iterativo genético puede considerarse como un nuevo modelo para aprender con algoritmos genéticos, diferente a los modelos Michigan y Pittsburgh, que permite reducir el espacio de búsqueda gracias a la descomposición del problema y además permite que cada subproblema pueda aprenderse de forma paralela. Los primeros algoritmos que han utilizado este enfoque son SIA [157] y SLAVE [51]. Posteriormente, han surgido otros sistemas de aprendizaje con este enfoque tales como [21, 22].

El algoritmo genético necesario para obtener las soluciones parciales de segundo orden, representa en cada individuo de su población una única regla atributo-valuada al igual que la aproximación Michigan, pero a diferencia de ésta, solamente se considera la mejor regla de entre las obtenidas durante el proceso genético.

Podemos destacar dos ventajas fundamentales, comparando este modelo genético con los modelos Michigan y Pittsburgh:

- El modelo iterativo génetico, al igual que el modelo Pittsburgh, no requiere prácticamente ninguna adaptación especial del algoritmo de aprendizaje.
- El modelo iterativo génetico reduce el espacio de búsqueda considerablemente, debido fundamentalmente a su esquema de descomposición del problema. El algoritmo genético busca, en cada iteración, una única solución. En la tabla 2.1, donde N es el número de reglas necesario para describir el problema, K es el número de clases, n es el número de atributos y V es el número de valores por atributo, propuesta en [157] se muestra el tamaño del espacio de búsqueda para el algoritmo genético en cada una de las diferentes propuestas, y se pone de manifiesto la reducción del espacio de búsqueda en el modelo iterativo genético.

Modelo genético	Tamaño del espacio de búsqueda
Pittsburgh	$N^{K(V+1)^n}$
Michigan	$K(V+1)^n$
Iterativo	2^n

Tabla 2.1: Tamaño del espacio de búsqueda para los distintos modelos de aprendizaje con genéticos.

A continuación nos centraremos en describir el sistema de aprendizaje SLAVE (Structure Learning Algorithm in Vague Enviroment) [51], un algoritmo de aprendizaje inductivo, basado en el modelo genético iterativo, definido para extraer en términos de reglas atributo-valuadas el conocimiento de un sistema.

En las siguientes secciones, describimos cada uno de los elementos que componen este algoritmo de aprendizaje.

2.2.1 Representación del conocimiento

El objetivo de un algoritmo de aprendizaje inductivo consiste en establecer una hipótesis para explicar un conjunto de ejemplos que describen el comportamiento de un determinado sistema. El conocimiento que dicha hipótesis aporta para comprender a dicho sistema, viene determinado por el tipo de representación que el algoritmo de aprendizaje utiliza para almacenar la información. Así, si se usa una red neuronal, el algoritmo aprenderá el sistema, pero no es fácil determinar a partir de esta representación las relaciones existentes entre las distintas variables que intervienen en el sistema. Si se desea conocer dicha relación, es necesario utilizar otra representación del conocimiento. Las reglas atributo-valuadas son un tipo de representación del conocimiento que permiten establecer las relaciones existentes entre las variables involucradas en el sistema. El lenguaje DNF (forma normal disyuntiva modificada) propuesto en [115] y descrito en el primer capítulo, es un lenguaje útil para representar conceptos complejos en base a un conjunto disyuntivo de reglas de clasificación. La parte antecedente o descripción de la regla se compone de la conjunción de una o más variables, mientras la parte consecuente indica el concepto que debe ser asignado a un ejemplo cuando este se adapte a la parte condición. Además en la parte antecedente se permiten disyunciones internas, es decir, una misma variable puede contener una disyunción de valores de su dominio.

Ejemplo **2.1** *Sean X_1, X_2 y X_3 tres variables cuyos dominios asociados son D_1={rojo, verde, blanco, azul}, D_2= {esférico, cilíndrico, cúbico} y D_3 ={ 1, 2, 3, 4, 5}respectivamente. Una regla válida en el lenguaje DNF podría ser:*

IF (X_1 is rojo) and (X_2 is esférico o cúbico) and (X_3 is 3 ó 4 ó 5)
THEN Y is C

donde Y es la variable de clasificación y C es un valor de dicha variable.

La posibilidad dada a una variable de contener una disyunción de valores de su dominio, otorga a este modelo de regla una herramienta para generar nuevos valores, en base a la agregación de los mismos. Esta ventaja en la agrupación de valores, se pone de manifiesto en problemas donde existen variables con dominios asociados que mantienen algún tipo de estructura jerárquica o alguna ordenación entre sus valores. Por ejemplo, en el problema anterior, la variable X_3 tiene asociado un dominio ordenado. Esto implica, que la regla anterior, se puede reformular en el siguiente sentido:

IF (X_1 is rojo) and (X_2 is esférico o cúbico) and (X_3 is mayor o igual que 3) THEN Y is C

donde, mayor o igual que 3, *representa la agrupación de los valores* {3,4,5} *del dominio D_3 para la variable X_3.*

Esta agrupación de valores confiere al modelo la posibilidad de construir descripciones más comprensibles. Además, usando este tipo de lenguaje y mediante la agrupación de *valores, es posible determinar si una variable es útil para describir una clase. Si una variable toma como valor todo su dominio de definición, esto indica que esta variable no es necesaria para describir la clase. Por ejemplo, en la siguiente regla*

$$\begin{gathered}\textit{IF } (X_1 \textit{ is rojo o verde o blanco o azul) and}\\ (X_2 \textit{ es cúbico) and}\\ (X_3 \textit{ is 1 o 2 o 3 o 4)THEN } Y \textit{ is } C\end{gathered}$$

la única variable relevante de la regla es X2, y la regla se puede reformular como:

$$\textit{IF } (X_2 \textit{ is cúbico) THEN } Y \textit{ is } C$$

Dada esta propiedad que ofrece el lenguaje DNF y que permite no sólo determinar la relación entre las variables predictivas y la variable de clasificación, sino que además permite establecer el subconjunto de variables predictivas que son necesarias para explicar la variable de clasificación, en el desarrollo de este algoritmo de aprendizaje usamos el siguiente modelo de regla basado en este lenguaje.

Sean n variables antecedente X_i, cada una con un referencial U_i y un dominio finito asociado D_i. Dada una variable consecuente Y con referencial V y un dominio finito asociado F, se propone el modelo de regla

$$\text{IF } X_1 \textit{ is } A_1 \textit{ and } \ldots \textit{ and } X_n \textit{ is } A_n \text{ THEN } Y \textit{ is } B$$

tal que el valor de la variable Y es B, donde $B \in F$ y el valor de la variable X_i is A_i, donde $A_i \in P(D_i)$ y $P(D_i)$ denota el conjunto de las partes D_i.

Por tanto, el conjunto de todas las posibles reglas es

$$\Delta = P(D_1) \times P(D_2) \times \ldots \times P(D_n) \times F.$$

Sobre este modelo de regla realizamos las dos siguientes consideraciones:

1. El modelo de regla podría contener más de una variable de clasificación, pero lo restringimos al uso de una única variable para facilitar tanto la comprensión de este modelo, como del resto de los componentes que constituyen este algoritmo de aprendizaje. Así, la extensión para trabajar con más de una variable de clasificación es directa.

2. En este modelo supondremos, sin pérdida de generalidad, que las variables X_i son variables lingüísticas en el sentido que toman valores sobre un dominio finito difuso D_i, y cada A_i se puede interpretar como una disyunción de etiquetas difusas. En función del tipo de dominio F de la variable consecuente podemos considerar dos

clases diferentes de clasificación: cuando el dominio toma valores sobre un referencial nominal o discreto y cuando el dominio toma valores sobre un referencial continuo. Cada uno de estos problemas requiere un tratamiento distinto, que será estudiado más adelante en esta memoria.

Las variables lingüísticas aportan dos características fundamentales desde el punto de vista del aprendizaje:

a) Permiten establecer discretizaciones flexibles sobre variables con referencial continuo.

b) Permiten dotar de semántica a dichas discretizaciones, de manera que cada etiqueta corresponde a la descripción de un término lingüístico. Esta característica es importante, si se desea que el conocimiento extraído de un determinado problema, sea fácilmente interpretable desde un punto de vista humano.

Una vez definida la herramienta de representación del conocimiento obtenido, es necesario establecer cómo se utilizará dicha herramienta. En la siguientes subsecciones se define el concepto de adaptación entre un ejemplo y una regla, que es necesario tanto para determinar en qué medida los ejemplos refuerzan la veracidad de una regla, cómo para determinar la clasificación de un ejemplo una vez extraído el conjunto de reglas.

2.2.2 Concepto de adaptación

Una vez determinado el modelo de regla a utilizar, es necesario establecer como dichas reglas serán utilizadas durante el proceso de predicción. El concepto clave para determinar la aplicación de las reglas es el de adaptación. La adaptación permite establecer si una regla se puede emplear para clasificar un ejemplo dado. Cuando trabajamos con ejemplos definidos de forma precisa y reglas atributo-valuadas clásicas, este concepto esta claramente definido. Si los valores asociados a las variables predictivas pertenecen al conjunto definido por la descripción de la regla, entonces dicha regla es aplicable para clasificar el ejemplo.

Sin embargo, cuando trabajamos con reglas atributo-valoradas difusas y/o ejemplos expresados de forma imprecisa, es necesario extender el concepto de adaptación, ya que no es posible una aplicación directa del concepto anterior. El primer paso para establecer esta extensión, es determinar la adaptación entre dos conjuntos difusos.

Definición 2.3 *Sean a y b dos conjuntos difusos sobre un mismo conjunto de referencia U, y $*$ una t-norma. Definimos la compatibilidad entre a y b como la siguiente función:*

$$\sigma(a, b) = \sup_{x \in U} \{\mu_a(x) * \mu_b(x)\}$$

donde μ_c es la función de pertenencia al conjunto difuso c.

El modelo de regla descrito anteriormente permite disyunciones internas en la valoración de las variables del antecedente. Por tanto, debemos extender a conjuntos de conjuntos difusos.

Definición 2.4 *Sean Dom_1 y Dom_2 dos dominios constituidos por conjuntos difusos sobre un mismo referencial U. Sean $C_1 \subseteq Dom_1$ y $C_2 \subseteq Dom_2$ cada uno de ellos un conjunto de conjuntos difusos. Definimos la compatibilidad entre estos conjuntos como:*

$$\sigma(C_1, C_2) = \sup_{a \in C_1} \sup_{b \in C_2} \sigma(a, b)$$

Usamos el mismo símbolo σ para ambas definiciones ya que la última definición es una extensión de la primera, y ambas coinciden cuando los conjuntos son unitarios.

En base a estos conceptos, para calcular la adaptación entre un valor v (crisp o difuso) y un conjunto de conjuntos difusos $L \subseteq D$, ambos valores de una misma variable X con dominio difuso D, usamos la siguiente valoración:

$$Poss(v|L) = \frac{\sigma(v, L)}{\sigma(v, D)}$$

que establece el grado de adaptación entre X y v dado que $X = L$ y que se puede interpretar como una distribución de posibilidad.

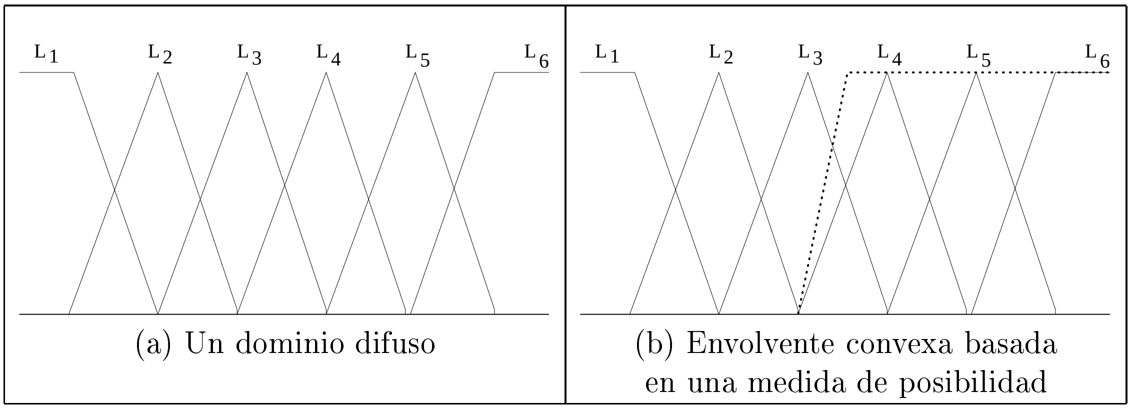


Figura 2.1: Envolvente convexa de etiquetas difusas

$\boxed{\textbf{Ejemplo}}$ **2.2** *Sea X_i una variable lingüística cuyo dominio es el que se describe en la figura 2.1(a). Supongamos que el valor asignado a la variable X_i es $\{L_4, L_5, L_6\}$ en una determinada regla. Utilizando la anterior medida para calcular la adaptación entre un valor v y la valoración $\{L_4, L_5, L_6\}$ se puede interpretar como X_i* es mayor o igual que L_4 *ya que $Poss(v|\{L_4, L_5, L_6\})$ es la envolvente convexa normalizada que se describe en la figura 2.1(b).*

Es importante señalar que la normalización se añade para mantener la propiedad de agrupación de valores descrita en el ejemplo 2.1, ya que directamente los conjuntos difusos no tienen esa propiedad.

Sea e un ejemplo compuesto por diferentes características

$$e = (e_1, e_2, \ldots, e_n, c(e))$$

donde e_i es un conjunto difuso sobre el referencial U_i, y $c(e)$ es la clase asignada al ejemplo e.

Los ejemplos normalmente son valores crisp, sin embargo, el algoritmo de aprendizaje puede utilizar ejemplos difusos sin ninguna dificultad. Además, admitir ejemplos difusos es útil para incluir un grado de imprecisión en los datos. Por ejemplo, podemos utilizar ejemplos difusos cuando tenemos reglas imprecisas en las cuales no tenemos plena confianza y decidimos incluirlas como nuevos ejemplos. Además éstos aparecen cuando tenemos valores perdidos del tipo "no importa" y los reemplazamos por sus dominios completos (un conjunto de conjuntos difusos) los cuales representan que cualquier valor es posible.

En el ejemplo anterior, el valor *no importa* se reemplaza por $v = D$ y

$$Poss(D|L) = \frac{\sigma(D, L)}{\sigma(D, D)} = 1 \; \forall L \subseteq D$$

cuando trabajamos con etiquetas difusas normalizadas.

Para que el concepto de adaptación sea aplicable al modelo de regla definido, es necesario extenderlo al producto cartesiano.

Definición 2.5 *Sea $R_B(A)$ una regla cuya descripción es $A = (A_1, \ldots, A_n)$ $A_i \subseteq D_i$ y su parte consecuente es $B \in F$. Definimos la adaptación entre un ejemplo $e = (e_1, e_2, \ldots, e_n, c(e))$ y el antecedente de una regla como la combinación a través de una t-norma de las adaptación parcial de cada A_i con su componente asociada del ejemplo.*

$$U(e, A) = Poss(e_1|A_1) * Poss(e_2|A_2) * \ldots * Poss(e_n|A_n).$$

y definimos la adaptación entre un ejemplo y el consecuente de una regla como

$$U(e, B) = Poss(c(e)|B).$$

donde c(e) representa la clase o concepto asociado al ejemplo.

La función U establece una medida de adaptación entre la descripción de una regla y un ejemplo, teniendo en cuenta las medidas de posibilidad asociadas a cada variable de la descripción dada la evidencia soportada por el ejemplo. Estas medidas de posibilidad se combinan mediante una t-norma, ya que el nexo de unión entre variables en la descripción del modelo de regla empleado se realiza mediante el conectivo *y*. Por contra, la adaptación del ejemplo con el consecuente de la regla, es directamente la valoración definida anteriormente, al asumir en el modelo de regla que únicamente se admite una variable de clasificación. Para extender esta medida a un modelo de regla con más de una variable de clasificación, se aplica la extensión anteriormente propuesta para el cálculo de la adaptación con el antecedente de la regla.

En base a la definición del grado de adaptación entre un ejemplo y cada una de las reglas que forman parte del conocimiento extraido, se establece el modelo de inferencia. Podemos distinguir dos situaciones diferentes en función del tipo de variable de clasificación que se defina en el problema.

b) **Caso difuso**. La variable consecuente tiene referencial continuo o tiene un referencial crisp con un número elevado de elementos, en cualquier caso se utiliza un dominio difuso para discretizar el referencial. En la literatura se han propuesto distintos modelos de inferencia para este tipo de problemas ([34]). Para nuestro sistema hemos considerado el modelo de inferencia max-min, asociado al método de nitificación (defuzzification) de la media de las alturas ponderada por la adaptación y a la t-norma mínimo.

a) **Caso crisp**. La variable consecuente tiene referencial crisp y no necesita utilizarse un dominio difuso al tener un número suficientemente pequeño de elementos. En estos casos, el dominio suele corresponder al propio referencial.

   En este caso, se devuelve como clasificación, la clase de la regla que presenta mejor adaptación entre su antecedente y el ejemplo.

   Definición 2.6 *Dado un ejemplo e y un conjunto de reglas difusas $R_{B_j}(A^i)$, la regla que determina la clasificación de este ejemplo es $R_{B_s}(A^c)$ si verifica que*

$$U(e, A^c) \geq U(e, A^i) \; / i = 1 \ldots p$$

   y B_s el valor consecuente de la regla con antecedente A^c, donde p es el número de reglas del conjunto de reglas.

   A diferencia del caso anterior donde las reglas cooperan para obtener una salida, en esta segunda situación las reglas compiten entre si y una única regla puede ser aplicada para obtener la clasificación del ejemplo. Por tanto, durante este proceso se puede producir una situación donde la adaptación no sea suficiente para determinar la salida y se producen conflictos. Un conflicto se produce, cuando existen dos o más reglas que representan diferentes conceptos y tienen el mismo mejor grado de adaptación entre sus antecedentes y el ejemplo. Por consiguiente, se hace necesario establecer mecanismos que resuelvan estos conflictos, de forma que asociados a la inferencia produzcan una mejor representación del sistema. Normalmente, los mecanismos de resolución de conflictos se establecen teniendo en cuenta cierto meta-conocimiento que se extrae sobre el problema durante el proceso de aprendizaje, es decir, conocimiento que indica como usar el conocimiento extraido.

En la literatura, podemos encontrar diferentes mecanismos de resolución de conflictos [21, 83].

De lo anterior, se deduce que un algoritmo de aprendizaje inductivo que use el modelo de regla y el módulo de inferencia previamente descrito para el caso crisp, debe extraer dos tipos de conocimiento:

- El conocimiento que permite establecer las relaciones entre las variables predictivas y la variable de clasificación.
- El conocimiento que permite definir como se debe emplear el anterior conocimiento.

Para extraer este segundo tipo de conocimiento es necesario conocer el modelo general de aprendizaje que utiliza este modelo de regla. Así que más adelante se estudiarán detenidamente mecanismos de resolución de conflictos para el caso crisp.

2.2.3 Ejemplos positivos y negativos

Los algoritmos de aprendizaje inductivo basados en el modelo iterativo no pueden utilizar directamente el modelo de inferencia para determinar las reglas que se deben obtener para establecer una correcta descripción de un concepto. Por esta razón, estos sistemas establecen valoraciones para determinar la bondad de las reglas atendiendo a otros parámetros. Prácticamente la totalidad de estos algoritmos, tales como [51, 115, 157] y algunos otros basados en otros modelos como [31, 85, 125], utilizan como medida principal en dicha valoración, la capacidad que tiene cada descripción de representar correctamente los ejemplos asociados al concepto que se desea aprender en relación a los ejemplos que representa de otros conceptos y que se encuentran en el conjunto de entrenamiento. Estos son conocidos como ejemplos positivos y ejemplos negativos a una regla.

Debido a que la definición de ejemplo positivo y negativo esta estrechamente relacionada con el concepto de adaptación entre las reglas y los ejemplos, y a que SLAVE al trabajar con reglas difusas y ejemplos expresados de forma inexacta tiene una definición propia de adaptación, es necesario redefinir dichos conceptos para esta nueva situación.

La utilización de la función de adaptación definida en SLAVE, provoca una relajación de estos conceptos. En el caso de que la regla sea crisp, si un ejemplo afecta a una regla, entonces, o bien éste apoya completamente la veracidad de la regla, o bien la contradice completamente. En el caso de que las regla sea difusa, un mismo ejemplo puede apoyar y contradecir la veracidad de la regla en un cierto grado. Por consiguiente, la positividad o negatividad de los ejemplos a las reglas, se encuentra graduada por la medida de adaptación. Teniendo en cuenta esta situación, podemos definir el conjunto de ejemplos positivos y negativos a la misma.

Definición 2.7 *Dado un conjunto de entrenamiento E y una regla $R_B(A)$, diremos que el conjunto de ejemplos positivos para la regla $R_B(A)$ es el siguiente subconjunto difuso de E*

$$E^+(R_B(A)) = \{(e, U(e, A) * U(e, B)) \mid e \in E\}$$

*donde $A = (A_1, \ldots, A_n)$, y * es una t-norma.*

Definición 2.8 *Dado un conjunto de entrenamiento E y una regla $R_B(A)$, diremos que el conjunto de ejemplos negativos para la regla $R_B(A)$ es el siguiente subconjunto difuso de E*

$$E^-(R_B(A)) = \{(e, U(e, A) * U(e, \overline{B})) \mid e \in E\}$$

*donde $A = (A_1, \ldots, A_n))$, $\overline{B} = \{b \in F / b \neq B\}$ y * es una t-norma.*

Definición 2.9 *Dado un conjunto de entrenamiento E y una regla $R_B(A)$, diremos que el conjunto de ejemplos cubiertos por la regla $R_B(A)$ es el siguiente subconjunto difuso de E*

$$E(R_B(A)) = \{(e, U(e, A)) \mid e \in E\}$$

*donde $A = (A_1, \ldots, A_n))$, y * es una t-norma.*

Una vez que hemos definido los anteriores conjuntos, podemos determinar el número de ejemplos positivos y negativos para una regla. Así, notaremos $|J| = \sum_{u \in U} \mu_J(u)$, como el cardinal de un conjunto difuso J. Aplicando este cardinal a los anterior conjuntos, obtenemos las siguientes definiciones:

Definición 2.10 *En base a los conceptos anteriores, definimos:*

- *El número de ejemplos positivos a la regla $R_B(A)$: $n^+(R_B(A)) = |E^+(R_B(A))|$*
- *El número de ejemplos negativos a la regla $R_B(A)$: $n^-(R_B(A)) = |E^-(R_B(A))|$*
- *El número de ejemplos cubiertos por la regla $R_B(A)$: $n(R_B(A)) = |E(R_B(A))|$*

Las medidas aquí propuestas formarán la base para establecer la valoración de las reglas en el sistema de aprendizaje SLAVE.

2.2.4 Selección de reglas

Las soluciones parciales de segundo orden deben satisfacer la condición de consistencia. Sin embargo, la preservación de esta propiedad en la aplicación de los algoritmos a problemas reales es dificil, debido fundamentalmente a la aparición de inconsistencias, ruido o imprecisiones en el conjunto de entrenamiento. En estos casos, lo que parece más apropiado es buscar entre las descripciones posibles, aquellas que ésten más cerca de verificar la condición de consistencia. Una de las alternativas más comunes para evaluar esta cercanía consiste en graduar la verificación de la condición.

Para la gradación de la condición de consistencia proponemos una restricción estricta en el espacio de soluciones parciales de segundo orden, de manera que establecemos que descripciones son admisibles y que descripciones no lo son para la representación de un concepto. Esta separación en el espacio de soluciones se realiza teniendo en cuenta el nivel de distorsión que es admisible en una descripción.

Definición 2.11 *Definimos el conjunto de descripciones admisibles como solución parcial de segundo orden al conjunto Δ^k, tal que*

$$\Delta^k = \{R_B(A) \mid n^-(R_B(A)) < kn^+(R_B(A)) \text{ y } n^+(R_B(A)) > 0\}$$

donde $n^+(R_B(A))$ y $n^-(R_B(A))$ son el número de ejemplos positivos y negativos a la regla $R_B(A)$ y $k \in]0,1]$.

En la anterior definición, inicialmente propuesta en [51], el parámetro k establece el máximo porcentaje de ruido que es admitido para considerar a la descripción como una posible solución. En este sentido, el concepto de ruido se aleja de su descripción física como un pequeña distorsión en los datos y se generaliza a todos aquellos ejemplos que existen en el conjunto de entrenamiento que no permiten que una determinada descripción verifique la condición de consistencia. Llamaremos a k, **parámetro de consistencia** porque permite relajar la condición de consistencia variando dicho parámetro y por consiguiente, consideramos que cualquiera de las descripciones existentes en dicho conjunto son una solución parcial de segundo orden.

Además, realizamos un definición especial del conjunto de soluciones que satisfacen completamente la condición de consistencia que denominamos como

$$\Delta^0 = \{R_B(A) \mid n^-(R_B(A)) = 0 \text{ y } n^+(R_B(A)) > 0\}.$$

La siguiente proposición pone de manifiesto la relación existente entre dos espacios de soluciones admisibles.

Proposición 2.1 *Sea $R_B(A) \in \Delta$ una regla del espacio de posibles soluciones, y sean k_1 y k_2 dos parámetros de consistencia que verifican que $k_1 \leq k_2$, entonces*

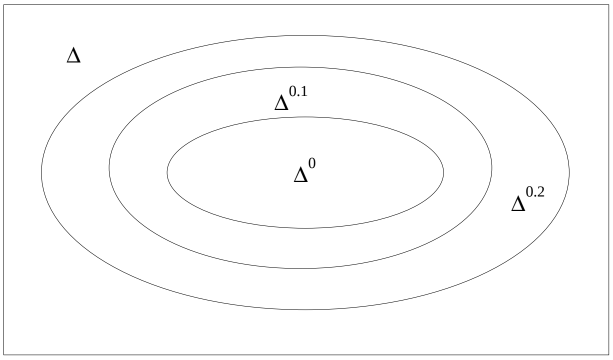


Figura 2.2: Estructura del espacio de soluciones parciales

$$\text{Si } R_B(A) \in \Delta^{k_1} \Rightarrow R_B(A) \in \Delta^{k_2}$$

Demostración. *Si $R_B(A) \in \Delta^{k_1}$ con $k_1 \neq 0$, entonces se verifica que $n^+(R_B(A)) > 0$ y que $n^-(R_B(A)) < k_1 n^+(R_B(A))$.*

Por ser $k_2 > k_1$, se satisface que $k_1 n^+(R_B(A)) < k_2 n^+(R_B(A))$.

Por tanto, $n^-(R_B(A)) < k_1 n^+(R_B(A)) < k_2 n^+(R_B(A))$, lo que implica que

$$R_B(A) \in \Delta^{k_2}$$

Proposición 2.2 *El conjunto de descripciones que satisfacen completamente la condición de consistencia es un subconjunto de cualquier espacio de soluciones parciales Δ^k, es decir,*

$$\Delta^0 \subseteq \Delta^k \;,\; \forall k \in]0,1].$$

La propiedad anterior refleja que el espacio de soluciones Δ mantiene una estructura que relaciona cualesquiera dos subespacios de soluciones parciales a través de la operación $\subseteq$, de manera que dicho espacio tiene una estructura semejante a la propuesta en la figura 2.2. donde cualquier restricción Δ^k contiene las descripciones que satisfacen completamente la condición de consistencia.

El conjunto Δ^k presenta además otra propiedad que dota a dichas restricciones sobre el espacio global de soluciones parciales de consistencia interna. Antes de la presentación de dicha propiedad necesitamos el siguiente lema:

Lema 2.1 *Sean $R_{B_1}(A)$ y $R_{B_2}(A)$ dos reglas con $B_1 \neq B_2$, entonces*

$$n^-(R_{B_1}(A)) \geq n^+(R_{B_2}(A))$$

Demostración. *Usando la medida de adaptación tenemos que*

$$U(e,\overline{B_1}) = \frac{sup_{b\in\overline{B_1}}\sigma(c(e),b)}{\sigma(c(e),F)} \geq \frac{\sigma(c(e),B_2)}{\sigma(c(e),F)} = U(e,B_2)$$

Por tanto,

$$n^-(R_{B_1}(A)) = \sum_{e \in E} U(e,A) * U(e,\overline{B_1}) \geq \sum_{e \in E} U(e,A) * U(e,B_2) = n^+(R_{B_2}(A))$$

Proposición 2.3 *Si la regla* $R_{B_1}(A) \in \Delta^k$ *entonces la regla* $R_{B_2} \notin \Delta^k$, $\forall B_2 \in F$, *tal que* $B_2 \neq B_1$.

Demostración. *Supongamos que ambas reglas* $R_{B_1}(A)$ *y* $R_{B_2}(A)$ *pertenecen a* Δ^k. *Por tanto, ambas reglas satisfacen que:*

$$n^-(R_{B_1}(A)) < kn^+(R_{B_1}(A))$$

$$n^-(R_{B_2}(A)) < kn^+(R_{B_2}(A))$$

Usando en estas expresiones el lema anterior tenemos:

$$n^+(R_{B_2}(A)) \leq n^-(R_{B_1}(A)) < kn^+(R_{B_1}(A))$$

$$n^+(R_{B_1}(A)) \leq n^-(R_{B_2}(A)) < kn^+(R_{B_2}(A))$$

Por consiguiente se obtiene que

$$n^+(R_{B_2}(A)) < kn^+(R_{B_1}(A)) < k^2 n^+(R_{B_2}(A))$$

y estas expresión implica que

$$k^2 > 1$$

ya que $n^+(R_{B_2}(A)) > 0$, *y este resultado es contradictorio con el rango en el que se mueve el parámetro* $k \in [0,1]$.

La proposición anterior, nos asegura que si una descripción es admisible para representar un concepto, esa misma descripción es inadmisible para el resto de los conceptos. Por consiguiente, en el conjunto Δ^k no existen reglas contradictorias, para cualquier valor que tome el parámetro k.

A pesar de que la anterior restricción sobre el espacio de soluciones, permite resolver el problema de la ausencia de descripciones completamente consistentes, y por tanto, es aplicable a conjuntos de ejemplos con ruido presenta el siguiente inconveniente que ilustramos con el siguiente ejemplo:

Ejemplo 2.3 *Sean* r_1 *y* r_2 *dos reglas que pertenecen al espacio* Δ^k *con* $k > 0$. *Por tanto, ambas reglas son válidas como solución parcial. Supongamos que para un* $k_1 \leq k$, *se verifica que* $r_1 \in \Delta^{k_1}$ *pero* $r_2 \notin \Delta^{k_1}$. *Bajo esta situación la regla* r_1 *esta más cerca de satisfacer la condición de consistencia que la descripción* r_2, *sin embargo, ambas soluciones en el espacio* Δ^k *son igualmente deseables.*

La situación expuesta en el ejemplo, pone de manifiesto que además de discriminar entre soluciones admisibles y no admisibles, es necesario también discriminar dentro de las soluciones admisibles la cercanía a la verificación de la condición de consistencia. Con este planteamiento, proponemos una definición graduada de la condición de consistencia en base a dos párametros k_1 y k_2, de manera que si la regla pertenece a Δ^{k_1} la nueva condición se satisface a grado uno, si la regla no pertenece a Δ^{k_2} la condición se satisface a grado cero y entre ambas situaciones se establece el grado de satisfacción como una variación lineal entre ambos parámetros.

Definición 2.12 *Definimos el grado en el cuál una descripción $R_B(A)$ satisface la condición de consistencia débil como*

$$\Gamma_{k_1k_2}(R_B(A)) = \begin{cases} 1 & si\ R_B(A) \in \Delta^{k_1} \\ \frac{k_2n^+(R_B(A))-n^-(R_B(A))}{n^+(R_B(A))(k_2-k_1)} & si\ R_B(A) \notin \Delta^{k_1}\ y\ R_B(A) \in \Delta^{k_2} \\ 0 & en\ otro\ caso \end{cases}$$

donde $k_1, k_2 \in [0,1]$ y $k_1 < k_2$, $n^+(R_B(A))$, es el número de ejemplos positivos y $n^-(R_B(A))$ es el número de ejemplos negativos a la regla $R_B(A)$.

Los parámetros k_1 y k_2 se pueden interpretar como umbrales de ruido. El parámetro k_1 fija el nivel máximo de ruido admisible para considerar que una regla satisface completamente la condición de consistencia débil. En el otro extremo se situa el parámetro k_2, fijando el nivel mínimo de ruido a partir del cual una regla no se debe considerar como solución parcial de segundo orden.

Esta nueva definición graduada para la evaluación de la consistencia permite resolver el inconveniente puesto de manifiesto en el ejemplo anterior, pero es interesante comprobar si esta definición mantiene las buenas propiedades que presentan los conjuntos Δ^k descritos anteriormente. El lema siguiente nos sirve de base para plantear las propiedades de la condición de consistencia débil.

Lema 2.2 *Las reglas que son completamente consistentes satisfacen a grado uno la condición de consistencia débil.*

Demostración. *Las reglas que satisfacen a grado uno la condición de consistencia débil, son aquellas que pertenecen al conjunto de soluciones admisibles Δ^{k_1}. Aplicando el corolario 2.2, tenemos que el conjunto de reglas que satisfacen completamente la condición de consistencia Δ^0 son un subconjunto de cualquier Δ^k para cualquier $k \in [0,1]$. Por consiguiente, también son un subconjunto en especial de Δ^{k_1}.*

Proposición 2.4 *Dada una descripción A, existe un único concepto B_1 para el cuál la regla $R_{B_1}(A)$ tiene un grado de consistencia mayor estrictamente que cero, es decir, $\Gamma_{k_1k_2}(R_{B_1}(A)) > 0$ y $\Gamma_{k_1k_2}(R_{B_2}(A)) = 0$, $\forall B_2 \neq B_1$.*

Demostración. *Supongamos que* $\Gamma_{k_1k_2}(R_{B_1}(A)) > 0$. *Aplicando la definición de la condición de consistencia débil sabemos que* $R_{B_1}(A) \in \Delta^{k_2}$.

De la proposición 2.3 se deduce que si $R_{B_1}(A) \in \Delta^{k_2}$ *entonces* $R_{B_2}(A) \notin \Delta^{k_2}$ $\forall B_2 \neq B_1$.

Si $R_{B_2}(A) \notin \Delta^{k_2}$, *aplicando la definición de grado de satisfacción de la condición de consistencia débil, sabemos que* $\Gamma_{k_1k_2}(R_{B_2}(A)) = 0$, $\forall B_2 \neq B_1$.

En base a los lemas y proposiciones anteriores, se demuestra que la condición de consistencia débil, mantiene las propiedades de los conjuntos Δ^k en el sentido de que el conjunto de descripciones que son completamente consistentes tienen grado uno de consistencia débil y dada una descripción A existe un único concepto B para el cuál la regla $R_B(A)$ tiene un grado de consistencia mayor estrictamente que cero, es decir, no existen en el espacio de búsqueda reglas contradictorias.

2.2.4.1 Grado de completitud

Como ya se ha dicho en esta memoria, las soluciones parciales de segundo orden deben satisfacer la condición de consistencia. Este requisito que es necesario para mantener la verificación de dicha condición no parece suficiente si queremos extraer un conocimiento que aporte una comprensión del sistema que se trata de aprender. Supongamos que para cada ejemplo del conjunto de entrenamiento se construye una regla. Si suponemos además que no existen inconsistencias en el conjunto de ejemplos, cada descripción satisface la condición de consistencia, y el conjunto de todas las descripciones verifica también la condición de completitud. Por consiguiente este conjunto de descripciones es una solución parcial de primer orden. Llegados a este punto, cabe preguntarse ¿Se ha extraido algún conocimiento nuevo sobre el conjunto de ejemplos? En este sentido, parece lógico pedir que la solución parcial de segundo orden satisfaga alguna otra propiedad.

Una de las propiedades que pueden ser exigibles a un algoritmo de aprendizaje es que las soluciones parciales de primer orden contenga el mínimo número de descripciones posibles sobre el concepto que se desea aprender. Si extendemos esta idea a las soluciones de segundo orden podemos establecer que la mejor descripción es aquella que verificando la condición de consistencia este *más cerca* de satisfacer la condición de completitud. ¿Cómo evaluar la cercanía a la verificación de esta condición? La condición de completitud se satisface cuando todos los ejemplos del conjunto de entrenamiento que describen un determinado concepto se representan por al menos una de las descripciones obtenidas para dicho concepto. En base a esta característica definimos el concepto de grado de completitud para una regla.

Definición 2.13 *El grado de completitud de una regla* $R_B(A)$ *se define como*

$$\Lambda(R_B(A)) = \frac{n^+(R_B(A))}{n_B}$$

donde $n^+(R_B(A))$ es el número de ejemplos del conjunto de entrenamiento del concepto B que son cubiertos por la regla $R_B(A)$ y n_B es el número de ejemplos existentes en el conjunto de entrenamiento que describen el concepto B,

De la anterior definición se deduce que si $\Lambda(R_B(A)) = 1$ entonces la regla $R_B(A)$ satisface la condición de completitud, y por tanto, cuanto más cercano a 1 este este valor, más cerca se encuentra la descripción de satisfacer esta condición.

Por consiguiente, parece hasta el momento, que una regla que sea solución parcial de segundo orden debe satisfacer en alto grado tanto la condición de consistencia débil como el grado de completitud. Por consiguiente, si ambos factores son relevantes para establecer una medida de la bondad de la regla, es necesario articular algún criterio que permita combinar ambos factores. En el siguiente apartado estudiamos diferentes combinaciones de estos factores.

2.2.4.2 Combinación de consistencia y completitud

De acuerdo con las definiciones anteriores, debemos adaptar el criterio utilizado para establecer una solución parcial de segundo nivel. Este criterio debe tener en cuenta tanto la maximización del grado de consistencia débil, como el grado de completitud, pero ¿cómo combinarlos? Podemos aplicar varias posibilidades:

a) Establecer una evaluación funcional lexicográfica (EFL). Una EFL se define como secuencia de criterios:

$$EFL : (c_1, c_2, \ldots, c_s)$$

donde c_1 es un criterio elemental.

Dado un conjunto de descripciones $D = \{d_1, d_2, \ldots, d_m\}$ y una secuencia ordenada de criterios de evaluación $(c_1, c_2, \ldots, c_s)$, el EFL determina la descripción más preferible de la siguiente forma: En un primer paso, todas las descripciones se evaluan utilizando el criterio c_1. Todas aquellas descripciones que alcanzan el mismo máximo valor de satisfacción del criterio son retiradas del conjunto D. Las descripciones retiradas se evaluan según el criterio c_2 y se reducen de forma similar. El proceso continua hasta que únicamente una descripción es retirada y esta descripción es considerada como la mejor.

Por tanto, existen dos posibles evaluaciones funcionales lexicográficas considerando el grado de completitud y la condición de consistencia débil como dos criterios.

a1) **Considerar el orden** $(\Gamma_{k_1,k_2}, \Lambda)$ Esta ordenación provoca que se prefieran descripciones que satisfagan en un alto grado la condición de consistencia débil, frente a su grado de completitud. Este criterio de preferencia conserva la idea original, es decir, lo importante es que la regla sea consistente. Este criterio está sometido al mismo inconveniente que se describió anteriormente, se tenderá a seleccionar reglas que esten soportadas por pocos ejemplos y por consiguiente, serán necesarias muchas descripciones para representar un concepto.

a2) **Considerar el orden** $(\Lambda, \Gamma_{k_1,k_2})$ Esta ordenación prefiere descripciones que satisfagan la completitud frente a la consistencia. La aplicación de este EFL puede llevar a la obtención de pocas descripciones para los conceptos, pero excesivamente ruidosas.

b) Maximizar una combinación de ambas valoraciones mediante una función de agregación de información, ya que lo que se desea es una descripción que satisfaga en un alto grado la completitud y en un alto grado la condición de consistencia débil. Si utilizamos como función de agregación el producto, la mejor descripción será aquella que haga máximo esta combinación, es decir, aquella que verifique

$$\max_{A \in D} \{\Lambda(R_B(A)) \times \Gamma_{k_1 k_2}(R_B(A))\}$$

donde $D = P(D_1) \times P(D_2) \times \ldots \times P(D_n)$.

La combinación de ambos valores permite obtener descripciones que simultaneamente son completas y consistentes en el máximo grado posible. Además, utilizamos la t-norma producto, por ser la combinación a través de este operador muy interactiva frente a otras posibles propuestas.

2.2.4.3 Simplicidad y comprensibilidad

Utilizando el criterio anterior podemos encontrar varias soluciones que obtengan la máxima valoración. El módulo de selección de reglas tiene que decidir cual de ellas se devolverá como la mejor descripción para la clase que se está aprendiendo. Un criterio definido para resolver estas situaciones podría decir que cualquiera de ellas es una buena solución. Sin embargo, uno de los objetivos propuestos en la definición de este algoritmo es encontrar reglas que esten soportadas por los ejemplos, y sean fáciles de comprender. En este sentido, el criterio de decisión ante estas situaciones se debe establecer en función de la simplicidad de la regla. A continuación pasamos a formalizar estos criterios.

Definición 2.14 *Dada una regla $R_B(A)$ tal que $A = (A_1, A_2, \ldots, A_n)$ y cada $A_i \subseteq D_i$ representa el valor asignado a un variable X_i. Diremos que la variable X_i es irrelevante para describir el concepto B si A_i es la disyunción de todos los valores del dominio D_i, es decir,*

$$A_i = D_i$$

Definiremos además como $IRREV(A)$ al número de variables de la regla $R_B(A)$ que satisfacen la condición de irrelevancia y $REV(A)$ al número de variables de la regla $R_B(A)$ que no satisfacen la anterior condición.

Sobre la definición anterior debemos especificar una aclaración: cuando una variable X_i toma como valor A_i todo su dominio de definición D_i podemos eliminarla de la descripción de la regla gracias al concepto de adaptación definido en la sección 2.2.2. El uso de la normalización permite que la valoración asignada a cualquier valor sobre el conjunto completo de estiquetas sea

$$Poss(D_i|v) = 1 \ \forall v$$

y además

$$a * 1 = a \text{ para cualquier t-norma } *.$$

Podemos considerar que una regla es más simple cuanto mayor es el número de variables irrelevantes, puesto que esto conlleva la obtención de descripciones más generales de los conceptos. Por tanto, a partir del concepto de irrelevancia podemos definir el concepto de simplicidad.

Definición 2.15 *Dada una regla $R_B(A)$ tal que $A = (A_1, A_2, \ldots, A_n)$, diremos que el grado de simplicidad asociado a esta regla es:*

$$S(R_B(A)) = \frac{IRREV(A)}{n}$$

Cuando tratamos de extraer el conocimiento necesario para comprender el comportamiento de un sistema, podemos querer que dicho conocimiento represente dos tipos descripciones diferente con respecto al concepto:

a) Descripciones que permiten distinguir el concepto del resto de los que se desean aprender. Este tipo de descripciones se llaman **descripciones discriminantes**.

b) Descripciones que permitar conocer todas las variables que tienen influencia para describir este concepto. Este tipo de descripciones se llaman **descripciones características**.

Estos dos tipos de descripciones se asemejan a las propuestas en [115] y descritas en el primer capítulo, aunque nosotros proponemos un sentido distinto para las descripciones características. Según [115] las descripciones características únicamente deben verifican la condición de completitud, sin embargo, desde nuestra interpretación, estas descripciones deben verificar también la condición de consistencia.

Por tanto, de los anteriores tipos de descripciones, las que buscaremos serán las descripciones discriminantes minimales, es decir las que implican el menor número de variables relevantes y las descripciones características maximales, es decir, las descripciones consistentes que implican el menor número posible de variables irrelevantes.

Utilizando el concepto de simplicidad, podemos obtener ambos tipos de descripciones, de manera, que las descripciones discriminantes minimales se obtienen maximizando el valor de la simplicidad, mientras que las descripciones características máximales se pueden obtener mediante la minimización del concepto de simplicidad.

Por consiguiente, un algoritmo de aprendizaje que tenga en cuenta el concepto de simplicidad puede ser aplicable para obtener estos tipos de descripciones.

Una vez que hemos propuesto el concepto de simplidad de una regla, vamos a definir el concepto de comprensibilidad. Este concepto trata de evaluar la simplicidad de los valores asignados a una determinada variable en una regla. Veamos de forma intuitiva su comportamiento con el siguiente ejemplo:

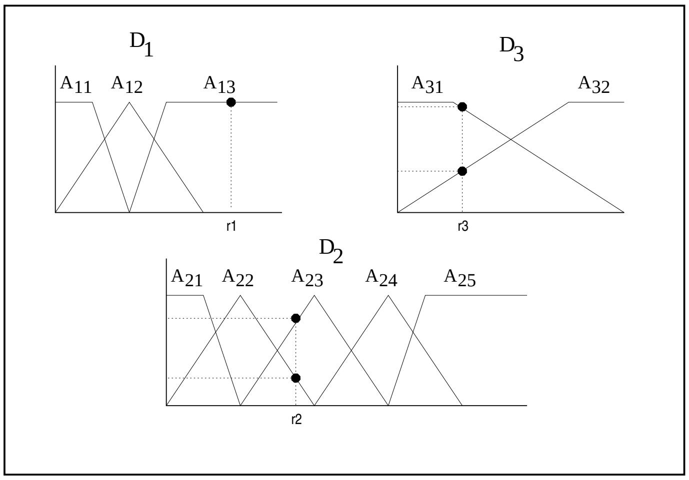


Figura 2.3: Dominio de las variables X_1, X_2, X_3

$\boxed{\textbf{Ejemplo}}$ **2.4** *Sea X_2 una variable cuyo dominio ordenado D_2 es el definido en la figura 2.3. Supongamos los dos siguientes valores para la variable X_2: $X_2 = \{A_{33}, A_{34}, A_{35}\}$ y $X_2 = \{A_{33}, A_{35}\}$. El primer valor se puede expresar en la descripción de una regla como*

$$\ldots \ y \ X_2 \ es \ \{A_{33}, A_{34}, A_{35}\} \ y \ \ldots$$

pero, podemos reformularlo como

$$\ldots \ y \ X_2 \ es \ mayor \ o \ igual \ que \ A_{33} \ y \ \ldots$$

mientras el segundo valor sólo puede ser expresado como

$$\ldots \ y \ X_2 \ es \ \{A_{33}, A_{35}\} \ y \ \ldots$$

Si tuviéramos que decidir entre estos valores en una situación de igual valoración, podemos observar que el primer valor presenta una descripción más comprensible desde un punto de vista humano.

Por tanto, si los valores que toma una determinada variable siguen una secuencia continua sobre un dominio ordenado, el valor de dicha variable, se puede interpretar como la envolvente convexa de dichos valores. Este hecho, implica que los valores asociados a las variables, puedan ser agrupados aumentando la comprensibilidad de esta regla. En este sentido, definimos el concepto de valor agrupado para una variable.

Definición 2.16 *Dado un valor $A_i \subseteq D_i$ para una variable X_i, y siendo D_i un dominio ordenado asociado a la variable X_i, decimos que la variable X_i toma un valor estable A_i, sii el valor A_i esta constituido por una secuencia de valores primitivos adyacentes en el dominio D_i.*

Además, dada una regla $R_B(A)$ tal que $A = (A_1, A_2, \ldots, A_n)$ y cada $A_i \subseteq D_i$ representa el valor de un variable X_i, definimos como $EST(A)$ el número de variables X_i que satisfacen:

i) X_i tiene un dominio asociado ordenado D_i.

ii) El valor A_i que toma D_i es un valor estable.

A partir de esta definición, podemos establecer una valoración sobre la comprensibilidad de una regla:

Definición 2.17 *Dada una regla $R_B(A)$ definimos el grado de comprensibilidad de una regla como:*

$$CP(R_B(A)) = \frac{1 + EST(A)}{1 + p}$$

donde p es el número de variables que tienen un dominio asociado ordenado.

Con la medida propuesta establecemos un criterio de preferencia en el proceso de búsqueda, de manera que el sistema tenderá a obtener soluciones parciales de segundo orden que maximizen lo más posible la comprensibilidad de las asignaciones a las variables con dominio ordenado.

Teniendo en cuenta las definiciones de simplicidad y de comprensibilidad, cada posible descripción puede ser evaluada atendiendo a tres factores. Por tanto, utilizaremos una EFL, donde consideraremos como primer criterio la satisfacción de la combinación de consistencia y completitud, como segundo criterio la simplicidad de las misma, que podrá ser valorada, como la maximización o minimización del concepto de simplicidad y por último se establecerá el criterio de comprensibilidad por ser el menos importante de los tres.

Por consiguiente, dada una regla $R_B(d)$, definimos la valoración de la regla $R_B(d)$ como la función

$$\rho : \Delta \longrightarrow [0, 1]^3$$

$$\rho(R_B(d)) = (\Lambda(R_B(d)) \times \Gamma_{k_1k_2}(R_B(d)), S(R_B(d)), CP(R_B(d))).$$

Usaremos una evaluación funcional lexicográfica que para problemas que requieran descripciones discriminantes usará los criterios

$$EFL(\rho(R_B(A))) = (\max_{A\in D}\{\ \Lambda(R_B(A)) \times \Gamma_{k_1k_2}(R_B(A))\}, \max_{A\in D}\{S(R_B(A))\}, \max_{A\in D}\{CP(R_B(A))\})$$

y para descripciones características usará

$$EFL(\rho(R_B(A))) = (\max_{A\in D}\{\ \Lambda(R_B(A))\times\Gamma_{k_1k_2}(R_B(A))\}, \max_{A\in D}\{1-S(R_B(A))\}, \max_{A\in D}\{CP(R_B(A))\}).$$

Asumiremos a partir de este momento y para el resto de la memoria que utilizamos la formulación que permite encontrar descripciones discriminantes, a menos que explícitamente se diga lo contrario.

2.2.5 El algoritmo genético de SLAVE

Hasta el momento hemos descrito el criterio que utiliza SLAVE para determinar la mejor solución parcial de segundo orden. Como dijimos anteriormente, SLAVE es un sistema de aprendizaje basado en el modelo iterativo genético. Así, partiendo del esquema 2.2 usaremos un algoritmo genético en el paso 2.1. que nos permita encontrar una descripción que sea solución parcial de segundo orden. Por tanto, la arquitectura básica de SLAVE es la que se muestra en el esquema 2.3.

Vamos ahora a detallar cada una de las componentes básicas del algoritmo genético del apartado 2.1.:

- Representación de los elementos de la población.
- Generación de la población inicial.
- Función de evaluación.
- Los operadores genéticos.
- Condición de terminación.

2.2.5.1 Representación de un elemento de la población

Como se comento en el primer capítulo, la representación de la población en los algoritmos genéticos es un elemento clave en la operatividad del mismo. En nuestro caso, si la base de datos contiene n variables antecedente $X_1, \ldots, X_n$ y cada una de ellas tiene un dominio difuso asociado D_i con m_i etiquetas, entonces usamos el siguiente metodo de codificación

Esquema 2.3 *Arquitectura de SLAVE*

0. Sean $X_1, X_2, \ldots, X_n$ las variables predictivas que son incluidas para aprender el comportamiento de un determinado sistema. Sea Y la variable de clasificación definida sobre un dominio discreto de m conceptos. Además, sea E un conjunto con k ejemplos que describen el comportamiento en el pasado de dicho sistema. Sean $GLOBAL$ y $PARCIAL$ dos conjuntos de reglas que inicialmente estan vacíos.

1. Seleccionar un concepto B que aún no se ha aprendido.

2. Mientras $PARCIAL$ no represente suficientemente el concepto B hacer:

   2.1. Utilizar un algoritmo genético para encontrar una regla $R_B(d)$ que satisfaga que
   $$\rho(R_B(d)) = EFL(\rho(R_B(A)))$$
   sobre un subconjunto de los ejemplos de entrenamiento E, y $A \subseteq D$.

   2.2. Añadir d al conjunto de descripciones $PARCIAL$.
   $$PARCIAL = PARCIAL \bigcup d$$

   2.3 Penalizar las descripciones de $PARCIAL$ para que no vuelvan a ser selecionadas.

3. Añadir las nuevas descripciones al conjunto $GLOBAL$.
$$GLOBAL = GLOBAL \bigcup PARCIAL$$

4. Eliminar las descripciones de $PARCIAL$; $PARCIAL = \emptyset$.

5. Si existen más conceptos a aprender, volver al paso 1., en otro caso devolver $GLOBAL$ como solución al problema de aprendizaje.

binaria para los elementos de Δ como un vector de $m_1+\ldots+m_n$ de componentes cero-uno, tal que

$$componente(m_1+\ldots+m_{r-1}+s) = \begin{cases} 1 & \text{si el s-ésimo elemento del dominio } D_r \\ & \text{es un valor de la variable } X_r \\ 0 & \text{en otro caso} \end{cases}$$

$\boxed{\textbf{Ejemplo}}$ **2.5** *Supongamos las siguientes tres variable X_1, X_2 y X_3, tal que sus dominios difusos asociados son los mostrados en la figura 2.3. En este caso, el vector 1010010111 representa el siguiente antecedente*

$$X_1 \ es \ \{A_{11}, A_{13}\} \ y \ X_2 \ es \ \{A_{23}, A_{25}\} \ y \ X_3 \ es \ \{A_{31}, A_{32}\}.$$

Como X_3 toma todos los valores de su dominio D_3, el antecedente anterior es equivalente a

$$X_1 \ es \ \{A_{11}, A_{13}\} \ y \ X_2 \ es \ \{A_{23}, A_{25}\}.$$

Este método de codificación binaria, donde la información representa la inclusión o no de los valores de los dominios de las variables, se utilizó en la primera versión de SLAVE [51] y se ha empleado en diversos sistemas de aprendizaje que trabajan con algoritmos genéticos. Por ejemplo, [157] utiliza la misma codificación de los elementos de la población, y en los sistemas GABIL [31] y GIL [85] una codificación semejante aunque adaptada al hecho de ser sistemas que trabajan con el modelo Pittsburgh.

Por otro lado, este tipo de codificación ha sido ampliamente utilizado por este tipo de sistemas de aprendizaje, ya que esta codificación es una forma simple y natural de recoger el lenguaje DNF modificado, que es el que suelen utilizar la mayoría de estos sistemas.

2.2.5.2 Generación de la población inicial

La población inicial es el punto de partida del proceso evolutivo. En el capítulo primero destacamos que la forma más habitual de generar esta primera población consiste en la construcción completamente aleatoria de los individuos. Sin embargo, desde el punto de vista de los algoritmos de búsqueda, cuanto más cercano se encuentre el punto de partida de las soluciones a alcanzar, menor debe ser el costo para encontrar dichas soluciones. En este sentido, en el algoritmo genético de SLAVE, la generación de la población inicial consiste en seleccionar ejemplos de forma aleatoria del conjunto de ejemplos de la clase que ha de ser aprendida y obtener para cada uno de ellos el antecedente más específico que mejor lo recubra, es decir, cada variable tomará como valor solamente una etiqueta, y esta etiqueta será aquella que presenta mayor grado de pertenencia con su correspondiente componente en el ejemplo. A continuación mostramos un ejemplo para construir en base a un ejemplo el antecedente más específico que mejor lo recubre.

Ejemplo **2.6** *Sean X_1, X_2 y X_3 variables cuyos dominios difusos asociados son los descritos en la figura 2.3, y sea (r1,r2,r3) un ejemplo seleccionado de forma aleatoria del conjunto de ejemplos para una determinada clase. El antecedente más específico que mejor lo representa es:*

$$X_1 \text{ es } A_{13} \text{ and } X_2 \text{ es } A_{23} \text{ and } X_3 \text{ es } A_{21}$$

cuyo representación binaria es 0010010010.

Cuando existen varias posibilidades, se selecciona aleatoriamente una de ellas.

Esta selección de la población inicial puede considerarse similar a la usada por los algoritmos de la familia AQ [109, 114], ya que para la generación del antecedente inicial para una clase, tomamos ejemplos de esta clase en el conjunto de entrenamiento como punto de partida, y el algoritmo genético puede considerarse como un proceso de generalización sobre los antecedentes seleccionados.

2.2.5.3 Función de evaluación (FITNESS)

El algoritmo genético de SLAVE presenta como entradas, un conjunto de ejemplos de entrenamiento, un conjunto de variables antecedente y una clase fija de la variable de clasificación. El objetivo del algoritmo genético consisten en encontrar la mejor solución parcial de segundo orden para esta clase, es decir, la mejor asignación de valores a las variables que permiten describir a dicha clase. Anteriormente, se han definido los criterios para valorar la bondad de las descripciones, que vienen determinadas por la aplicación de una evaluación funcional lexicográfica sobre la consistencia, completitud, simplicidad y comprensibilidad de las mismas.

Por tanto, la valoración de una descripción A se define como:

$$fitness(R_B(A)) = \rho(R_B(A))$$

sobre el conjunto de ejemplos de entrenamiento y el objetivo del algoritmo genético consiste en

$$EFL(fitness(R_B(A)))$$

con $A \in D$.

2.2.5.4 Los operadores genéticos

Para generar las nuevas poblaciones se usan un conjunto de operadores genéticos, que podemos clasificar en dos tipos: operadores genéticos básicos y operadores genéticos definidos para el problema del aprendizaje. A continuación pasamos a describir cada uno de ellos.

Operados genéticos básicos

- **Operador de selección lineal con elitismo**. Este es un modelo de selección que ordena los elementos de la población atendiendo a su valoración y asigna una probabilidad de selección a cada posición de la población. La selección de los individuos que pasarán a formar parte de la siguiente población se realiza mediante un muestro aleatorio simple. A este operador se le asocia la estrategia adicional que se conoce por elitismo ([29]), el cual asegura que el mejor individuo de una generación estará presente en la siguiente.

- **Operador de mutación.** Este es el clásico operador de mutación que cambia un gen de un elemento en la población con una cierta probabilidad.

- **Operador de cruce sobre dos puntos**. Este tipo de cruce fija dos puntos de corte entre dos elementos de la población e intercambia el segmento central como muestra la figura 2.4. Este operador de cruce es una variante del operador de cruce simple ([48, 72]).

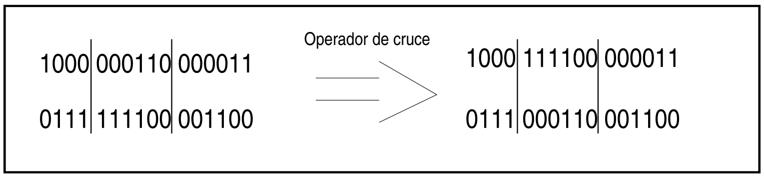


Figura 2.4: Operador de cruce sobre dos puntos

Operadores genéticos especificos para este problema

- **Operador de rotación**([55]). Este operador es una versión modificada del tradicional operador de inversión que provoca el incremento del nivel de diversidad en la población. Este operador toma un punto de corte en un elemento de la población e intercambia los dos segmentos como muestra la figura 2.5.

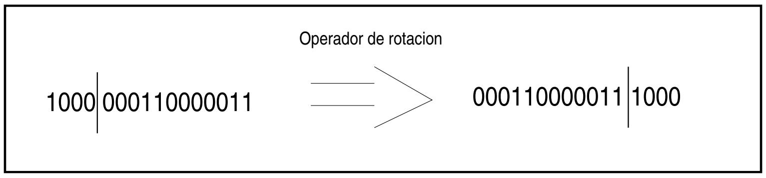


Figura 2.5: Operador de rotación

- **Operador de generalización**. Este operador trata de simplificar y aumentar la comprensibilidad de las reglas devueltas por el proceso genético. El operador generaliza las reglas de la población utilizando la misma idea que fué descrita para el criterio de comprensibilidad. Por consiguiente, este operador favorece la modificación en los individuos de la población para mejorar su comprensibilidad. Al igual que sucede con el criterio de compresibilidad, este operador afecta únicamente a aquellas variables que tiene un dominio ordenado asociado.

Previo a la descripción del operador, es necesario introducir algunos conceptos. En la definición 2.16 se establece el concepto de variable estable. Dicha definición se puede caracterizar en base a la representación genética propuesta de la siguiente manera:

Definición 2.18 *Diremos que una variable antecedente con dominio asociado ordenado es estable, si en la representación binaria de su valor aparece un única secuencia continua de unos.*

De igual forma, y por oposición a este concepto, definimos los dos siguientes conceptos:

Definición 2.19 *Diremos que una variable es inestable, si en la representación binaria de su valor existe más de una secuencia continua de unos.*

Definición 2.20 *Dada la representación genética de un valor de una variable inestable, llamaremos zona inestable a cada secuencia continuo de ceros.*

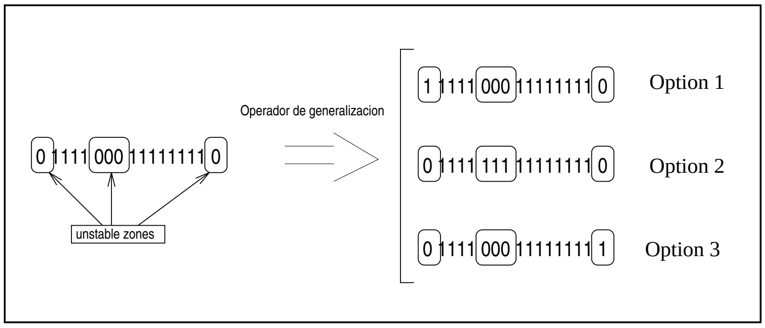


Figura 2.6: Operador de Generalización

El operador de generalización trata de obtener variables estables eliminando sus zonas inestables. Este operador se puede describir de forma algorítmica como indica el esquema 2.4.

La figura 2.6 muestra las zonas inestable del valor de una variable y además se muestran las tres posibles opciones de generalización.

- **Operadores AND y OR**. Son dos operadores de cruce especiales que tratan de representar procesos de generalización y especificación entre los individuos de la población. Al igual que en el operador de cruce definido anteriormente, estos dos operadores seleccionan dos individuos y establecen dos puntos de corte. Los segmentos centrales de ambos individuos se combinan bit a bit mediante la operación lógica AND en un caso, y mediante la operación lógica OR en el otro caso.

Esquema 2.4 *Enfoque iterativo con dos niveles*

1. Se selecciona aleatoriamente una variable con dominio asociado ordenado de un elemento de la población.
2. Si esta variable es inestable hacer lo siguiente
   - 2.1 Detectar sus zonas inestables y seleccionar aleatoriamente una de ellas.
   - 2.2 Cambiar la secuencia de ceros de la zona inestable selecionada por una secuencia de unos.
   - 2.3 Calcular la valoración del nuevo elemento usando la función de valoración. Si su valoración es igual o mejor que la del elemento original, la modificación se mantiene. En otro caso, se restablece el valor original de la variable.

2.2.5.5 Condición de terminación

La forma más habitual de establecer la condición de terminación en los algoritmos genéticos consiste en establecer un número fijo de iteraciones. En nuestro caso, con la idea de realizar búsquedas más adecuadas al tipo de descripciones que se intentan buscar hemos considerado una condición de terminación múltiple. En la implementación de esta condición distinguimos entre el caso en el que estamos aprendiendo la primera regla para una clase y el caso en que al menos ya tenemos alguna regla para describir dicha clase. En el primer caso realizamos una búsqueda más exhaustiva mientras para el segundo caso optamos por relajar el proceso de búsqueda.

El algoritmo genético termina devolviendo la mejor regla de la última población si alguna de las siguientes condiciones se verifica:

- el número de iteraciones es mayor que un límite fijo N_ITER_1,
- la valoración de la mejor regla no mejora durante un número fijo de iteraciones N_ITER_2, donde N_ITER_2 < N_ITER_1, y existen reglas obtenidas para esta clase,
- no hay todavía reglas obtenidas de esta clase, pero la función de valoración se mantiene durante un número fijo de iteraciones (N_ITER_2) y el mejor elemento de la población recubre al menos un ejemplo del conjunto de entrenamiento.

2.2.6 Penalización de reglas

Las subsecciones anteriores presentan definiciones y criterios para obtener soluciones parciales de segundo orden. Sin embargo, para que la agregación de estas soluciones pueda ser considerada como solucion parcial de primer nivel para algún concepto, es necesario establecer un criterio de penalización de las descripciones obtenidas según describe el paso 2.3. del esquema 2.3. Como en dicha sección dijimos, el criterio más extendido para penalizar reglas en este tipo de enfoques consiste en eliminar del conjunto de entrenamiento aquellos ejemplos que apoyan la veracidad de las descripciones del conjunto $PARCIAL$. La implementación de este criterio es muy fácil cuando trabajamos con ejemplos y reglas crisp, sin embargo, cuando trabajamos con ejemplos y reglas difusas, un ejemplo apoya la veracidad de la regla en un cierto grado. Por consiguiente, es necesario incluir una definición que permita determinar cuando un ejemplo apoya la veracidad de una regla en un grado suficiente, de manera que dicha regla sea útil para describir dicho ejemplo. Para definir esta extensión nos apoyamos en la siguiente definición.

Definición 2.21 *Sea E un conjunto de ejemplos y sea B un concepto. Definimos el conjunto de ejemplos de E que describen el concepto B como:*

$$E_B = \{e \in E \mid U(e, B) > 0\}$$

En base a la definición anterior podemos establecer el concepto de λ-recubrimiento

Definición 2.22 *Sea $R_B(A)$ una regla, E un conjunto de ejemplos y $\lambda \in [0, 1]$. El λ-recubrimiento de la regla $R_B(A)$ sobre el conjunto E es el λ-corte de $E^+(R_B(A))$. Llamaremos a λ parámetro de recubrimiento.*

A partir de la anterior definición, podemos determinar el conjunto de ejemplos de un determinado concepto que aún permanecen en el conjunto de entrenamiento, y por tanto, a partir de este concepto fijar un criterio de penalización de reglas.

Dado un conjunto $PARCIAL$=$\{R_B(A^1), R_B(A^2), \ldots, R_B(A^j)\}$ de reglas ya aprendidas para el concepto B y un conjunto de ejemplos E, definimos E_B^j como el subconjunto de ejemplos que describen al concepto B sin considerar los ejemplos que son cubiertos por las reglas existentes en el conjunto $PARCIAL$, es decir,

$$E_B^j = \{e \in E_B \mid e \notin \bigcup_{i=1}^{j} E_\lambda^+(R_B(A^i))\}$$

Siguiendo este criterio de penalización de reglas, comprobar si un conjunto de descripciones constituye una solución parcial de primer orden es tan simple como comprobar que no existe ningún ejemplo del concepto que se trata de aprender en el conjunto de entrenamiento, despues de haber eliminado todos los ejemplos cubiertos por las descripciones ya obtenidas.

Definición 2.23 *Dado un conjunto $PARCIAL$=$\{R_B(A^1), R_B(A^2), \ldots, R_B(A^j)\}$ de reglas ya aprendidas para el concepto B y un conjunto de ejemplos E, diremos que el conjunto $PARCIAL$ verifica la condición de completitud para el concepto B, sii*

$$|E_B^j| = 0$$

La verificación de esta propiedad por un conjunto de descripciones para un ejemplo implica que dicho conjunto se puede considerar como una solución parcial de primer orden. Pero realmente, la condición anterior solamente es aplicable cuando el conjunto de ejemplos no contiene ni ruido ni inconsistencias. Por consiguiente, se hace necesario, también en este caso, relajar esta condición. A continuación se propone una nueva condición relajada sustituyendo la condición de completitud, cuya verificación indica que el concepto esta suficientemente representado por las descripciones ya obtenida. Esta nueva condición tiene en cuenta dos consideraciones:

1. Todos los ejemplos de un determinado concepto están cubiertos de forma adecuada por alguna de las descripciones obtenidas.
2. Existen ejemplos que no son cubiertos adecuadamente por las descripciones obtenidas, pero no es posible encontrar una nueva descripción útil.

Más formalmente, podemos definir la nueva condición de la siguiente manera:

Definición 2.24 *Dado un conjunto* $PARCIAL$=$\{R_B(A^1), R_B(A^2), \ldots, R_B(A^j)\}$ *de reglas ya aprendidas para el concepto* B*, y dada una nueva regla* $R_B(A^{j+1})$*, diremos que el conjunto* $PARCIAL$ *representa suficientemente el concepto* B*, si verifica al menos una de las siguientes condiciones:*

a) $|E_B^j| = 0$

b) Si al añadir $R_B(A^{j+1})$ *al conjunto* $PARCIAL$ *se satisface que:*

$$|E_B^j| - |E_B^{j+1}| = 0.$$

En este caso $R_B(A^{j+1})$ *no se incluye en el conjunto* $PARCIAL$*.*

La primera condición determina si el conjunto de reglas obtenido hasta el momento, $PARCIAL$, cumple la condición de completitud. La segunda condición comprueba si la última regla que se ha obtenido permite representar algún ejemplo que no sea cubierto por alguna de las reglas anteriores. Esta comprobación se realiza comparando el número de ejemplos que quedaban por representar, y los que después de incluir la regla aún quedan. Para que esta segunda condición cumpla su cometido correctamente, es necesario que la descripción obtenida sea en realidad la mejor solución parcial de segundo orden que se puede obtener sobre el conjunto de ejemplos que aún permanecen en el conjunto de entrenamiento.

En conclusión, un conjunto de reglas que describen un determinado concepto se considerada como una solución parcial de primer orden si satisface la anterior definición, y por tanto, el paso 2 del esquema 2.3 utiliza esta definición para comprobar si son necesarias más reglas para describir este concepto.

2.3 Mecanismo de resolución de conflictos

El mecanismo de resolución de conflictos trata de dar respuesta a las dificultades que se le presentan a un modelo de inferencia en el caso crisp, para tomar una decisión. Estas dificultades aparecen porque normalmente el algoritmo de aprendizaje extrae conocimiento sobre el comportamiento del sistema, pero no extrae información que le permita determinar como utilizar dicho conocimiento. Este metaconocimiento es importante, ya que permite comprender mejor el sistema que se ha aprendido.

Cuando trabajamos con reglas, es posible que dos o más reglas que describen diferentes conceptos sean aplicables ante una misma entrada al sistema. En este caso, es necesario establecer una norma de actuación para determinar cuál de las posibles reglas es la más adecuada.

Nosotros establecemos un mecanismo de resolución de conflictos basado en la confianza que las reglas nos ofrecen. Esta confianza se establece en función del comportamiento que las mismas presentaban sobre el conjunto de entrenamiento. En este sentido, proponemos una evaluación funcional lexicográfica para la resolución de los conflictos en base a los siguientes criterios:

Criterio 1: Adaptación entre el ejemplo y la regla.

Criterio 2: Confianza que la regla ofreció durante el proceso de aprendizaje.

Criterio 3: Simplicidad de la regla.

En cuanto al primer criterio, este viene determinado por la valoración obtenida de la adaptación entre ejemplo y regla. Con respecto al segundo criterio, la confianza de una regla debe estar relacionada con la capacidad de predicción de la misma. Dicha capacidad se puede medir en función de su comportamiento en la fase de aprendizaje. Con este fin, definimos el concepto de peso de una regla como la relación entre los ejemplos que representó correctamente sobre el conjunto de entrenamiento y todos los ejemplos cubiertos por ella.

Definición 2.25 *Sea E un conjunto de ejemplos y sea $R_B(A)$ una regla. Definimos el peso de la regla $R_B(A)$ como:*

$$w(R_B(A)) = \frac{n^+(R_B(A))}{n(R_B(A))}$$

El tercer criterio, mide la simplicidad de la reglas. Como se ha definido anteriormente en este capítulo, la simplicidad esta relacionada con el número de condiciones que un ejemplo debe verificar para respaldar la veracidad de una regla. Desde un punto de vista objetivo, parece sensato que si dos reglas son aplicables a una misma entrada, y no hay posiblidad de establecer cual de ellas es más adecuada en función de las medidas anteriores, se seleccione aquella que necesita verificar más condiciones. Por tanto, en este último criterio, se considerá la más adecuada, la regla con menor simplicidad entre las aplicables.

2.4 Pruebas experimentales

En esta sección estudiaremos el comportamiento del sistema de aprendizaje SLAVE y la posibilidad de incorporar algunas modificaciones en su implementación. Para este estudio, hemos seleccionado una amplia serie de bases de datos que se describen en el apéndice A. Estas bases de datos pueden considerarse como una extensa representación de las diferentes situaciones con las que se puede encontrar un algoritmo de aprendizaje inductivo, tales como valores perdidos, ruido e inconsistencias, un número elevado de variables antecedente, variable de clasificación crisp y variable de clasificación difusa.

Con el fin de establecer el comportamiento de SLAVE, sus resultados se comparan con los obtenidos por otros algoritmos de aprendizaje que se describen en el apéndice B. Cada uno de estos algoritmos se puede considerar como un representante de diferentes filosofías de aprendizaje.

La estrategia utilizada para la realización de la experimentación es la siguiente: cada base de datos esta dividida en una serie de particiones, donde el número de las mismas viene determinado por las siguientes características:

- Si la base de datos se ha obtenido en un único fichero que contiene toda la información, nosotros hemos procedido a realizar cinco particiones de la misma en dos ficheros de entrenamiento y prueba, donde cada fichero de entrenamiento contiene el 70 % de los ejemplos del fichero original, y el fichero de prueba contiene el 30 % restante.
- Si la base de datos contenía particiones de entrenamiento y prueba que han sido utilizadas por algunos autores para comprobar empíricamente el comportamiento de sus sistemas, hemos conservado para nuestras pruebas dichas particiones.

Sobre cada prueba se han extraído los siguientes parámetros: porcentaje de acierto sobre la partición de entrenamiento y la partición de prueba, número de reglas y tiempo de ejecución en segundos. Los resultados considerados para cada partición son la media sobre cada uno de estos parámetros sobre cinco ejecuciones. En las tablas, se representará para cada base de datos, la media obtenida sobre cada una de las particiones. Por consiguiente, el significado de cada valor de la tabla será el anteriormente definido, a menos que se diga explícitamente otra cosa.

Parámetro	Valor
Número máximo de iteraciones	300
Número máximo de iteraciones sin mejora	100
Número de individuos en la población	20
Probabilidad de mutación	0.01
Probabilidad de cruce	0.1
Parámetro de consistencia k_1	0
Parámetro de consistencia k_2	1
Parámetro de recubrimiento λ	0.8

Tabla 2.2: Parámetros fijados para la ejecución de SLAVE

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.63	95.01	5.6	85.2
MONK1	96.13	95.0	6	41
MONK2	90.06	69.95	12.2	97
MONK3	98.20	89.16	7.4	47
INFARTO	98.29	79.68	7.68	252.4
WINE	99.61	95.54	8.2	839.8
PIMA	79.89	79.99	20.8	1818
IONOSPHERE	99.65	81.52	44.6	15112
SOYBEAN	94.42	93.34	41	12314

Tabla 2.3: Resultados de la propuesta original para caso crisp

En este primer capítulo, estudiaremos experimentalmente los siguiente puntos:

1. Los resultados que proporciona SLAVE considerando únicamente como operadores géneticos el cruce sobre dos puntos y la mutación, comparando estos resultados con los obtenidos por los otros algoritmos de aprendizaje propuestos.

2. Las ventajas que aporta a SLAVE la inclusión de los otros operadores genéticos.

2.4.1 Propuesta original

Con la estrategia de experimentación anteriormente descrita, estudiamos el comportamiento de SLAVE considerando que el algoritmo genético encargado de extraer las reglas únicamente utiliza los operadores de cruce y mutación para evolucionar la población hacia la solución final. La tabla 2.2 muestra los parámetros fijados para la ejecución del algoritmo de aprendizaje, mientras que en la tabla 2.3 se muestran los resultados obtenidos para las diferentes bases de datos con consecuente crisp usando los anteriores parámetros.

	C4.5	CN2		LVQ
Bases de datos	Prueba	Prueba	Reglas	Prueba
IRIS	92.7	94.16	6.4	95.72
MONK1	75.6	100	9	76.62
MONK2	64.3	84.0	43	67.59
MONK3	92.6	91.7	15	85.19
INFARTO	81.4	76.16	5.8	56.41
WINE	93.2	89.76	8.8	72.60
PIMA	67.7	74.5	38	67.71
IONOSPHERE	86.5	83.96	16.4	87.97
SOYBEAN	97.9	98.6	32	91.75

Tabla 2.4: Resultados obtenidos con C4.5, CN2 y LVQ

Con el fin de establecer la bondad del algoritmo propuesto, comparamos estos resultados con los obtenidos con otros algoritmos de aprendizaje. Para realizar esta comparación en el caso crisp, hemos seleccionado tres algoritmos de aprendizaje muy conocidos y referenciados en la literatura para el problema de la clasificación. Estos algoritmos son C4.5, CN2 y LVQ cuyas características de implementación se comentan en el apéndice B. Para la experimentación, hemos ejecutado cada uno de estos algoritmos, usando los parámetros por defecto que indican sus implementaciones, sobre las mismas particiones definidas para SLAVE tomando de cada uno de ellos la media sobre su porcentaje de acierto en la predicción. Además, en CN2 que es un algoritmo de aprendizaje que expresa el conocimiento adquirido en forma de reglas, hemos añadido como parámetro de comparación adicional el número medio de las mismas que son necesarias sobre cada base de datos. En la tabla 2.4 se muestran estos resultados.

Los resultados reflejan que SLAVE obtiene el mejor resultado únicamente en dos de las nueve bases de datos propuestas (WINE y PIMA), aunque por el contrario, SLAVE únicamente obtiene el peor resultado de los cuatro algoritmos para la base de datos IONOSPHERE (figura 2.7). Este hecho pone de manifiesto que el modelo de regla difusa descrito en este capítulo, es un forma válida de representación del conocimiento que presenta un correcto funcionamiento en los problemas de clasificación comparable al de otros algoritmos de aprendizaje. Además, esta forma de representación permite que el conocimiento extraido sea más fácil de entender desde un punto de vista humano. Así, por ejemplo en una ejecución de C4.5 para la base de datos WINE obtenemos el siguiente árbol:

```
Att7 <= 0.96 : 3 (27.0)
Att7 > 0.96 :
|   Att13 <= 750 :
|   |   Att11 <= 0.68 : 3 (6.0)
|   |   Att11 > 0.68 :
```

```
|   |   |   Att7 <= 1.22 : 2 (2.0/1.0)
|   |   |   Att7 > 1.22 :
|   |   |   |   Att1 <= 13.17 : 2 (47.0)
|   |   |   |   Att1 > 13.17 :
|   |   |   |   |   Att1 <= 13.29 : 1 (2.0)
|   |   |   |   |   Att1 > 13.29 : 2 (4.0)
|   Att13 > 750 :
|   |   Att10 <= 3.4 : 2 (4.0)
|   |   Att10 > 3.4 : 1 (37.0)
```

Para la misma partición de esta base de datos, las reglas obtenidas por SLAVE son las siguientes:

```
 SI
      Att5 es {Muy Pequeno, Pequeno}
      Att13 es {Medio}
   ENTONCES
      Clase es Tipo1
---------------------------------------

 SI
      Att1 es Mayor o Igual que {Medio}
      Att5 es Menor o Igual que {Medio}
      Att7 es Mayor o Igual que {Pequeno}
   ENTONCES
      Clase es Tipo1
---------------------------------------

 SI
      Att10 es {Ex. Pequeno, Muy Pequeno}
   ENTONCES
      Clase es Tipo2
---------------------------------------

 SI
      Att1 es Menor o Igual que {Medio}
      Att2 es {Medio}
      Att10 es Menor o Igual que {Pequeno}
   ENTONCES
      Clase es Tipo2
---------------------------------------
```

```
SI
      Att11 es Menor o Igual que {Pequeno}
  ENTONCES
      Clase es Tipo3
```

Tomando ambas representaciones del conocimiento sobre un mismo sistema, normalmente la transmisión de dicho conocimiento entre dos personas esta mucho más cerca de la que proporciona SLAVE a la que proporciona C4.5.

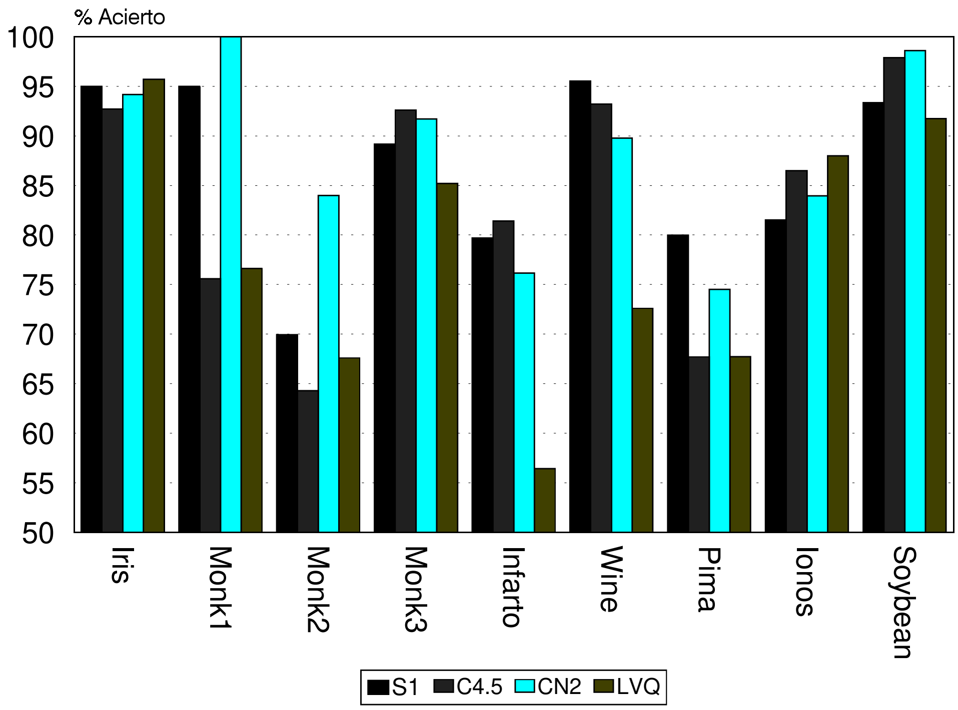


Figura 2.7: Comparativa del acierto en los conjuntos de prueba (1)

Centrándonos de nuevo en los resultados el número medio de reglas que obtiene SLAVE en cada base de datos es semejante al que proporciona CN2. Sin embargo, podemos detectar que para la base de datos IONOSPHERE el número de reglas que produce SLAVE es extremadamente alto comparandolo con el número de reglas de CN2. Esto indica que SLAVE con esta definición tiene un comportamiento aceptable con bases de datos que contienen un número no excesivamente alto de variables de entrada, y que dicho comportamiento empeora ante la presencia de un número elevado de las mismas.

También estamos interesados en comprobar el comportamiento del algoritmo propuesto ante bases de datos con consecuente difuso. Así, utilizando los mismos parámetros

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
SIN	0.0026	0.0015	6	51
PARABOLA	0.0057	0.0016	2	34
3DPARABOLA	0.0066	0.0112	3	1676
MONTE	0.0038	0.0038	3	2408
PENDULO	22.5634	7.1376	6	83
RICE	0.1354	0.092	5	80.2

Tabla 2.5: Resultados de SLAVE para el caso difuso

Bases de datos	Entrenamiento	Prueba	N.Reglas
SIN	0.0038	0.0412	7
PARABOLA	0.0040	0.0021	7
3DPARABOLA	0.0032	0.0055	49
MONTE	0.0028	0.0024	49
PENDULO	1.2044	7.6490	13
RICE	0.0515	0.0604	43

Tabla 2.6: Resultados obtenidos con el algoritmo WM

definidos para la experimentación anterior, obtenemos los resultados que se muestran en la tabla 2.5.

Al igual que hicimos anteriormente para determinar la bondad del comportamiento de SLAVE en el caso crisp, lo realizaremos en el caso difuso, comparando SLAVE con algún algoritmo de aprendizaje para este tipo de problemas. Para ello seleccionamos el algoritmo de aprendizaje definido en [159] que denominaremos algoritmo WM. Para la ejecución de la parte experimental con WM asumiremos que las variables de entrada y de salida tienen la misma discretización que fué definida para la experimentación con SLAVE. Los resultados obtenidos con este algoritmo se presentan en la tabla 2.6.

A diferencia del caso crisp, en los resultados obtenidos por SLAVE en el caso difuso son comparativamente menos competitivos con los algoritmos especialmente diseñados para abordar estos problemas como es el algoritmo WM. A pesar de esto, podemos observar que el resultado de SLAVE para las bases de datos RICE y SIN es sensiblemente mejor que el obtenido por WM.

Un dato a destacar en esta comparación es el reducido número de reglas que necesita SLAVE para representar las distintas bases de datos que contrasta con el número de las mismas necesitado por WM. Además, se puede observar que esta diferencia entre los números de reglas aumenta en gran medida cuantas más variables de entrada se encuentren involucradas en la base de datos. Esta diferencia es debida a que ambos algoritmos utilizan diferentes modelos de reglas difusas, de manera, que el modelo de regla propuesto por SLAVE puede incluir un conjunto de reglas de modelo definido en el otro algoritmo. Este

Parámetro	Valor
Probabilidad del operador de rotación	0.1
Probabilidad del operador de generalización	0.1
Probabilidad del operador AND	0.05
Probabilidad del operador ORD	0.1

Tabla 2.7: Parámetros adicionales fijados para la ejecución de SLAVE

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	96.56	95.40	4.6	78
MONK1	96.77	94.45	6	42
MONK2	91.25	70.37	13.8	119
MONK3	97.21	92.64	6.8	44
INFARTO	98.39	79.98	7.2	241
WINE	99.56	95.93	8	809.2
PIMA	79.37	80.94	23	1757
IONOSPHERE	99.65	81.90	39.2	6064
SAT	96.26	94.15	40	11372

Tabla 2.8: Resultados de SLAVE caso crisp incluyendo los nuevos operadores genéticos

hecho, al igual que sucede en el caso crisp, permite una mayor compresión del conocimiento adquirido.

2.4.2 Propuesta original con los nuevos operadores genéticos

En esta segunda parte de la experimentación estamos interesados en estudiar que aporta la inclusión de los nuevos operadores genéticos definidos anteriormente en este capítulo. Con tal fin repetimos la experimentación anterior fijando como parámetros de SLAVE, además de los propuestos en la tabla 2.2 los que muestra la tabla 2.7.

Con esta configuración de partida, la tabla 2.9 muestra los resultados obtenidos para las bases de datos del caso difuso, mientras que la tabla 2.8 muestra los resultados con las bases de datos del caso crisp.

En ambos casos podemos observar que en general se produce una ligera mejora de los resultados obtenidos por SLAVE, siendo estos más acusados en el acierto en la partición de prueba y en la reducción del tiempo necesario para la extracción del conocimiento. Esta mejora en los resultados se debe a que la inclusión de los operadores genéticos provoca un comportamiento más estable del algoritmo de aprendizaje. En la figura 2.8, donde **S2** denota los resultados obtenidos por SLAVE con los nuevos operadores genéticos y **S1** los resultados obtenidos con la implementación anterior, podemos observar que SLAVE mantiene el mejor resultado sobre las bases de datos WINE y PIMA, y obtiene el mejor resultado en la base de datos MONK3.

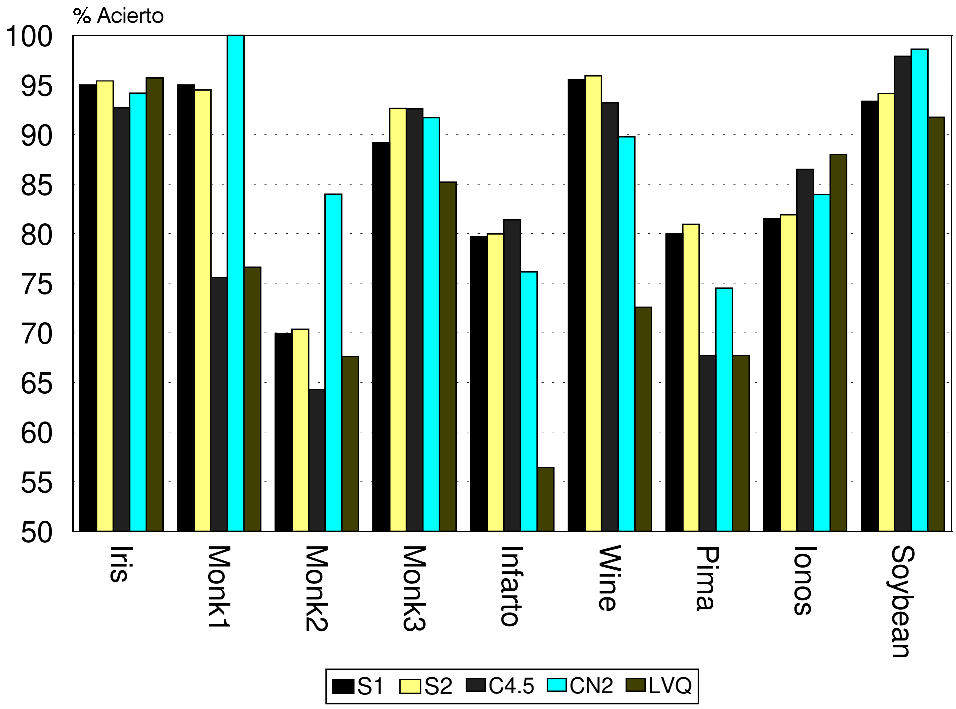


Figura 2.8: Comparativa del acierto en los conjuntos de prueba (2)

A la vista de los resultados, el dato que más resalta, es la gran reducción en el tiempo de ejecución obtenido para la base de datos IONOSPHERE, lo que pone de manifiesto que la aportación que pueden proporcionar estos operadores es mayor cuantas más variables de entrada se encuentren involucradas en el problema.

Por último, podemos extraer las siguientes conclusiones sobre esta parte experimental:

- Los resultados obtenidos por SLAVE son similares a los que producen otros algoritmos de aprendizaje, aunque en términos generales son ligeramente peores.
- La inclusión de los nuevos operadores genéticos mejoran ligeramente los resultados anteriores en términos de capacidad de predicción, y notablemente en cuanto al número de reglas y al tiempo de respuesta.

En el siguiente capítulo proponemos nuevas vías para mejorar el comportamiento del algoritmo de aprendizaje SLAVE.

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
SIN	0.0021	0.0015	6	51
PARABOLA	0.0051	0.0043	2	34
3DPARABOLA	0.0064	0.0112	3	1676
MONTE	0.0038	0.0041	3	2408
PENDULO	5.0690	6.3751	5	83
RICE	0.315	0.0105	6.2	80.2

Tabla 2.9: Resultados de SLAVE para el caso difuso incluyendo los nuevos operadores genéticos

Capítulo 3

Colaboración entre las soluciones parciales

3.1 Inconvenientes del modelo iterativo

El modelo iterativo descompone el problema en tantos subproblemas como conceptos se desean aprender, de manera, que la solución global se construye como la agregación de las descripciones obtenidas para cada concepto. Para que esta forma de construir la solución global proporcione una descripción correcta de los conceptos que se desean aprender, es necesario que no existan interacciones entre los distintos conceptos, es decir, las descripciones de los conceptos deben ser independientes entre sí.

La agregación de las soluciones parciales proporciona una buena solución global cuando cada solución parcial de primer orden satisface la condición de completitud y cada solución parcial de segundo orden satisface la condición de consistencia. El objetivo de forzar la satisfacción de la condición de consistencia es para garantizar que

$$\bigcap_{i=1}^{j} A_i = \emptyset.$$

donde A_i es la descripción de la regla i-ésima. Sin embargo, como la satisfacción de esta propiedad se restringe a los ejemplos del conjunto de entrenamiento, lo único que se puede asegurar cuando se verifica la condición anterior, es que no existe un ejemplo de ninguna clase que satisfaga la descripción de ninguna regla de cualquier otra clase en el conjunto de ejemplos.

Bajo estas circunstancias, podemos asegurar que no existe interacción entre las distintas soluciones parciales sobre el conjunto de entrenamiento, y por consiguiente, la agregación de las mismas es una buena solución global.

Pero, como ya se comentó anteriormente, en problemas extraídos del mundo real, donde los conjuntos de ejemplos normalmente contienen ruido o están sujetos a descripciones imprecisas, es difícil mantener las anteriores condiciones. La no verificación de la condición de completitud y fundamentalmente la no satisfacción de la condición de consistencia en las soluciones parciales, provoca que la construcción de la solución global como agregación de estas soluciones parciales contenga una descripción imprecisa del sistema que se desea aprender, en el sentido de que varias reglas de diferentes conceptos son igualmente aplicables sobre determinadas zonas del espacio de los ejemplos, es decir, existen al menos dos reglas cuyas descripciones verifican que

$$A_1 \bigcap A_2 \neq \emptyset.$$

En estas situaciones, parece conveniente mantener la solución global más cercana a la satisfacción de las dos condiciones (consistencia y completitud), aún a costa de ser menos exigentes con su satisfacción en las soluciones parciales. Para tener en cuenta este extremo, es necesario que el algoritmo de aprendizaje tenga una visión menos local del problema y contemple desde un punto de vista más global el aprendizaje de los diferentes conceptos.

Esta visión global del problema, no sólo es útil ante la presencia de ruido e imprecisión. Aún conociendo que no existe ruido ni imprecisión en el conjunto de ejemplos, esta visión global puede en un momento dado, sacrificar la satisfacción de las dos condiciones a costa de mejorar la comprensibilidad de las descripciones parciales manteniendo lo más posible la capacidad de predicción sobre el conjunto de ejemplos.

Un ejemplo típico de esta situación se presenta cuando un concepto se puede describir como un caso particular de un concepto superior.

Ejemplo **3.1** *Supongamos que se desea clasificar un conjunto de aves en base a su capacidad de volar. Sea el conjunto de ejemplos el siguiente:*

Ave	*Concepto*
Aguila	*Vuela*
Albatros	*Vuela*
Buitre	*Vuela*
Cuervo	*Vuela*
Petirrojo	*Vuela*
Golondrina	*Vuela*
Pingüino	*No Vuela*
Gaviota	*Vuela*
Paloma	*Vuela*
Flamenco	*Vuela*

Si cada concepto se aprenden de forma independiente, se pueden obtener las dos siguientes descripciones que verifican las condiciones de consistencia y completitud:

1a) Si el ave es un Pingüino entonces No Vuela.

1b) Si el ave es un Aguila o es un Albatros o es un Buitre o es un Cuervo o es un Petirrojo o es una Golondrina o es una Gaviota o es una Paloma o es un Flamento entonces Vuela.

Sin embargo, tomando una visión más global de problema, se podría sugerir la siguiente clasificación:

2a) Si el ave es un Pingüino entonces No Vuela.

2b) Si es cualquier ave entonces Vuela.

Estas dos nuevas reglas son más faciles de entender desde un punto de vista humano, ya que se describe la situación del Pingüino, como un caso particular de ave que es incapaz de volar. Sin embargo, la descripción (2b) no satisface la condición de consistencia, y por consiguiente, vulnera ser una solución parcial de segundo orden.

Si nos detenemos a analizar el segundo conjunto de descripciones, no parece tan relevante que la descripción 2b no satisfaga la condición de consistencia si estas situaciones se tienen en cuenta durante el proceso de inferencia. Si a la entrada del sistema de inferencia se le presenta un ejemplo que es cubierto por varias descripciones de diferentes conceptos, la salida del mismo debería ser el concepto asociado a la descripción más específica de entre las descripciones aplicables.

En el caso anterior, ante el ejemplo "Pingüino", ambas descripciones (2a y 2b) son aplicables, pero la descripción 2a esta contenida en la descripción 2b, por tanto, es más específica que esta. En consecuencia, la salida diría que ese ave No Vuela.

De todas maneras, en la mayoría de las situaciones, no es fácil que el conjunto de descripciones satisfaga un orden completo con respecto a la operación $\subseteq$. Es decir, nos podemos encontrar con que para una misma entrada dos descripciones de dos conceptos diferentes son aplicables, sin que entre ellos exista un relación de contenido, aunque si de intersección no vacía.

Basándonos en esto, podemos caracterizar las reglas que pueden provocar conflictos durante el proceso de inferencia en función de las intersecciones entre las descripciones.

Definición 3.1 *Sean dos reglas $R_{B_1}(A_1)$ y $R_{B_2}(A_2)$ que describen dos conceptos diferentes B_1 y B_2. Diremos que estas dos reglas pueden entrar en conflicto durante el proceso de inferencia, si la intersección entre sus descripciones no es vacía, es decir,*

$$A_1 \bigcap A_2 \neq \emptyset$$

Además, en base al concepto de intersección entre descripciones, podemos determinar cuando una regla es una excepción a una regla más general.

Definición 3.2 *Sean dos reglas $R_{B_1}(A_1)$ y $R_{B_2}(A_2)$ que describen dos conceptos diferentes. Diremos que la regla $R_{B_2}(A_2)$ es una excepción a la regla $R_{B_1}(A_1)$ si y sólo si,*

$$A_1 \bigcap A_2 = A_2$$

Además, en estas circunstancias diremos que la regla $R_{B_1}(A_1)$ es una regla más general que $R_{B_2}(A_2)$ o que la regla $R_{B_2}(A_2)$ es una regla más específica que $R_{B_1}(A_1)$.

Si durante el proceso de inferencia entran en conflicto reglas que satisfacen la anterior definición, es decir, existe una regla que es más específica que el resto de las reglas aplicables, parece que la entrada propuesta satisface una excepción a las reglas más generales, y por tanto, la salida del proceso de inferencia debe coincidir con el concepto asociado a esta regla más específica.

Como ya se describió en el capítulo anterior, el mecanismo de resolución de conflictos definido en SLAVE, toma como primer criterio el peso asociado a las reglas. El objetivo que se persigue con este criterio, es dar prioridad a las reglas que presentaron una mayor capacidad de predicción sobre el conjunto de entrenamiento.

En base a lo anterior, una propiedad deseable que debería cumplir el sistema para garantizar un correcto funcionamiento, es que si existen dos reglas para dos conceptos diferentes, y una de ellas es una excepción a la otra, entonces la regla más específica debe tener un peso mayor que la más general. Sin embargo, esta propiedad no la satisface SLAVE.

Ejemplo 3.2 *Supongamos que el algoritmo de aprendizaje devuelve un conjunto de reglas para describir el comportamiento de un determinado sistema. Supongamos además que $R_{B_1}(A_1)$ y $R_{B_2}(A_2)$ son dos de las reglas extraídas por el algoritmo para dos conceptos distintos que verifican que:*

- $A_1 \bigcap A_2 = A_1$, *es decir, la regla $R_{B_1}(A_1)$ es una excepción a la regla $R_{B_2}(A_2)$*
- $\omega(R_{B_2}(A_2)) > \omega(R_{B_1}(A_1))$ *y por consiguiente el peso asociado a la regla más general es mayor que el asociado a la regla más específica.*

Bajo estas condiciones, todo ejemplo que sea cubierto por la regla $R_{B_1}(A_1)$, lo es también por la regla $R_{B_2}(A_2)$. Al tener la regla más general mayor peso que la más específica, en caso de conflicto siempre se seleccionará la regla $R_{B_2}(A_2)$, por lo que podemos asegurar que la regla $R_{B_1}(A_1)$ no será seleccionada nunca durante el proceso de predicción. Por tanto, dicha regla no es útil para describir el sistema y puede ser eliminada del conjunto de reglas.

Podría parecer que el problema que pone de manifiesto el ejemplo anterior parte de una inadecuada definición de los criterios establecidos en el mecanismo de resolución de conflictos. Sin embargo, este problema plantea un error en la propia concepción del algoritmo de aprendizaje. Desde el punto de vista del aprendizaje, no parece lógico que la excepción a una regla tenga menor capacidad de predicción que la regla más general, por que si esto ocurre, es que la excepción planteada no es cierta.

El aprendizaje individual y aislado de cada una de las reglas que son necesarias para describir el sistema es el desencadenante de este tipo de problemas. En consecuencia, para que este tipo de inconvenientes no aparezcan en los algoritmos de extracción automática de conocimiento, es necesario establecer relaciones de colaboración-competición entre las reglas que componen dicho conocimiento.

Por consiguiente, si se desea que se establezcan relaciones de colaboración-competición entre las reglas aprendidas es necesario tener una visión más global del problema. Obviamente, esta concepción más global del problema del aprendizaje esta intimamente relacionada con el conocimiento de como las reglas se comportarán durante el proceso de predicción, es decir, con el conocimiento del uso de las reglas aprendidas.

De entre las formas posibles de dotar a los algoritmos basados en el modelo iterativo de esta capacidad de visión más global del problema, en este capítulo proponemos dos posibles vías para solucionarlo: la primera consiste en añadir al algoritmo de aprendizaje un módulo de refinamiento de teorías como fase final del proceso de aprendizaje. La entradas a este módulo de refinamiento son el conjunto de ejemplos y el conjunto de reglas aprendidas. El objetivo del refinamiento consiste en mejorar el conocimiento aprendido en varios sentidos: capacidad de predicción, comprensibilidad de las descripciones, reducción de las descripciones, etc.

La segunda vía consiste en modificar el algoritmo de aprendizaje para establecer relaciones de colaboración-competición entre las soluciones parciales. La modificación propuesta consiste básicamente en incorporar la información que se conoce hasta un momento dado, del comportamiento que tendrán las reglas durante el proceso de predicción para evaluar la bondad de las siguientes reglas a aprender.

En las dos siguientes secciones se estudia más detenidamente cada una de estas alternativas.

3.2 Algoritmo de refinamiento de teorías difusas

Dado un conocimento incompleto y/o incorrecto (teoría) que trata de describir el comportamiento de un determinado sistema, un algoritmo de refinamiento de teorías trata de modificar este conocimiento inicial para producir un mejor funcionamiento del sistema.

En nuestro caso, el conocimiento inicial sobre el sistema se describe como una base de reglas difusas del tipo DNF (teoría difusa) y se determina la mejora de dicho conocimiento utilizando un conjunto de ejemplos del sistema que se desea aprender. El objetivo del algoritmo de refinamiento consiste en modificar el conjunto inicial de reglas para obtener una mejor adaptación sobre el conjunto de ejemplos, una mayor comprensibilidad de las descripciones y una reducción, si es posible, del número de reglas.

Los operaciones básicas de este tipo de algoritmos, cuando el conocimiento se representa mediante reglas, son los procesos de especificación, generalización, adición y eliminación de reglas. En consecuencia, la principal diferencia entre dos algoritmos de refinamiento de reglas es la heurística usada por el algoritmo para aplicar estos procesos. Por ejemplo, el algoritmo EITHER [120] mejora el conjunto de reglas causando el mínimo número de cambios sobre el conocimiento inicial. Sin embargo, podemos encontrar otros algoritmos como JoJo [37] donde su heurística solamente se guía por la precisión del conjunto de reglas sobre el conjunto de ejemplos. Estos dos sistemas se diseñaron para trabajar con reglas crisp.

Por las razones anteriomente expuestas, en [55] se propone un algoritmo de refinamiento, que asociado a SLAVE, trata de verificar que la unión de las reglas individuales puede considerarse como una buena solución global. La asociación de estos dos algoritmos recibe el nombre de SLAVE+R, y presenta una estructura semejante a la propuesta en la figura 3.1. En trabajos posteriores, tales como [57, 58], se modifica la la definición original de SLAVE+R, proponiéndose dos versiones diferentes de dicho algoritmo de refinamiento, uno para el caso difuso y otro para el caso crisp. Además, se estudia el comportamiento del algoritmo de refinamiento de teorías propuesto con el conocimiento extraído por otros algoritmos de aprendizaje.

Los elementos básicos del algoritmo de refinamiento de teorías son:

a) **El modelo de regla**. Este elemento se hereda de SLAVE y es una característica importante de este sistema porque permite deteminar las variables relevantes para cada regla.

b) **La heurística**. La manera especial en la cual se combinan los procesos de especificación, generalización, adición y eliminación de reglas.

c) **El proceso de inferencia**. Podemos distinguir dos procesos diferentes de inferencia: caso crisp y caso difuso. Como se muestra en el capítulo anterior, SLAVE trata ambos tipos de problemas de aprendizaje. Para el primero de ellos, SLAVE usa una función para determinar la compatibilidad entre un ejemplo y el conjunto de reglas y utiliza un mecanismo especial de resolución de conflictos basado en una medida sobre la capacidad de predicción de cada regla. Por consiguiente, el algoritmo de refinamiento supone que el conjunto de reglas que se ofrece como entrada está ordenado siguiendo este criterio. Para problemas con consecuente difuso, el orden de las reglas es menos importante ya que la salida se obtiene mediante la combinación de las reglas. En este caso, se usa un proceso de inferencia y un método de nitificación.

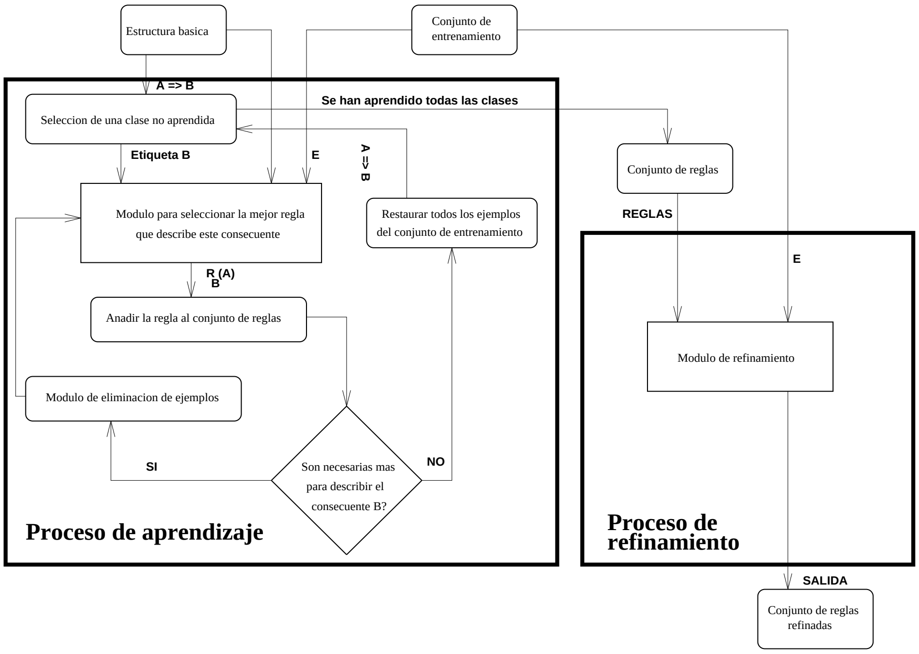


Figura 3.1: Estructura del algoritmo de aprendizaje SLAVE+R

En los siguientes apartados, describimos las características fundamentales del algoritmo de refinamiento. En primer lugar definimos la propuesta para el caso crisp y posteriormente la propuesta para el caso difuso.

3.2.1 Refinamiento de reglas difusas con consecuente crisp

El algoritmo de refinamiento intenta mejorar el conjunto de reglas obtenido por SLAVE, reduciendo los problemas descritos previamente. El algoritmo se compone fundamentalmente de dos tareas. La primera de ellas consiste en mejorar la precisión de cada regla. Con este propósito, se usa un proceso de especificación, el cual intenta que cada regla recubra el mayor número de ejemplos correctamente clasificados de la regla original sin recubrir sus ejemplos mal clasificados (precisión). La aplicación de esta tarea, puede provocar que algunos de los ejemplos mal clasificados por la regla original, pasen a estar no clasificados por ninguna. Por tanto, la siguiente tarea intenta, mediante un proceso de generalización, recubrir estos ejemplos no clasificados. Se utilizan dos mecanismos para recubrir estos ejemplos: o bien generalizando las reglas ya existentes o bien añadiendo nuevas reglas que recubran específicamente estos ejemplos. Las dos tareas anteriores se repiten hasta que se satisfaga una condición de terminación. La figura 3.2 muestra los diferentes pasos de este algoritmo.

El refinamiento usa una función heurística y una estrategia de ascensión de colinas para seleccionar la acción más prometedora en cada paso del algoritmo. Consideramos

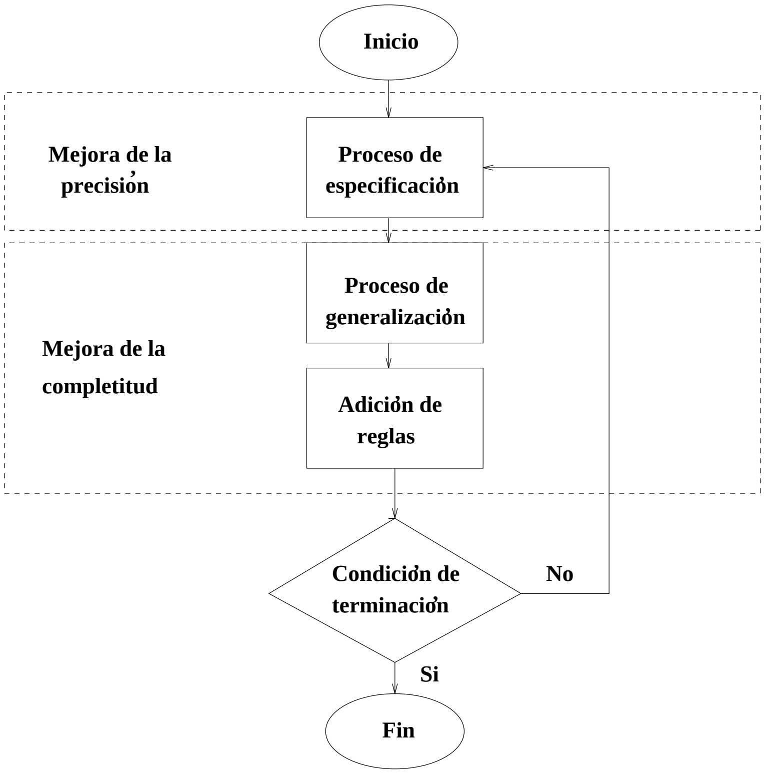


Figura 3.2: Algoritmo de refinamiento

una función que mide la capacidad de predicción global de un conjunto de reglas sobre el conjunto de ejemplos. Esta función determina la precisión que ofrece el proceso de inferencia de SLAVE (descrito en el capítulo anterior) sobre el conjunto de entrenamiento.

La componente heurística del algoritmo de refinamiento seleciona las reglas en el orden previamente descrito en el conjunto de reglas. Sin embargo, no se ha considerado ninguna ordenación especial para las variables y sus valores y se toma sobre ellas un orden por defecto.

A continuación, describimos cada uno de los pasos del algoritmo de refinamiento de una forma más detallada.

1) Mejorar la precisión de las reglas

En este paso, tratamos de mejorar la capacidad de predicción del conjunto de reglas obtenido por SLAVE sobre el conjunto de entrenamiento, incrementando la precisión de cada regla.

Obtener un conjunto de reglas, donde todas las reglas sean completamente precisas no es posible en todos los casos. Por ejemplo, cuando trabajamos con conjuntos de ejemplos que contienen ruido, puede ocurrir que un subconjunto de ellos se pueda clasificar correctamente, pero no podemos decir nada sobre el resto. Por esto, la precisión de la regla establece un grado en términos de la relación existente entre

Esquema 3.1 *Mejora de la precisión de las reglas*

1. Se selecciona la regla más relevante.
2. Se selecciona una variable relevante y se hace lo siguiente:
    - 2.1. Se selecciona un valor activo.
    - 2.2. Se desactiva este valor.
    - 2.3. Si se mantiene o se mejora la precisión global, entonces la modificación se acepta. En otro caso, el valor se vuelve a activar.
    - 2.4. Si hay más valores activos, ir al paso 2.1.
    - 2.5. Si todos los valores de la variable están no activos, entonces esta regla se elimina del conjunto de reglas.
    - 2.6. Si hay variables relevantes aún no seleccionadas, ir al paso 2.
3. Si hay quedan más reglas, seleccionar la siguiente regla e ir al paso 2., en otro caso el proceso termina.

aciertos y fallos que produce una regla sobre el conjunto de ejemplos. Asociando este grado con el concepto de completitud, el primer objetivo del algoritmo de refinamiento consiste en encontrar un buen conjunto de reglas, en el cual cada regla debe tener un grado de completitud tan alto como sea posible.

SLAVE tiene tendencia a generalizar las reglas tanto como los parámetros de consistencia se lo permiten. Por esta razón, el primer paso del algoritmo de refinamiento intenta contrarestar este comportamiento mediante la especificación de las reglas. El objetivo es reducir el número de fallos en cada regla. Así, el proceso empieza especificando las reglas en el orden en el que aparecen en el conjunto de reglas.

Diremos que un valor para una variable está activo si este valor aparece para esa variable en la descripción de la regla. Por contra, diremos que un valor para una variable está no activo o desactivado si este valor no aparece para esta variable en la descripción de la regla. De esta manera, una operación de especificación se puede considerar como el cambio a no activo de un valor que previamente estaba activo. Diremos en este caso que el valor se desactiva.

Siguiendo la notación anterior, el esquema 3.1 describe este proceso.

Debemos destacar que el proceso anterior permite eliminar reglas del conjunto de reglas. Por consiguiente, la eliminación de reglas se considera como un caso particular de especificación que se produce cuando todos los valores de una variable están desactivados.

Otra consideración a destacar del proceso anterior es la justificación del orden elegido para realizar la especificación. Supongamos que la teoría que deseamos refinar está constituida por las reglas que se proponen en el ejemplo 3.2. Con esta configuración, a la regla más general le corresponde una posición menor que a la regla más específica, ya que su peso es superior. Por tanto, es posible que la especificación del antecedente de la regla más general provoque una mejora en el sistema al hacer útil a la más específica. Sin embargo, si se aplica el orden contrario en el proceso de especificación, el sistema terminará eliminando la reglas más especificas, con lo que el algoritmo de refinamiento pierde parte de la información que le proporcionó el algoritmo de aprendizaje.

2) **Mejorar la completitud generalizando las reglas**

En este segundo paso, el algoritmo de refinamiento intenta mejorar la completitud de las reglas usando un proceso de generalización ordenado. En este proceso, se añaden nuevos valores a la descripción de cada una de las reglas. El proceso acepta la generalización de una regla cuando la capacidad de predicción global del conjunto de reglas se mantiene o mejora la del conjunto de reglas sin la modificación. La mejora de la capacidad de predicción en este proceso tiene en cuenta las siguientes dos situaciones:

a) Los ejemplos no clasificados pueden ser cubiertos correctamente mediante la generalización de alguna de las reglas.

b) Los ejemplos que son clasificados incorrectamente por una regla con un orden menor, pueden ser clasificados correctamente cuando se generaliza alguna de las reglas con un orden mayor.

Por consiguiente, este paso del algoritmo mejora tanto la completitud como la precisión de algunas reglas de la teoría.

Este proceso tiene una estructura similar al paso anterior, pero se usa el orden inverso para la selección de las reglas, es decir, se empieza por las reglas de menor orden y se termina con las reglas de mayor orden. El esquema 3.2 muestra el algoritmo que describe el funcionamiento de este proceso.

La razón de utilizar el orden inverso en este proceso es la siguiente: el proceso de inferencia en caso de conflicto toma las reglas que aparecen en las primeras posiciones del conjunto de reglas. Una regla que ocupa la posición T en el conjunto de reglas puede incluir en su descripción, las descripciones de todas las reglas que tienen una posición inferior a T, por supuesto, si esto no implica una reducción en la capacidad de predicción del sistema. Este hecho provoca que las primeras reglas tiendan a ser más específicas que las que ocupan las últimas posiciones en el conjunto de reglas. Por tanto, las últimas reglas tiene una tendencia mayor a ser generalizadas. Por

Esquema 3.2 *Mejora de la completitud de las reglas*

1. Se selecciona la regla menos relevante.
2. Se selecciona una variable relevante y se hace lo siguiente:
   - 2.1. Se selecciona un valor no activo.
   - 2.2. Se activa este valor.
   - 2.3. Si se mantiene o se mejora la precisión global, entonces la modificación se acepta. En otro caso, el valor se vuelve a desactivar.
   - 2.4. Si hay más valores no activos, ir al paso 2.1.
   - 2.5. Si hay variables relevantes aún no seleccionadas, ir al paso 2.
3. Si aún quedan más reglas, seleccionar la siguiente regla e ir al paso 2., en otro caso el proceso termina.

consiguiente, esta reglas están más cerca de sus descripciones definitivas y permiten mayor flexibilidad con respecto a la generalización de las reglas de orden inferior.

3) **Mejorar la completitud añadiendo nuevas reglas**

El siguiente paso en el algoritmo, consiste en añadir nuevas reglas para recubrir los ejemplos que no se pudieron recubrir por ninguna de las reglas en los pasos anteriores. Este paso añade la regla más específica con mejor adaptación para cada ejemplo no cubierto del conjunto de entrenamiento. La forma de construcción de esta regla se describe en el ejemplo 2.6 del capítulo 2.

4) **Condición de terminación**

Los pasos anteriores se repiten hasta que el conjunto de reglas es estable. En el algoritmo, se determina que un conjunto de reglas es estable cuando el número de reglas y la precisión se mantienen en dos iteraciones consecutivas.

3.2.2 Refinamiento para reglas con consecuente difuso

El algoritmo de refinamiento descrito anteriormente es válido cuando la variable de clasificación es crisp. Sin embargo, cuando la variable de clasificación es difusa, es necesario modificar el algoritmo de refinamiento, ya que los problemas de este tipo tienen un mecanismo de inferencia diferente. El modelo de inferencia propuesto para el caso difuso contempla la colaboración entre las reglas para determinar la salida del sistema. Esta característica es una de las diferencias fundamentales entre ambas implementaciones. Mientras en el caso crisp, se realiza un proceso previo de ordenación de reglas atendiendo al peso de las mismas, en el caso difuso, no se aplica este proceso, y se toma el orden en el que las reglas aparecen en el conjunto de reglas.

Por esta razón, definimos una readaptación al caso difuso del refinamiento de teorías. Básicamente, el algoritmo mantiene la estructura del anterior (figura 3.2), pero incorporando dos nuevos componentes: un mecanismo de inferencia para este tipo de reglas y una medida de error que establece la bondad del conjunto de reglas.

El mecanismo de inferencia coincide con el descrito en SLAVE para este tipo de problemas, es decir, el sistema de inferencia max-min y la media de las alturas como método de nitificación, usando el operador mínimo como t-norma [34].

En base a este sistema de inferencia, definimos una medida de error que establece la precisión del conjunto de reglas sobre el conjunto de ejemplos. Esta medida de error es una modificación sobre la media del error cuadrático que se define como:

$$Error = \sum_{i=1}^{n} \frac{(y_i - y_i^1)^2}{2n}$$

donde y_i es la salida del sistema, y_i^1 es la salida que propone el mecanismo de inferencia usando el conjunto de reglas y n es el número de ejemplos.

Por tanto, el algoritmo de refinamiento para el caso de variables de clasificación difusa es también un algoritmo heurístico que intenta maximizar la bondad del conjunto de reglas minimizando la medida de error.

Los pasos del algoritmo son los siguientes:

1) **Mejorar la precisión de las reglas:** Cuando trabajamos con problemas de variable consecuente crisp, la precisión establece una relación entre el número de ejemplos correctamente clasificados por una regla y el número total de ejemplos que son cubiertos por la misma. En el caso difuso se debe extender este concepto, ya que la salida del mecanismo de inferencia depende de la colaboración de las reglas que son aplicables. Por esta razón, en el caso difuso, definimos la precisión de una regla como la media de los errores que produce dicha regla en las salidas donde ésta se ha usado.

   Desde este punto de vista, este paso consiste en mejorar la precisión de la regla mediante procesos simples de especificación que permiten eliminar valores irrelevantes de la descripción de cada regla. En este sentido, un proceso de especificación trata de eliminar valores de las variables que provocan colaboraciones anómalas entre las reglas durante el proceso de inferencia. Además, es posible que reglas que provoquen un alto nivel de distorsión en las salidas del sistema (es decir, que tengan un nivel bajo de precisión) pueden ser eliminadas del conjunto de reglas. Esto es posible, ya que al igual que en el caso crisp, la eliminación de reglas se considera un proceso especial de especificación, donde todos los valores de una determinada variable son desactivados.

El comportamiento de este primer paso del algoritmo de refinamiento coincide con el propuesto en el esquema 3.1, considerando la nueva interpretación del concepto de precisión de las reglas.

2. **Aumentar el grado de influencia de las reglas en la interpolación de la salida:** En este paso, al contrario que en el anterior, tratamos de incrementar la interacción entre las distintas reglas mediante un proceso de generalización. Este incremento de la interacción entre las reglas intenta potenciar la colaboración, con el objetivo de aumentar la precisión global de la teoría. El proceso consiste en añadir un nuevo valor a una variable antecedente siempre y cuando la precisión global se mantenga o mejore estrictamente. De tal forma, que sólo se permite que una regla forme parte del conjunto de reglas que influyen en la interpolación para un subconjunto de ejemplos, si su participación no provoca un aumento del error global.

   El funcionamiento de este paso es semejante al propuesto en el esquema 3.2 teniendo en cuenta el cambio de significado del concepto de precisión de las reglas, al igual que en el paso anterior.

3. **Añadir nuevas reglas:** Supongamos que mediante el proceso anterior la interacción entre las distintas reglas no puede mejorarse generalizando algunas de las reglas existentes. En este caso, es posible que se necesiten nuevas reglas para decrementar los errores de las salidas. Este paso es el encargado de introducir estas nuevas reglas para reducir el error, cuando el paso anterior, no es capaz de realizarlo.

   El problema que surge en este paso es: ¿Qué regla es la que debemos añadir al conjunto de reglas?

   Responder a esta pregunta implica establecer un criterio para decidir entre todas las posibles reglas que se pueden introducir. Nuestro objetivo consiste en encontrar el conjunto de reglas que produzcan, en media, el menor error posible sobre el conjunto de entrenamiento. Con esta finalidad, establecemos un criterio basado en la elección de la regla más específica (es decir, la que menor distorsión provocará en el conjunto de reglas) que reduce el error del ejemplo peor representado del conjunto de entrenamiento. En otras palabras, construimos a partir del ejemplo que posee el máximo error individual, una regla que trate de reducir el mismo. En el caso de que exista más de un ejemplo que tenga el máximo error individual se tomará aleatoriamente cualquiera de ellos. La nueva regla se construye de la siguiente manera:

   – como parte condición: el antecedente más específico que mejor recubra a dicho ejemplo.

– como parte consecuente: el valor del dominio que menor error global proporcione, al ser incluida esta regla en el conjunto de reglas ya existentes.

Esta filosofía de actuación asume la siguiente idea: la introducción de una regla que mejore el máximo error individual, también mejorará el error medio global.

Este paso va introduciendo reglas hasta que se encuentra con un ejemplo que no mejora ni el máximo error individual del ejemplo, ni el error medio global. La regla que satisface esta condición no es introducida en la base de reglas.

3.2.2.1 Un ejemplo del funcionamiento del algoritmo de refinamiento

Supongamos que queremos realizar una modelización cualitativa de un sistema descrito por la función $h(x, y) = x^2 + y^2$ (3.3(a)). La figura 3.3(b) representa la superficie aprendida usando el algoritmo SLAVE a través de un conjunto de ejemplos del sistema original. Podemos observar que las reglas propuestas por SLAVE aproximan bastante correctamente la zona central de la superficie, pero no así los extremos de la gráfica.

El algoritmo de refinamiento intenta corregir la función, disminuyendo el error. En la figura 3.3(c) se muestra la superficie después del primer paso. Aparentemente, esta nueva superficie presenta una aproximación peor que la inicial aunque el error es mejor. Este aspecto es debido a la influencia negativa o insuficiente de algunas reglas en el proceso de interpolación. En las siguientes iteraciones del algoritmo, se va suavizando este comportamiento, que inicialmente es brusco.

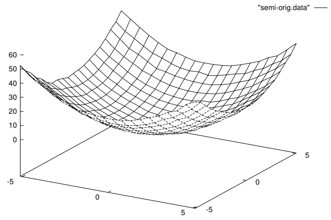
Despues de este paso, podemos ver la evolución del algoritmo de refinamiento como un proceso de aproximación suave a la superficie original. En la figura 3.3(d) podemos observar que la superficie experimenta una sensible mejora después del proceso de generalización. Esto es debido al aumento de las reglas que influyen positivamente en la interpolación sobre los subconjuntos de ejemplos mal representados.

La figura 3.3(e) muestra el comportamiento del conjunto de reglas después del paso 3. En esta gráfica podemos observar que las reglas existentes no eran suficientes para aproximar los extremos de la gráfica, pero después de incluir las nuevas reglas, esta aproximación se mejora.

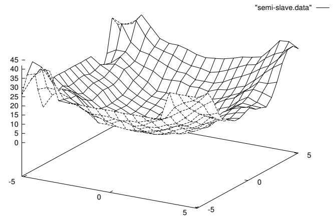
En este ejemplo, la gráfica después del cuarto paso no se muestra debido a que en este problema no existen variables irrelevantes, y por lo tanto, este paso devuelve la misma superficie que se obtuvo en el paso 3.

Para este ejemplo, el algoritmo de refinamiento se repitió ocho veces antes de hacer cierta la condición de terminación. En la figura 3.3(f) podemos ver la superficie resultante después de la ejecución del algoritmo. Se puede observar que es bastante similar a la superficie original, y que mejora notablemente la superficie de partida (figura 3.3(b)).

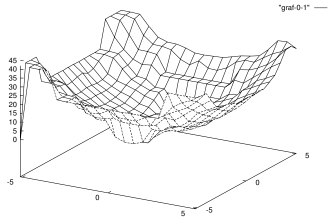
En la identificación de sistemas se suele usar un modelo de regla donde cada variable antecedente toma uno y sólo un valor de su dominio [22, 68, 159]. El modelo usado



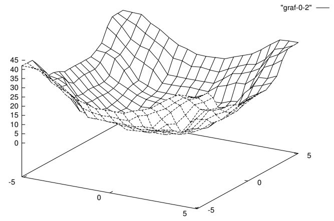
(a) Función $h(x, y) = x^2 + y^2$



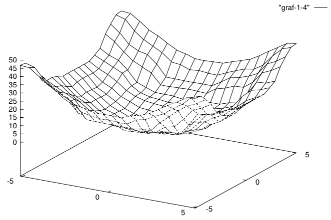
(b) Aproximación obtenida por SLAVE



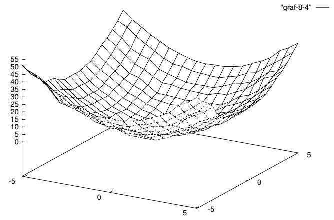
(c) Resultado después del paso 1



(d) Resultado después del paso 2



(e) Resultado después del paso 3



(f) Resultado final obtenido por el refinamiento

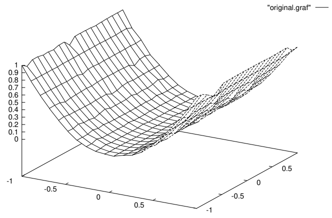
Figura 3.3: Funcionamiento del algoritmo de refinamiento

en SLAVE es más general, ya que el modelo anterior se puede incluir como un caso particular, presentando además la ventaja de obtener definiciones más simples. En el capítulo 2, mostramos las ventajas de este tipo de reglas como herramienta para detectar la relevancia de las variables antecedente para cada valor de la variable consecuente. En la figura 3.4 mostramos una función no lineal con variables irrelevantes para comprobar si el algoritmo de refinamiento mantiene las variables relevantes obtenidas por SLAVE. La función utilizada para esta prueba es

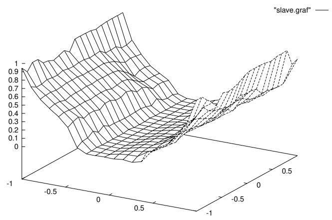
$$f(x, y) = x^2$$

(figura 3.4(a)) para el cual SLAVE devuelve 11 reglas (figura 3.4(b)). SLAVE+R (figura 3.4(c)) refina las reglas de SLAVE y devuelve 6 reglas donde la variable y se considera irrelevante para todas las reglas.

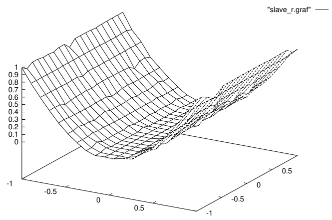
Por tanto, el algoritmo de refinamiento mantiene la tendencia a seleccionar, entre todas las reglas que permiten una correcta identificación del sistema, las que tienen una descripción más simple, es decir, que involucren el menor número posible de variables antecedente en su descripción.



(a) Función $f(x, y) = x^2$



(b) Resultado obtenido por SLAVE



(c) Resultado obtenido por SLAVE+R

Figura 3.4: Conservación de la simplicidad en el algoritmo de refinamiento

3.2.3 Estudios experimentales

Una vez definidos los algoritmos de refinamiento para el caso crisp y el caso difuso, pasamos a analizar el comportamiento real de dicho algoritmo. Para ello, realizamos una comparación entre los resultados que se obtienen al incluir dicho algoritmo de refinamiento como un paso más del proceso de aprendizaje con los resultados previamente obtenidos. Además, realizamos un estudio más detallado de la combinación de estos dos algoritmos en la aplicación práctica de un problema particular.

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	96.97	94.02	3.7	86
MONK1	100	100	4	48
MONK2	91.60	69.30	13.2	156
MONK3	97.21	92.64	5.4	57
INFARTO	98.69	81.56	4.9	323
WINE	100	89.69	5.3	2242
PIMA	82.99	77.08	16.2	3821
IONOSPHERE	100	83.23	16	12107
SOYBEAN	98.23	96.72	33	15903

Tabla 3.1: Resultados de SLAVE+R en el caso crisp

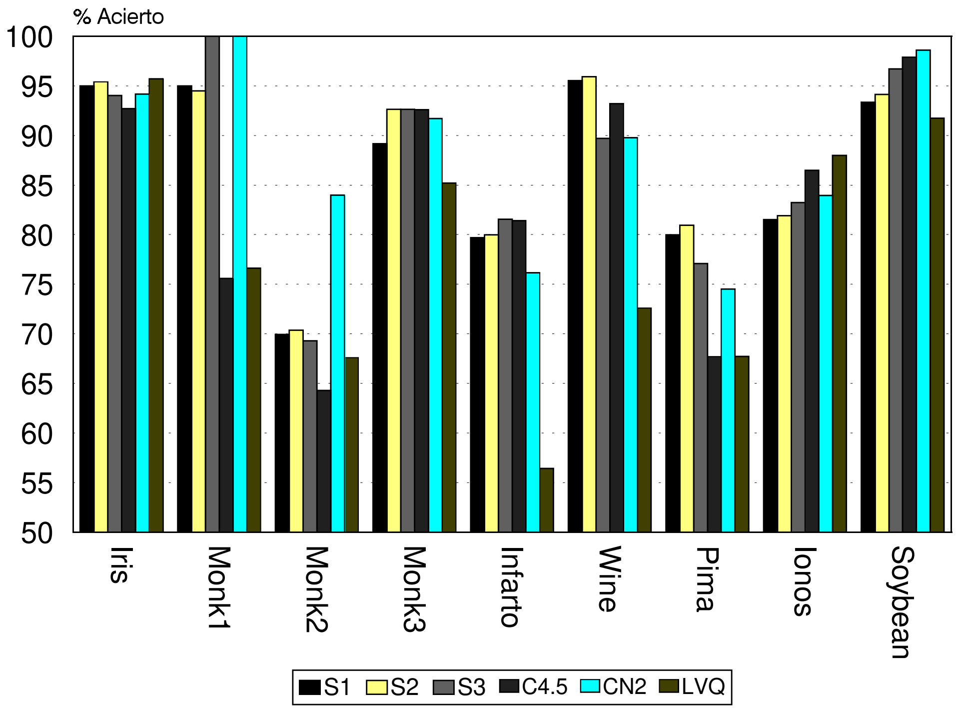
3.2.3.1 Comportamiento del algoritmo de refinamiento

En esta sección estudiamos los resultados que obtiene el algoritmo de aprendizaje SLAVE si se considera el algoritmo de refinamiento de teorías, definido anteriormente, como el último paso del algoritmo de aprendizaje. Para esta experimentación, se ha tomado como entrada al algoritmo de refinamiento las reglas obtenidas por SLAVE incluyendo los nuevos operadores, cuyos resultados se mostraron en la tabla 2.8. En la tabla 3.1 se muestran los resultados que se obtienen al incluir el algoritmo de refinamiento.

Debemos destacar que la medida de tiempo asociada a cada base de datos se obtiene como la suma del tiempo que consumió SLAVE para aprender el conjunto de reglas más el tiempo consumido por el algoritmo de refinamiento de teorías. Por esta razón, el tiempo de respuesta de SLAVE+R es peor que el obtenido al aplicar aisladamente SLAVE. Si comparamos estas diferencias de tiempo, podemos observar que el algoritmo de refinamiento tiene un tiempo de respuesta aceptable sobre bases de datos que contienen un número pequeño de reglas, mientras que este tiempo aumenta bastante cuando el número de reglas es grande.

Con respecto al número de reglas, podemos observar que el algoritmo de refinamiento lo reduce en todas las bases de datos propuestas, lo que conlleva que el conjunto final de reglas sea en términos generales más simple.

Sobre el porcentaje de acierto de los nuevos conjuntos de reglas podemos decir que hay un aumento en términos generales de los mismos, tanto sobre los conjuntos de entrenamiento como sobre los conjuntos de prueba (figura 3.5), aunque dicho aumento es más evidente sobre los primeros. Este hecho es razonable, ya que al igual que la mayoría de los algoritmos de refinamiento de teorías, su objetivos es ajustar los más posible sus reglas al conjunto de entrenamiento. Este comportamiento hace que los algoritmos de refinamiento tiendan de forma desmedida al sobreaprendizaje de los ejemplos del conjunto de entrenamiento, lo que ocasiona en la mayoría de las situaciones una reducción en la capacidad de predicción correcta sobre los ficheros de prueba. Sin embargo, pro-



S1	Algoritmo SLAVE descrito en el capítulo 2
S2	Algoritmo SLAVE descrito en el capítulo 2, incluyendo nuevos operadores genéticos
S3	Algoritmo SLAVE+R

Figura 3.5: Comparativa del acierto en los conjuntos de prueba (3)

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.09	94.87	3.6	26
MONK1	100	100	4	28
MONK2	92.90	66.20	16	203
MONK3	98.36	88.89	8	142
INFARTO	96.73	80.86	4.2	125
WINE	100	84.27	9.4	3134
PIMA	83.33	67.71	14	5503
IONOSPHERE	100	79.46	19	25448
SOYBEAN	97.96	92.14	36	30732

Tabla 3.2: Resultados usando el refinamiento como un algoritmo de aprendizaje

baremos experimentalmente, que esta tendencia hacia el sobreaprendizaje se reduce si el algoritmo de refinamiento parte del conjunto de reglas que propone SLAVE. Para ello, debemos resaltar que el algoritmo de refinamiento de teorías se comporta como un algoritmo de aprendizaje si parte inicialmente de una teoria nula, es decir, de un conjunto que no contiene ninguna regla. Así, siguiendo la misma estrategia para la confección de los experimentos, en la tabla 3.2 mostramos los resultados obtenidos por el algoritmo de refinamiento sobre las bases de datos propuestas, si lo utilizamos como un algoritmo de aprendizaje.

Estos resultados ponen de manifiesto lo comentado anteriormente, es decir, un aumento en la capacidad de clasificación sobre el conjunto de entrenamiento, pero una reducción en la capacidad de predicción con acierto sobre los conjuntos de prueba, siendo en algunas bases de datos esta diferencia bastante significativa. Por otro lado, el tiempo de respuesta del algoritmo de refinamiento empeora sobre todo en aquellas bases de datos que requieren más reglas para describir su comportamiento.

El algoritmo de refinamiento, parece un complemento adecuado para el algoritmo SLAVE que proporciona un comportamiento aceptable de los conjuntos de reglas obtenidos, mitigando su tendencia al sobreaprendizaje. En [58] se propone un estudio para comparar como afecta este algoritmo de refinamiento si se incorpora como fase final de algunos algoritmos de aprendizaje clásicos. En concreto el estudio se realizo sobre el algoritmo C4.5 comprobandose que se mejoraban los resultados de dicho algoritmo, pero que estos eran peores que los que propociona SLAVE+R.

Hasta el momento hemos puesto de manifiesto que asociar el algoritmo de refinamiento de teorías a SLAVE provoca en el caso crisp una mejora significativa en la capacidad de clasificación. Ahora, trataremos de comprobar si dicha mejora se refleja tambien cuando trabajamos con bases de datos con consecuente difuso. Así, siguiendo la filosofía anterior, y tomando como base los conjuntos de reglas obtenidos con SLAVE cuyo resultados se mostraron en la tabla 2.9 se propone aplicar a dichos conjuntos de reglas el algoritmo de refinamiento. Los resultados obtenidos se muestran en la tabla 3.3.

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
SIN	0.0012	0.0011	7	322
PARABOLA	0.0034	0.0039	3	132
3DPARABOLA	0.0017	0.0076	8	10237
MONTE	0.0031	0.0029	5	18401
PENDULO	5.3104	1.21	5	102
RICE	0.0099	0.0112	7.8	305

Tabla 3.3: Resultados de SLAVE+R para el caso difuso

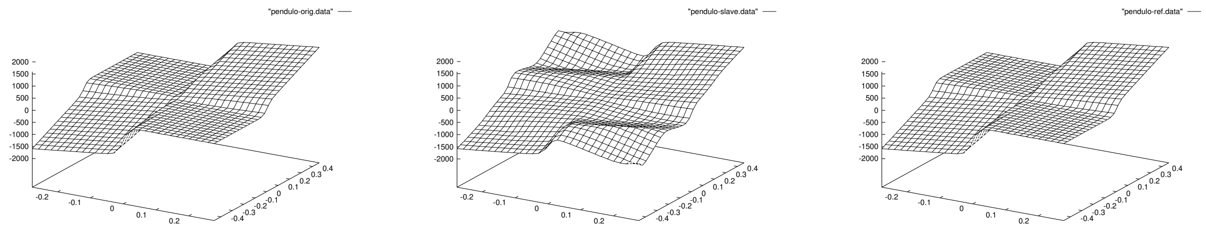
Al igual que sucede en el caso crisp, los resultados muestran una mejora en el comportamiento de los conjuntos de reglas obtenidos, tanto sobre los conjuntos de entrenamiento como sobre los conjuntos de prueba. Además, todas las consideraciones establecidas anteriormente para el caso crisp son aplicables al caso difuso excepto una. A diferencia del caso anterior, el refinamiento para el caso difuso tiene una tendencia a incluir más reglas que las que propone la ejecución aislada de SLAVE. Esta hecho refleja que el sistema de evaluación de reglas incluido en SLAVE tiene problemas para obtener el número adecuado de reglas en el caso difuso. Dejando a un lado esta situación, podemos concluir que asociar, en el caso difuso, el algoritmo de refinamiento a SLAVE produce resultados que son comparables a los que proponen otros algoritmos diseñados específicamente para resolver este tipo de problemas (ver tabla 2.6).

De entre los distintos resultados obtenidos, destacan por su bondad los obtenidos sobre la base de datos PENDULO. Para esta base de datos, SLAVE+R obtiene un número reducido de reglas simples que muestra la tabla 3.4.

1	If θ is NS and ω is NS Then f is NS
2	If θ is PS and ω is PS Then f is PS
3	If ω is ZR Then f is ZR
4	If θ is NM or NS and ω is NM Then f is NM
5	If θ is PS or PM and ω is PM Then f is PM

Tabla 3.4: Conjunto de reglas obtenido por SLAVE+R para el problema del péndulo

Además, podemos comprobar, como muestra la figura 3.6 que dicho conjunto de reglas produce una mejora significativa sobre la obtenida con SLAVE y que se asemeja considerablemente con la función original que describe el comportamiento del péndulo.



(a) Superficie original

(b) Superficie obtenida con SLAVE

(c) Superficie obtenida con SLAVE+R

Figura 3.6: Superficies obtenidas para el problema del péndulo

3.3 Incorporación de colaboración/competición entre reglas en SLAVE

El algoritmo de aprendizaje SLAVE determina la bondad de una regla en términos del número de ejemplos positivos y negativos que recubre dicha regla. Frente a esta posición, el algoritmo de refinamiento de teorías difusas, descrito anteriormente, determina la bondad de una regla en función de su comportamiento en el proceso de inferencia.

En el primer criterio, las reglas se aprenden de forma aislada, mientras que usando el segundo, se establecen relaciones de colaboración/competición entre las reglas, lo cual permite la determinación de un conocimiento más apropiado al problema. Por tanto, parece conveniente, si se quiere dotar a los algoritmos que usan el modelo iterativo de una visión más global durante el proceso de aprendizaje, el establecer relaciones de colaboración/competición entre las reglas. Para dotarlo de esta nueva visión, es necesario que la evaluación de las reglas no tenga en cuenta únicamente el recubrimiento de ejemplos positivos y negativos, sino que además se considere cual será su comportamiento durante el proceso de inferencia.

La incorporación de información sobre el comportamiento de las reglas en el proceso de inferencia no se puede realizar de forma directa en los algoritmos de aprendizaje que utilizan el modelo iterativo, ya que en un estado intermedio del proceso de aprendizaje, sólo se conocen las reglas aprendidas hasta el momento para un subconjunto de conceptos, desconociéndose absolutamente las reglas que se obtendrán para el resto. Por esta razón, no es posible realizar una inferencia de la misma forma como se hace en el algoritmo de refinamiento, donde se conoce el conjunto completo de reglas.

Sin embargo, una vez que se selecciona una regla en el modelo iterativo, se puede determinar como se comportará ésta durante el proceso de inferencia sobre un subconjunto de ejemplos del conjunto de entrenamiento. Si esta información se tiene en cuenta para aprender las siguientes reglas, se dota al modelo de la capacidad de establecer relaciones de colaboración/competición que permiten obtener una descripción más apropiada del sistema. En definitiva, si el algoritmo de refinamiento de teorías supone la introducción de colaboración entre soluciones parciales a través de un postproceso del conjunto de reglas obtenidas por SLAVE, ahora queremos modificar la propia estructura de SLAVE para que pueda incorporar información sobre la posible colaboración/competición de las reglas.

En consecuencia, el planteamiento que se desarrollará en las siguientes subsecciones consiste en modificar el algoritmo de aprendizaje SLAVE para que en la evaluación de las nuevas reglas se contemple la información de las reglas aprendidas previamente. Dado que esta información esta íntimamente relacionada con el mecanismo de inferencia asociado, deberemos definir dos aproximaciones diferentes de SLAVE, una para el caso crisp y otra para el caso difuso.

3.3.1 Colaboración/competición en el caso crisp

El método de inferencia asociado al caso crisp, establece una competición entre las diferentes reglas aplicables para determinar una respuesta. En el modelo iterativo, debido a que las reglas se aprenden de forma aislada, estas adolecen de esta componente competitiva. Este aislamiento provoca que no se tenga en cuenta el comportamiento de las reglas ya aprendidas sobre las nuevas reglas que se deben aprender.

Supongamos que se han aprendido T reglas para un determinado concepto y se desea aprender la siguiente regla T+1 para este mismo concepto. La información que maneja el algoritmo para aprender esta nueva regla se reduce a los ejemplos que las T reglas ya aprendidas no recubrieron. Sin embargo, se desconoce cómo las reglas ya aprendidas para otros conceptos, afectan a estos ejemplos. Además, cuando trabajamos con ejemplos y reglas difusas, la falta de información es más grave, ya que los ejemplos son cubiertos por las reglas en un cierto grado.

Por tanto, para problemas con consecuente crisp es importante conocer como se comportan las reglas ya aprendidas sobre cada ejemplo del conjunto de entrenamiento, para que esta información se tenga en cuenta durante el aprendizaje de nuevas reglas.

Con este fin, definimos el concepto de máximo grado de recubrimiento de un ejemplo por un conjunto de reglas, que nos permite establecer el grado de representación que ofrecen el conjunto de reglas a un ejemplo concreto del conjunto de entrenamiento.

Definición 3.3 *Sea e un ejemplo del conjunto de entrenamiento y $REGLAS$ un conjunto de reglas. Definimos* **el máximo grado de recubrimiento** *del ejemplo e^B como:*

$$Q(REGLAS, e) = \max_{R_B(A) \in REGLAS} \{U(e, A)\}$$

Los valores de $Q(REGLAS, e_i)$ obtenidos para todos los ejemplos e_i del conjunto de entrenamiento E, nos permiten conocer el grado de representación de las reglas sobre el conjunto E. Estas valoraciones obtenidas teniendo en cuenta todas las reglas que componen el conocimiento extraído hasta el momento no aportan demasiada información. Sin embargo, dado que SLAVE fija un concepto para aprender la mejor descripción que lo representa, podemos establecer en base a la definición anterior nuevas valoraciones que son representativas de como se comportará el proceso de inferencia sobre cada uno de los ejemplos. Así, fijado un concepto B, podemos dividir el conjunto de reglas obtenido hasta el momento en dos subconjuntos: el conjunto de reglas que describen el concepto B (R_B) y el conjunto de reglas que describen el resto de los conceptos ($R_{\overline{B}}$), de manera que REGLAS $= R_B \bigcup R_{\overline{B}}$,

La definición anterior, aplicada aisladamente a cada uno de estos dos subconjuntos, nos proporciona información sobre el comportamiento de las reglas sobre cada uno de los ejemplos del conjunto de entrenamiento. Así, si e^B es un ejemplo del conjunto de entrenamiento que pertenece al concepto B, definimos el **máximo grado de recubrimiento positivo** de dicho ejemplo como el máximo grado de recubrimiento que ofrece el subconjunto de reglas R_B sobre el ejemplo e^B, es decir ,

$$Q(R_B, e^B) = \max_{R_B(A) \in R_B} \{U(e^B, A)\}$$

Esta medida nos ofrece el mayor grado de compatibilidad entre el ejemplo e^B y el conjunto de reglas que permiten clasificar correctamente este ejemplo.

En el mismo sentido, podemos definir **el máximo grado de recubrimiento negativo** del ejemplo e^B como

$$Q_{\overline{B}}(R_{\overline{B}}, e^B) = \max_{R_{\overline{B}}(A) \in R_{\overline{B}}} \{U(e^B, A)\}$$

donde $Q_{\overline{B}}(R_{\overline{B}}, e^B)$ representa el mayor valor de compatibilidad entre e^B y el conjunto de reglas que clasificarán incorrectamente a dicho ejemplo.

Estos valores, que se van modificando a medida que se aprenden nuevas reglas durante el proceso iterativo, permiten determinar el comportamiento que las reglas producirán sobre un subconjunto de ejemplos del conjunto de entrenamiento en el proceso de predicción, de manera que si

$$Q_B(R_B, e^B) > Q_{\overline{B}}(R_{\overline{B}}, e^B)$$

podemos decir que con el conjunto de reglas que poseemos actualmente, ese ejemplo será clasificado correctamente, mientras que si

$$Q_B(R_B, e^B) < Q_{\overline{B}}(R_{\overline{B}}, e^B)$$

dicho ejemplo será incorrectamente clasificado.

Pero, ¿qué ocurre si $Q_B(R_B, e^B) = Q_{\overline{B}}(R_{\overline{B}}, e^B)$? En esta situación, se producirá un conflicto durante el proceso de inferencia para determinar la salida de este ejemplo. Por consiguiente, para determinar la regla aplicable, es necesario tener en cuenta el mecanismo de resolución de conflictos que usará este proceso de inferencia.

Como ya se describió anteriormente, el primer criterio utilizado por el mecanismo de resolución de conflictos está asociado al peso de las reglas. Incorporando esta nueva información a cada uno de los ejemplos, podemos introducir las siguientes definiciones:

Definición 3.4 *Sea e^B un ejemplo del conjunto de entrenamiento que describe el concepto B. Sea REGLAS el conjunto de reglas aprendidas hasta el momento, definido como $REGLAS = R_B \bigcup R_{\overline{B}}$, donde R_B es el conjunto de reglas que se han aprendido para el concepto B, mientras $R_{\overline{B}}$ es el conjunto de reglas que se han aprendido del resto de los conceptos.*

- *Diremos que un ejemplo e^B está correctamente clasificado si satisface al menos una de las dos siguientes condiciones:*

  i) $Q_B(R_B, e^B) > Q_{\overline{B}}(R_{\overline{B}}, e^B)$.

  ii) $Q_B(R_B, e^B) = Q_{\overline{B}}(R_{\overline{B}}, e^B)$ *y* $\omega(Q_B(R_B, e^B)) > \omega(Q_{\overline{B}}(R_{\overline{B}}, e^B))$.

- *Diremos que un ejemplo e^B está incorrectamente clasificado si no satisface ninguna de las condiciones anteriores.*

donde $\omega(Q_B(R_B, e^B))$ y $\omega(Q_{\overline{B}}(R_{\overline{B}}, e^B))$ representan el mayor peso de entre las reglas que alcanzan el máximo grado de recubrimiento positivo y negativo respectivamente.

Teniendo en cuenta la información que se introduce en el proceso con las anteriores valoraciones, proponemos la definición de dos nuevos conceptos: aciertos y fallos.

Consideraremos un ejemplo como un acierto para una nueva regla, si el ejemplo es correctamente clasificado, mientras que consideraremos un ejemplo como fallo para una nueva regla, si el ejemplo es incorrectamente clasificado. Antes de establecer una definición formal de estos dos conceptos, es necesario estudiar algunas consideraciones que son provocadas por la dependencia del orden existente durante el proceso de aprendizaje de los conceptos, y que se producen al incluir esta nueva información.

En el capítulo 2 se describió que una de las ventajas que aporta el modelo iterativo, es que es posible el aprendizaje de los conceptos de forma paralela, ya que en principio, no se establece ninguna dependencia entre ellos. Este hecho, implica que no sea relevante el orden en el cual se aprenden los conceptos. Sin embargo, la introducción de información sobre el comportamiento de las reglas en el proceso de inferencia, supone el establecimiento de dependencias entre las descripciones de los conceptos. Por tanto, esta modificación incluida en el método iterativo, provoca que el conjunto de reglas resultante dependa estrechamente del orden en el que los conceptos se aprenden.

Dado que el orden es importante, debemos considerar algunas situaciones que surgen por la propia filosofía del modelo iterativo, que ilustramos con el siguiente ejemplo:

$\boxed{\textbf{Ejemplo}}$ **3.3** *Supongamos que aplicamos un algoritmo de aprendizaje que sigue el modelo iterativo. Supongamos además que se han aprendido las descripciones necesarias para representar un determinado concepto B_1 y se ha obtenido la primera regla para otro concepto B_2. Tomemos de entre los ejemplos del conjunto de entrenamientos dos ejemplos, e^{B_1} que es un ejemplo del concepto B_1 cuyas descripciones ya han sido aprendidas y e^{B_3} que es un ejemplo del concepto B_3 cuyas descripciones no se han aprendido todavia. Supongamos además que debido a la influencia de la única regla del concepto B_2 resulta que los ejemplos e^{B_1} y e^{B_3} son clasificados incorrectamente.*

Ante esta situación, aunque ambos ejemplos son clasificados incorrectamente con el conocimiento obtenido hasta el momento, sabemos que e^{B_1} es un ejemplo incorrectamente clasificado, independientemente del conjunto de reglas que se obtengan para el resto de los conceptos, ya que no será posible incorporar una nueva regla para el concepto B_1. Sin embargo, e^{B_3} es un ejemplo cuya situación de clasificación incorrecta es temporal, ya que existe la posibilidad de que se seleccione una regla para el concepto B_3 que permita clasificarlo correctamente.

Este comportamiento implica que los ejemplos que son correctamente clasificados durante el proceso de aprendizaje, solamente pueden conocerse cuando se obtenga el conjunto definitivo de regla, mientras que algunos de los ejemplos que son incorrectamente clasificados pueden conocerse durante el proceso de aprendizaje, si los ejemplos pertenecen a un concepto que ya ha sido aprendido.

La situación anterior abre las dos siguientes interrogantes:

a) ¿Deben cuantificarse al mismo nivel los ejemplo que con el conocimiento actual son incorrectamente clasificados, que los ejemplos que son de forma definitiva incorrectamente clasificados?

b) ¿No sería apropiado introducir una nueva conducta a la hora de establecer la medida de bondad de las reglas que sea más restrictivo con los ejemplos de los conceptos ya aprendidos y menos exigente con los ejemplos de los conceptos que quedan por aprender?

En la definición de los conceptos de aciertos y fallos, damos respuesta a los dos interrogantes anteriores, penalizando más severamente las incorrecciones definitivas de los ejemplos y optando por un criterio de aprendizaje sobre los conceptos, de manera, que los primeros conceptos que se aprenden tienen una tendencia superior a especificar que los últimos. Así, establecemos las siguientes definiciones:

Definición 3.5 *Sea e^B un ejemplo del conjunto de entrenamiento que verifica pertenecer al concepto B. Sea $REGLAS$ el conjunto de reglas aprendidas hasta el momento, definido como $REGLAS = R_B \bigcup R_{\overline{B}}$, donde R_B es el conjunto de reglas que se han aprendido para el concepto B, mientras $R_{\overline{B}}$ es el conjunto de reglas que se han aprendido del resto de los conceptos.*

- *Diremos que* e^B **es un acierto para la regla** $R_c(d)$ **sii** $c = B$ *y se satisface una de las siguientes condiciones:*

  i) $U(e^B, d) > Q(REGLAS, e^B)$

  ii) $U(e^B, d) = Q(REGLAS, e^B)$ *y* $\omega(R_c(d)) > \omega(Q(REGLAS, e^B))$

- *Diremos que* e^B **es un fallo provisional para la regla** $R_c(d)$ *sii* $c \neq B$ *y se satisface la siguiente condición:*

  i) $U(e^B, d) > Q_B(R_B, e^B)$ *y* $Q_B(R_B, e^B) = 0$

- *Diremos que* e^B **es un fallo definitivo para la regla** $R_c(d)$ *sii* $c \neq B$ *y se satisface una de las siguiente condiciones:*

  i) $U(e^B, d) > Q_B(R_B, e^B)$ *y* $Q_B(R_B, e^B) > 0$

  ii) $U(e^B, d) = Q_B(R_B, e^B)$, $Q_B(R_B, e^B) > 0$ *y* $\omega(R_c(d))) > \omega(Q_B(R_B, e^B)$

Por consiguiente, el sistema determina que un ejemplo es un acierto para la nueva regla que se propone si el ejemplo representa al concepto que se intenta aprender, y dicha regla será la utilizada en el proceso de inferencia para clasificar a este ejemplo, considerando las reglas que se han aprendido hasta el momento. Por contra, se establecen dos definiciones para considerar a un ejemplo como un fallo de la regla. Ambas consideran que, sin ser el ejemplo representativo del concepto que se está aprendiendo, la adaptación entre el ejemplo y la nueva regla es mayor que la adaptación con las reglas de su clase. El elemento diferenciador entre ambas definiciones lo establece el valor $Q_B(R_B, e^B)$. Si $Q_B(R_B, e^B) = 0$ solamente puede indicar dos cosas:

- El conjunto $R_B = \emptyset$, lo que indica, o bien que todavía no se han explorado las reglas para este concepto en el algoritmo de aprendizaje, o bien que se han explorado, pero no se ha encontrado ninguna regla que permita describir a dicho concepto.

- El ejemplo e^B no tiene adaptación con ninguna de las reglas obtenidas para clasificar el concepto B.

En cualquier caso, estos fallos asociados a la regla se consideran provisionales, bien por que todavía no se ha explorado el concepto B, o bien, porque se ha explorado y no ha sido posible representar a este ejemplo.

Por otro lado, si $Q_B(R_B, e^B) > 0$ indica inequivocamente que el concepto B ha sido explorado por el sistema de aprendizaje, y por eso, si la nueva regla tiene una adaptación superior con el ejemplo que la obtenida por las reglas de su clase, este fallo no podrá ser corregido por el algoritmo y será por consiguiente un fallo definitivo.

A pesar de lo dicho anteriormente, no podemos evaluar únicamente la bondad de una regla como relación entre aciertos y fallos por la siguiente razón: la evaluación de una regla se realiza en un punto intermedio del proceso de aprendizaje, por lo que la inclusión de nuevas reglas pueden alterar fuertemente su estimación de aciertos. Por consiguiente, dicha estimación de los aciertos de una regla debe asemejarse lo más posible a la evaluación real de los mismos al final del proceso de aprendizaje. Este hecho pone de manifiesto la necesidad de graduar los aciertos y los errores de una regla en la medida en que estos sean más o menos definitivos. La función de adaptación entre el antecedente de una regla y un ejemplo establece una medida del grado de compatibilidad entre ambos. Cuanto mayor sea este grado de compatibilidad entre ambos, con más certeza podemos asegurar que dicha regla será utilizada en el proceso de predicción para clasificar a dicho ejemplo. Por tanto, utilizaremos esta medida para cuantificar tanto los aciertos como los fallos. En base a lo anteriormente argumentado, establecemos las siguientes definiciones:

Definición 3.6 *Sea E un conjunto de ejemplos de entrenamiento, sea REGLAS el conjunto de reglas aprendidas hasta el momento y sea $R_c(d)$ una nueva regla que se desea incorporar al conjunto REGLAS. Definimos:*

- *Número de aciertos de la regla $R_c(d)$*

$$N^{+}(R_c(d)) = \sum_{e^B \in E} U(e^B, d) \mid B = c \text{ y } e^B \text{ es un acierto para la regla } R_c(d)$$

- *Número de fallos provisionales de la regla $R_c(d)$*

$$N^{-}(R_c(d)) = \sum_{e^B \in E} U(e^B, d) \mid B \neq c \text{ y } e^B \text{ es un fallo provisional o definitivo para la regla } R_c(d)$$

- *Número de fallos definitivos de la regla $R_c(d)$*

$$ND^{-}(R_c(d)) = \sum_{e^B \in E} \Upsilon \mid B \neq c \text{ y } e^B \text{ es un fallo provisional o definitivo para la regla } R_c(d)$$

donde e^B es un ejemplo del conjunto de entrenamiento que representa al concepto B y $\Upsilon \in [0,1]$ es un valor constante para la penalización de los errores definitivos.

Para ilustrar las anteriores definiciones, proponemos el siguiente ejemplo:

$\boxed{\textbf{Ejemplo}}$ **3.4** *Sea $E = \{e_1^{B_1}, e_2^{B_1}, e_3^{B_1}, e_1^{B_2}, e_2^{B_2}, e_3^{B_2}, e_4^{B_2}, e_1^{B_3}, e_2^{B_3}\}$ un conjunto de entrenamiento que describe tres clases diferentes B_1, B_2 y B_3. Sea REGLAS el conjunto de*

	$e_1^{B_1}$	$e_2^{B_1}$	$e_3^{B_1}$	$e_1^{B_2}$	$e_2^{B_2}$	$e_3^{B_2}$	$e_4^{B_2}$	$e_1^{B_3}$	$e_2^{B_3}$
Q_B	0.3	0.8	0.3	0.1	0.7	0.0	0.0	0,0	0.0
$\omega(Q_B)$	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.0	0.0
$Q_{\overline{B}}$	0.2	0.5	0.8	0.4	0.3	0.5	0.6	0.0	0.0
$\omega(Q_{\overline{B}})$	0.6	0.6	0.6	0.6	0.6	0.6	0.0	0.0	0.0

Tabla 3.5: Grados de recubrimiento y peso de las reglas asociados a cada ejemplo de E dado el conjunto de reglas REGLAS

reglas obtenido hasta el momento y supongamos que los valores de los máximos grados de recubrimiento positivo y negativo, junto con los respectivos pesos asociados a estos valores para cada ejemplo son los que se proponen en la tabla 3.5.

Con estas condiciones, queremos evaluar una nueva regla $R_{B_2}(A)$ para la clase B_2 asumiendo que todas las reglas necesarias para aprender el concepto B_1 ya se han aprendido y se encuentran en el conjunto REGLAS. La compatibilidad de cada uno de los ejemplos con esta nueva reglas y el peso de la misma se muestra en la tabla 3.6.

	$e_1^{B_1}$	$e_2^{B_1}$	$e_3^{B_1}$	$e_1^{B_2}$	$e_2^{B_2}$	$e_3^{B_2}$	$e_4^{B_2}$	$e_1^{B_3}$	$e_2^{B_3}$
$U(e_i, A)$	0.3	0.6	0.4	0.6	0.5	0.7	0.3	0.5	0.0
$\omega(R_{B_2}(A))$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0

Tabla 3.6: Compatibilidad de cada ejemplo con la regla $R_{B_2}(A)$.

Para determinar los valores $N^+(R_{B_2}(A))$, $N^-(R_{B_2}(A))$ y $ND^-(R_{B_2}(A))$ necesitamos establecer previamente los aciertos y los errores que provoca la inclusión de la nueva regla. Así, tenemos que:

- *El ejemplo $e_1^{B_1}$ no es un fallo para la nueva regla, ya que aunque el máximo grado de recubrimiento positivo coincide con el grado de compatibilidad de esta regla, el peso asociado a la primera es superior. Por tanto, la inclusión de esta regla, no impide que este ejemplo sea correctamente clasificado.*
- *Tampoco el ejemplo $e_2^{B_1}$ no es un fallo para esta regla, debido a que la adaptación con una regla de su clase es mejor que la que proporciona esta nueva regla.*
- *Por contra, el ejemplo $e_1^{B_3}$ si es un fallo que introduce esta regla, ya que no es de la clase de la regla que se introduce, y la adaptación de esta es mejor que la que se tenía sobre dicho ejemplo.*
- *El ejemplo $e_3^{B_1}$ aunque debería ser un fallo para esta regla, no es contabilizado como tal, ya que por la configuración de valores y por ser B_1 una clase ya aprendida, este ejemplo ya ha sido computado como un error definitivo para una regla anterior de*

la clase B_2. Por esta razón, los errores definitivos no se toman en consideración para evaluar la bondad de las nuevas reglas.

- *Los ejemplos $e_1^{B_2}$ y $e_3^{B_2}$ son aciertos de esta regla, por mejorar la anterior compatibilidad que ofrecian las reglas de su clase, y superar la compatibilidad que ofrecen las reglas del resto de las clases.*
- *Los ejemplos $e_2^{B_2}$ y $e_4^{B_2}$ no pueden computarse como aciertos para esta regla, ya que para el primero de ellos existe otra regla de su clase que lo representa mejor y para el segundo, el grado de compatibilidad negativo es superior al que ofrece esta regla.*

En definitiva, tenemos que $e_1^{B_2}$ y $e_3^{B_2}$ son aciertos para esta regla, y $e_1^{B_3}$ es el fallo que introduce esta regla. Además, debemos considerar que el fallo que introduce la regla es provisional, ya que las reglas para representar las clase B_3 no se han aprendido todavía.

Una vez establecidos los aciertos y los fallos, podemos calcular el número de aciertos y fallos que introduce la regla de la siguiente forma:

$$N^+(R_{B_2}(A)) = U(e_1^{B_2}, A) + U(e_4^{B_2}, A) = 0.6 + 0.7 = 1.3$$

$$N^-(R_{B_2}(A)) = U(e_1^{B_3}, A) = 0.5$$

$$ND^-(R_{B_2}(A)) = 0$$

Si suponemos que la regla $R_{B_2}(A)$ es la mejor solución parcial de segundo orden y debe incluirse en el conjunto REGLAS, entonces la tabla anterior se modificaría tal como muestra la table 3.7.

	$e_1^{B_1}$	$e_2^{B_1}$	$e_3^{B_1}$	$e_1^{B_2}$	$e_2^{B_2}$	$e_3^{B_2}$	$e_4^{B_2}$	$e_1^{B_3}$	$e_2^{B_3}$
Q_B	0.3	0.8	0.3	0.6	0.7	0.7	0.3	0,0	0.0
$\omega(Q_B)$	0.7	0.7	0.5	0.7	0.5	0.5	0.5	0.0	0.0
$Q_{\overline{B}}$	0.3	0.6	0.8	0.4	0.3	0.5	0.6	0.5	0.0
$\omega(Q_{\overline{B}})$	0.5	0.5	0.6	0.6	0.6	0.6	0.6	0.5	0.0

Tabla 3.7: Modificación de la tabla de recubrimientos

Tras la modificación, mirando la tabla de recubrimientos 3.7 podemos determinar que ejemplos serán correctamente clasificados si usamos el conjunto REGLAS obtenido hasta el momento. Si el máximo grado de recubrimiento positivo es mayor que máximo grado de recubrimiento negativo para un ejemplo o si ambos grados son iguales, pero el peso asociado al positivo es mayor, podemos asegurar que ese ejemplo se clasificará correctamente. Por contra, si las anteriores condiciones no se satisfacen, entonces dicho ejemplo no será correctamente clasificado.

El hecho de considerar las medidas definidas anteriormente para evaluar la bondad de una regla, provoca dos ventajas sustanciales sobre la anterior versión del algoritmo de aprendizaje SLAVE:

a) Los ejemplos sólamente se computan como positivos o negativos para una regla, cuando estos causan un cambio significación en el proceso de clasificación teniendo en cuenta las reglas que se aprendieron previamente.

b) El algoritmo de aprendizaje, manteniendo la información anterior, conoce hasta donde puede recubrir los ejemplos negativos para que estos no sean considerados como fallos o errores en el proceso de clasificación. Esta información dota al algoritmo de la capacidad de conocer hasta donde puede generalizar las reglas sin perjudicar la clasificación de los ejemplos, por consiguiente, esta forma de evaluar permite establecer límites a la generalización de las reglas.

Así, consideramos la inclusión de estas medidas para definir la función de evaluación de reglas en el algoritmo genético, alterando la definición del primero de sus criterios. Hasta este momento, hemos considerado como primer criterio de evaluación el producto entre el grado de consistencia débil de la regla por su grado de completitud. La definición anterior tomaba en consideración el número de ejemplos positivos y negativos para determinar el grado de consistencia. Ahora, alteramos dicha definición, cambiando el número de ejemplos positivos por el número de aciertos de la regla, y el número de ejemplos negativos por el número de fallos de la siguiente forma:

$$\Gamma^N_{k_1 k_2}(R_B(A)) = \begin{cases} 1 & \text{si } R_B(A) \in \Delta^{k_1} \\ \frac{k_2 N^+(R_B(A)) - (N^-(R_B(A)) + ND^-(R_B(A)))}{N^+(R_B(A))(k_2 - k_1)} & \text{si } R_B(A) \notin \Delta^{k_1} \text{ y } R_B(A) \in \Delta^{k_2} \\ 0 & \text{en otro caso} \end{cases}$$

donde $k_1, k_2 \in [0,1]$ y $k_1 < k_2$ son los parámetros de consistencia débil, $N^+(R_B(A))$, es el número de aciertos, $N^-(R_B(A))$ es el número de fallos y $ND^-(R_B(A))$ es el número de fallos definitivos de la regla $R_B(A)$. Además, en esta definición modificamos el concepto de conjunto de descripciones admisibles Δ^k de la siguiente manera:

$$\Delta^k = \{R_B(A) \mid (N^-(R_B(A)) + ND^-(R_B(A))) < kN^+(R_B(A)) \text{ y } N^+(R_B(A)) > 0\}$$

Debemos destacar que esta nueva definición del grado de consistencia débil conserva todas las propiedades de la definición anterior de la misma que se describieron en el capítulo anterior.

Por consiguiente, la modificación introducida en el primer criterio de evaluación de reglas queda como $(\Gamma^N_{k_1,k_2}(R) \times \Lambda(R))$, manteniendose la misma definición para el resto de los criterios.

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.83	95.93	4.8	74
MONK1	100	100	4.2	36
MONK2	94.67	83.19	14.2	125
MONK3	99.02	92.12	7.2	55
INFARTO	99.86	84.62	4.2	148
WINE	99.67	95.79	5	612
PIMA	85.45	76.56	31	2661
IONOSPHERE	99.84	73.53	53.6	9698
SOYBEAN	96.15	96.90	30	10190

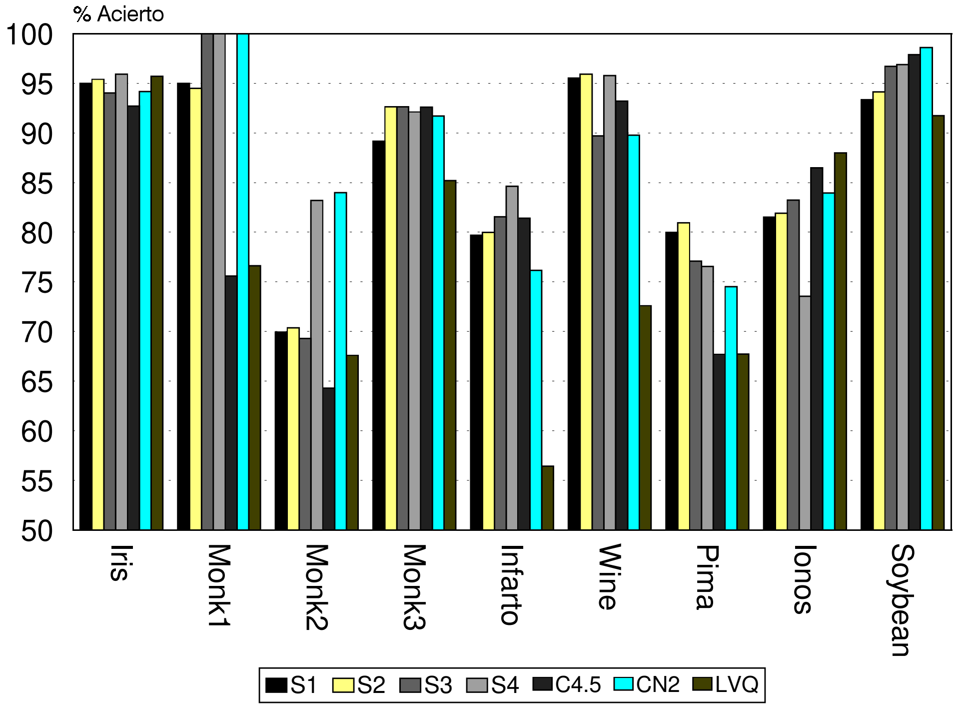
Tabla 3.8: Resultados con la nueva versión de SLAVE

3.3.1.1 Pruebas experimentales

A continuación, estudiaremos empíricamente los efectos que produce esta modificación con respecto a la versión anterior de SLAVE y que ventajas e inconvenientes introduce con respecto a la inclusión como paso final del proceso de aprendizaje de un algoritmo de refinamiento de teorías. Con tal fin, realizamos la ejecución de la nueva versión de SLAVE tomando como parámetros del mismo los que se propusieron en la tabla 2.2 incluyendo como constante de penalización de los errores definitivos $\Upsilon = 1$. Los resultados de la experimentación se muestran en la tabla 3.8.

De la comparación de estos resultados con los obtenidos por SLAVE+R, que se muestran en la tabla 3.1, podemos extraer las siguientes conclusiones:

- Aparece una significativa mejora en la capacidad de predicción con acierto del sistema sobre los conjuntos de prueba, si exceptuamos la base de datos IONOSPHERE (figura 3.7).
- El porcentaje de acierto sobre los conjuntos de entrenamiento son bastante parecidos, lo que indica que la modificación incluida en SLAVE potencia su capacidad de representación sobre el conjunto de entrenamiento a diferencia de la versión anterior.
- La nueva versión necesita un mayor número de reglas para representar los conceptos. Este hecho se hace patente fundamentalmente en aquellas bases de datos donde se encuentran involucradas muchas variables de entrada. En este sentido, parece que el sistema tiene problemas para generalizar las reglas, cuando aparece un gran número de ellas. En el siguientes capítulo estudiamos incorporar una mejora a SLAVE que le permita realizar una selección más adecuada de las variables.
- Sin lugar a dudas, la mejora más sustancial sobre la anterior versión de SLAVE y sobre SLAVE+R viene dada por el tiempo de respuesta del nuevo algoritmo, que se reduce de forma sustancial.



S1	Algoritmo SLAVE descrito en el capítulo 2
S2	Algoritmo SLAVE descrito en el capítulo 2, incluyendo nuevos operadores genéticos
S3	Algoritmo SLAVE+R
S4	La nueva propuesta de SLAVE

Figura 3.7: Comparativa del acierto en los conjuntos de prueba (4)

Básicamente, podemos considerar que la modificación incluida en SLAVE mejora en todos los aspectos a las versiones propuestas anteriormente excepto en el número de reglas. Por esta razón, desde este momento asumiremos para el resto del desarrollo de esta memoria, que cuando nos refiramos al algoritmo de aprendizaje SLAVE para el caso crisp, estaremos aludiendo a la modificación propuesta en la evaluación de la bondad de las reglas. Además, debemos destacar que el algoritmo de refinamiento de teorías expuesto en este capítulo sigue considerandose como una alternativa válida para el refinamiento de las reglas obtenidas por cualquier algoritmo de aprendizaje, y en concreto con la nueva versión propuesta para SLAVE.

3.3.2 Información necesaria para consecuente continuo

Como mencionamos anteriormente, el modelo de aprendizaje iterativo consiste en dado un concepto o valor de la variable consecuente, buscar la mejor descripción que permita representar a dicho consecuente. Como hemos visto anteriormente, cuando el dominio de la variable consecuente no contiene solapamiento entre sus distintos valores, el proceso de inferencia asociado establece una competición entre las reglas para determinar la salida del sistema.

Sin embargo, en aquellos problemas que presentan solapamiento en los valores de la variable consecuente, el proceso de aprendizaje no sólo debe establecer competición entre los distintos conceptos, sino que en algunos casos, se hace necesario establecer relaciones de competición entre ellos. En el siguiente ejemplo, ilustramos esta situación.

Ejemplo 3.5 *Supongamos que se desea aprender el conjunto de reglas que permita clasificar unas determinadas enfermedades en función de un conjunto de síntomas. Supongamos además que la variable consecuente, es una variable cuyos valores tienen la siguiente estructura:*

Cardiaca

Infarto

Soplo

Respiratoria

Gripe

Resfriado

En base a esta estructura, parece evidente que se deben establecer dos tipos de criterios en el proceso de aprendizaje. Por ejemplo, se debe establecer una competición entre valores tales como Cardiaca *y* Respiratoria*, o entre* Infarto *y* Soplo *o entre* Gripe *y* Resfriado*, mientras que sobre los valores* Cardiaca *e* Infarto *se debe establecer un colaboración entre las descripciones para ambos conceptos.*

En base a lo anterior, podemos decir que un ejemplo existente en el conjunto de entrenamiento que describa el concepto Infarto *no debe ser considerado como un ejemplo negativo al concepto* Cardiaca*. Por consiguiente, el proceso de aprendizaje debe tener en cuenta que todo* Infarto *es una enfermedad* Cardiaca*, y por tanto, para aprender esta última, será necesario que se establezca una competición entre ambos conceptos.*

Esta diferencia en el planteamiento del aprendizaje de los diferentes conceptos, permite estar más cerca de extraer un conocimiento válido para el problema, si el proceso de inferencia asociado presenta una actitud semejante.

Para aclarar aún más este punto, proponemos el siguiente ejemplo:

Ejemplo **3.6** *Supongamos que se desea obtener el conocimiento necesario para describir un sistema, donde el dominio de la variable consecuente está discretizado mediante cuatro intervalos con solapamiento como muestra la figura 3.8.*

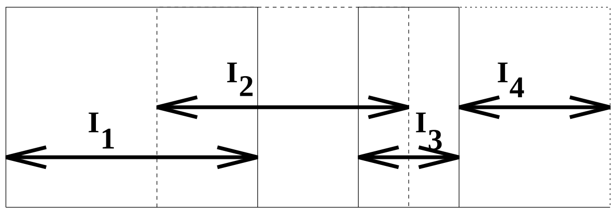


Figura 3.8: Dominio intervalar con solapamientos

Si deseamos aprender las reglas que permiten describir el intervalo I_1*, podemos optar por dos comportamientos:*

1. *Considerar como aciertos todos los ejemplos cuyo valor de la variable consecuente pertenezca el intervalo I_1, y considerar como errores todos los ejemplos e_i tal que $e_i \in I_j$, donde $j = 2, 3, 4$.*

2. *Considerar como aciertos todos los ejemplos cuyo valor de la variable consecuente pertenezca el intervalo I_1, y considerar como errores todos los ejemplos que no pertenezcan a este intervalo.*

Con la primera interpretación, estamos caracterizando los ejemplos que pertenecen al intervalo I_1 que no pertenecen al intervalo I_2, es decir, se establece una compentencia directa entre estos dos intervalos. Este comportamiento, que es apropiado cuando no existe solapamiento entre los valores de la variable consecuente, provoca que el algoritmo de aprendizaje no tome ninguna postura sobre los ejemplos que pertenezcan a $I_1 \bigcup I_2$.

Por contra, la segunda interpretación considera que los ejemplos que pertenecen a $I_1 \bigcup I_2$. son ejemplos positivos tanto para el intervalo I_1 como para el intervalo I_2.

La segunda de las interpretaciones es la más razonable en estos casos, que además coincide con la filosofia original de SLAVE: "Encontrar la mejor descripción que representa a un concepto". Sin embargo, en la propuesta inicial de SLAVE (que se desarrolló en el capítulo anterior) solamente se distinguía entre caso crisp y caso difuso durante el proceso de inferencia. Esta cirscunstancia, provoca que el proceso de aprendizaje utilizado para el caso difuso, mantiene una actitud de competición entre los conceptos, mientras que el proceso de inferencia requiere de la colaboración de las reglas para obtener una salida correcta.

Por consiguiente, ahora nos centraremos en corregir esta deficiencia en la concepción del aprendizaje para el caso difuso, readaptando el algoritmo para que además mantenga información del comportamiento del sistema durante el proceso de inferencia.

3.3.2.1 Aciertos y fallos en el caso difuso

Anteriormente, hemos ilustrado como debe comportarse el algoritmo de aprendizaje cuando existe algún tipo de solapamiento entre los valores de la variable consecuente. Este comportamiento es similar cuando el dominio de la variable consecuente se componen de un conjunto de etiquetas difusas. Al igual que en el caso crisp, bajo estas premisas, podemos establecer los conceptos de acierto y fallo en la clasificación de un concepto.

La segunda interpretación del ejemplo anterior, nos da la idea básica para la descripción de estos dos nuevos conceptos.

Definición 3.7 *Diremos que una regla $R_B(A)$ acierta en la clasificación de un ejemplo e, si y solo si, tanto la descripción de la regla como el consecuente de la misma tiene adaptación con el ejemplo, es decir,*

$$U(e, A) > 0 \text{ y } U(e, B) > 0$$

Definición 3.8 *Diremos que una regla $R_B(A)$ falla en la clasificación de un ejemplo e, si y solo si, la descripción de la regla tiene adaptación con el ejemplo y el ejemplo no tiene adaptación con el consecuente de la misma es decir,*

$$U(e, A) > 0 \text{ y } U(e, B) = 0$$

A partir de las definiciones anteriores, podemos establecer las definiciones de número de aciertos y número de fallos de una regla como la suma de los mismos de manera que dado un conjunto de ejemplos E y una regla $R_B(A)$, definimos el número de aciertos de la regla sobre el conjunto de ejemplos, como el cardinal del conjunto de aciertos, es decir,

$$N_{ac}^{+}(R_B(A)) = |\{e \in E \mid U(e, A) > 0 \text{ y } U(e, B) > 0\}|$$

y definimos el número de fallos de la regla sobre el conjunto de ejemplos, como el cardinal del conjunto de fallos, es decir,

$$N_{fa}^{-}(R_B(A)) = |\{e \in E \mid U(e, A) > 0 \text{ y } U(e, B) = 0\}|$$

Al igual que en la modificación propuesta para el caso crisp, el número de aciertos y errores de las reglas permiten determinar la bondad de la regla. A diferencia de la propuesta anterior, en el caso difuso, no es necesario graduar la confianza del acierto, ya que las reglas no compiten entre sí, sino que colaboran para obtener la salida del sistema.

Para el caso crisp, la modificación de SLAVE considera como una información muy importante el peso de las reglas. En ese contexto, el peso permite determinar la importancia de la regla y entra a formar parte como un elemento competitivo fundamental en el mecanismo de resolución de conflictos. En el caso difuso, este elemento deja de ser relevante, al obtenerse la salida en base a la competición de las distintas reglas aplicables. En esta situación, el elemento importante a considerar del proceso de inferencia es el error que se comete en las salidas del sistema. Este elemento para a formar parte en un segundo término, de la evaluación de la bondad del conjunto de reglas obtenido hasta el momento. De tal forma, que para la evaluación de las reglas se utiliza una EFL cuyo primer criterio se rige por el grado de satisfacción de la consistencia y completitud aplicados a los aciertos y fallos de las reglas, y como segundo criterio el error cometido hasta el momento por las salidas del sistema.

Al igual que sucede con el caso crisp, no es posible aplicar directamente el proceso de inferencia en el mismo sentido en el que se usa en el algoritmo de refinamiento de teorías, ya que sólo se conoce una porción de reglas que describen el sistema. Por esta razón, para determinar la medida del error, únicamente se toman del conjunto de entrenamiento, aquellos ejemplos que tienen alguna adaptación con los conceptos que han sido aprendidos. Por consiguiente y a semejanza del caso crisp, el orden en el cual se aprenden los conceptos es importante, pero a diferencia de este, donde es posible estudiar diferentes ordenes debido a la independencia de los conceptos, en este caso únicamente dos ordenes parecen apropiados, el considerar el orden creciente de las etiquetas, y el decreciente. En SLAVE asumiremos que se usa el orden creciente de las etiquetas.

Asumiendo que las etiquetas difusas se aprenden de forma creciente, y estamos intentando clasificar los ejemplos de la etiqueta t-ésima de la variable consecuente, podemos aplicar el método de inferencia sobre el conjunto de reglas obtenido hasta el momento para conocer el error que se comete sobre aquellos ejemplos que pertenecen a las t primeras etiquetas. Actuando de esta manera, cada vez que se aprende una nueva regla, no sólo se intenta máximizar el número de aciertos, sino que además el algoritmo trata de encontrar la regla que proporcionando el mismo número de aciertos, reduce el error con respecto a los ejemplos asociados a las etiquetas del consecuente que ya han sido aprendidas. Sin embargo, para que el proceso de reducción del error tenga un comportamiento más flexible y permita una mejora sustancial del mismo, es necesario que aquellos ejemplos que tienen pertenecia con una etiqueta de orden inferior a la que se intenta aprender y que tiene adaptación con el antecedente de la regla que se propone no sean considerados como error. Para aclarar este último extremo proponemos el siguiente ejemplo:

Ejemplo 3.7 *Supongamos que en el conjunto de entrenamiento tenemos un ejemplo e que tiene adaptación únicamente con dos etiquetas de la variable consecuente L_1 y L_2. Supongamos además que el algoritmo de aprendizaje obtuvo una regla que lo consideraba como un acierto para la etiqueta L_1, pero sin embargo, para ninguna de las reglas obtenidas de la etiqueta L_2 este ejemplo fué considerado como un acierto. Si el algoritmo ahora intenta aprender una nueva regla para la etiqueta L_3, es posible que si esta nueva regla tiene adaptación con el ejemplo anteriormente mencionado, la diferencia se reduzca entre la salida que propociona el conjunto de reglas y la salida real para este ejemplo. Como el ejemplo no pertenece en ningún grado a la etiqueta L_3, por tanto, dicho ejemplo debería considerarse como un fallo para la nueva regla, pero este hecho ocasiona que la incorporación de ninguna regla pueda reducir el error que produce este ejemplo. Por esta razón, los ejemplos con pertenencia a una etiqueta de orden menor, no se consideran como ejemplos.*

También podría suceder que el error al incluir la nueva regla aumentase. En dicho caso, el propio sistema tiene recursos para intentar obtener otra regla que no recubriera a este ejemplo, a menos que este hecho sea imposible. Por consiguiente, penalizar a estos ejemplos va en perjuicio de la capacidad de representación que es capaz de obtener el algoritmo sobre el sistema que se desea aprender.

En definitiva, la nueva versión de SLAVE para el caso difuso conlleva dos modificaciones sobre la evaluación de la bondad de la regla. La primera de ellas, al igual que la modificación para el caso crisp, se sustituyen los valores de número de ejemplos positivos y número de ejemplos negativos por el número de aciertos y número de fallos en el caso difuso respectivamente para obtener el grado de consistencia débil de la regla. La segunda modificación involucra la inclusión de un nuevo criterio que mide el error cometido en las salidas hasta el momento. La evaluación de este criterio se considera el más importante

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
SIN	0.0016	0.0014	7	26
PARABOLA	0.0030	0.0037	3	12
3DPARABOLA	0.0026	0.0048	7	597
MONTE	0.0021	0.0020	13	971
PENDULO	0.1204	0.1680	6	38
RICE	0.0052	0.0112	7.5	77

Tabla 3.9: Resultados de SLAVE para el caso difuso con la nueva función de evaluación de reglas

en la aplicación de la EFL después del producto entre el grado de consistencia débil y el grado de completitud.

3.3.2.2 Pruebas experimentales

A continuación mostramos las pruebas experimentales realizadas para comprobar el comportamiento del nuevo sistema. En la tabla 3.9 se muestran los resultados obtenidos tras la ejecución de SLAVE sobre las diferentes bases de datos propuestas.

Comparando estos resultados con los obtenidos por el algoritmo WM (tabla 2.6) podemos observar que para las bases de datos seleccionadas la modificación incluida en SLAVE obtiene mejores resultados, es decir, reduce de forma significantiva el error. Además, el número de reglas necesario para representar el conocimiento aprendido es también también menor. Así, si la versión inicial de SLAVE tan sólo mostraba buenos resultados en el caso crisp, la nueva versión del sistema tiene buenos resultados tanto en el caso crisp como en el caso difuso. Obviamente mejora a la versión previa de SLAVE, los resultados son mejores que los obtenidos por otros algoritmos sobre las bases de datos de prueba.

Por otro lado, si comparamos estos resultados con los obtenidos por SLAVE+R (tabla 3.3) no podemos afirmar de forma clara, que uno de ellos tenga un comportamiento mejor que el otro. Sin embargo, si asumimos que la diferencia en cuanto al error cometido entre ambos algoritmos no es significativo, entonces, en este caso, el tiempo de respuesta que ofrece esta versión modificada de SLAVE es significativamente menor.

Por tanto, al igual que se comento anteriormente, tampoco podemos descartar el algoritmo de refinamiento de teorías, como una vía válida para mejorar el comportamiento del conjunto de reglas, también en el caso de problemas con consecuente difuso. Sin embargo, nos fijamos como objetivo para el resto de memoria, mejorar en la medida de lo posible el funcionamiento de esta nueva propuesta en la evaluación de las reglas que se ha incluido en el algoritmo de aprendizaje SLAVE.

Capítulo 4

El problema de la selección de características

4.1 La selección de características

El objetivo de los algoritmos de aprendizaje inductivos consiste en establecer una hipótesis que sea capaz de explicar el comportamiento de un sistema. Los algoritmos clásicos de aprendizaje automático suelen tener asociados al proceso de inducción criterios que tratan de encontrar estructuras no excesivamente complejas que permitan describir correctamente al sistema. Los dos criterios más comunes son los basados en la filosofía *Occam's Razor* [8] o en la obtención de *la descripción de longitud mínima* [133].

Con la inclusión de estos criterios, se intenta obtener un comportamiento ideal del algoritmo inductivo, de manera que este sea capaz de seleccionar el subconjunto de características de entrada que proporcione la mejor descripción del sistema. Encontrar la estructura minimal es en la mayoría de los casos un problema NP-duro. Por tanto, los algoritmos que tratan este problema, son conducidos por funciones heurísticas a través del espacio de las hipótesis posibles.

Abordando el problema de esta manera, la selección de características implica encontrar un conjunto de características "bueno" bajo algún punto de vista y debe ser valorado mediante una función objetivo. Normalmente, estas funciones establecen la bondad en términos del tamaño de las estructuras obtenidas para representar la conducta del sistema.

Por consiguiente, la selección de las características relevantes y la eliminación de las irrelevantes es uno de los problemas centrales de los algoritmos de aprendizaje automático [95]. La detección de las variables irrelevantes conlleva no sólo la reducción del espacio de búsqueda de las posibles descripciones válidas para representar los conceptos, sino además la obtención de soluciones más comprensibles.

Planteada la selección de características como un problema de búsqueda heurística, cada estado en el espacio de búsqueda especifica un subconjunto de variables posibles.

La estructura de este espacio, según se propone en [95], sugiere un método que debe tomar cuatro decisiones básicas para determinar la naturaleza del proceso de búsqueda heurístico.

La primera de ellas, debe determinar el punto de partida en el espacio de búsqueda, que establece la dirección que inicialmente tomará la búsqueda. La segunda de las decisiones está relacionada con la organización de la búsqueda. Dado que una búsqueda exhaustiva del espacio es imposible, debido a la explosión combinatoria del mismo, se han propuesto aproximaciones más aplicables a este problema. Entre ellas, podemos mencionar las técnicas greedy, o métodos de escalada que permiten moverse en el espacio en base a decisiones locales. La tercera determina la estrategia utilizada para evaluar las diferentes alternativas de subconjuntos de variables. Básicamente, podemos considerar dos métodos generales, los métodos por filtro y los métodos por envoltura. Los primeros determinan el conjunto de variables relevantes antes de aplicar el proceso de inducción, mientras que los segundos generan un conjunto de variables candidatas durante el proceso de inducción, y en base a los resultados obtenidos evalúan el conjunto de variables.

La última decisión hace referencia a los criterios que deben ser definidos para detener el proceso de búsqueda, determinando el subconjunto de variables relevantes al problema.

De lo anterior se deduce que la selección de un conjunto minimal de características está estrechamente relacionado con el concepto de relevancia. En la literatura podemos encontrar diversas definiciones de este concepto. Por ejemplo la definición siguiente fue propuesta en [1] y define la relevancia para variables que toman valores booleanos, asumiendo que no existe ruido.

Definición 4.1 *Diremos que una variable X_i es relevante para un determinado concepto C, si X_i aparece en toda fórmula booleana que representa a C. En otro caso, diremos que es irrelevante.*

Frente a la definición anterior, en [87] se define de forma más general el concepto de relevancia, en base a la siguiente afirmación: *Las características son relevantes si sus valores varían sistemáticamente por la pertenencia a una clase.*

Definición 4.2 *Diremos que X_i es relevante si y sólo si existen algunos x_i e y para los cuales $p(X_i = x_i) > 0$ y*

$$p(Y = y|X_i = x_i) \neq p(Y = y).$$

donde x_i es un valor del dominio de la variable X_i, siendo Y la variable concepto e y uno de los valores que puede tomar dicha variable.

Esta definición establece que una variable es relevante si el hecho de conocer el valor que se le asigna cambia la estimación para la variable concepto. En definitiva, la relevancia se relaciona con la dependencia condicionada entre la variable de clasificación y la variable predictiva.

La anterior definición falla en algunas situaciones para obtener la relevancia y se puede mejorar de la siguiente forma:

Definición 4.3 *Sea S_i el conjunto formado por todas las características excepto X_i, es decir, $S_i = X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m$, y sea s_i una asignación de valores para todas las características de S_i.*

Diremos que X_i es relevante si y sólo si existen algunos x_i, s_i e y para los cuales $p(X_i = x_i, S_i = s_i) > 0$ y

$$p(Y = y|X_i = x_i, S_i = s_i) \neq p(Y = y|S_i = s_i).$$

Es decir, X_i es relevante si la probabilidad para alguna de sus posibles asignaciones, dada el resto de las características, puede cambiar cuando eliminamos el conocimiento sobre el valor de la variable X_i.

Esta última definición está propuesta en [87]. En este trabajo se define la relevancia desde dos puntos de vista: la anterior que se denomina *relevancia fuerte* y una nueva definición más relajada de relevancia, denominada *relevancia débil*.

Definición 4.4 *Una característica X_i es débilmente relevante si y sólo si no es fuertemente relevante y existe un subconjunto de características S_i^1 de S_i y algún x_i, s_i^1 e y con $p(X_i = x_i, S_i^1 = s_i^1) > 0$ tal que*

$$p(Y = y|X_i = x_i, S_i^1 = s_i^1) \neq p(Y = y|S_1^1 = s_i^1).$$

La relevancia fuerte implica que la característica es indispensable ya que si no se considera para la descripción del sistema, se perderá capacidad de predicción. Por el contrario, la relevancia débil implica que la característica puede contribuir algunas veces para aumentar la capacidad de predicción.

SLAVE aborda la selección de características manteniendo básicamente la idea propuesta en la definición de relevancia débil, ya que el algoritmo afronta la determinación de la relevancia o irrelevancia de las características desde un punto de vista local, sin tener en cuenta cómo estas características incidieron en las descripciones previamente aprendidas, ni de como incidirán en las que quedan por aprender. Por tanto, en el planteamiento de SLAVE, las variables relevantes se buscan individualmente para cada regla, de manera que, fijado un concepto a aprender, el algoritmo trata de encontrar el menor subconjunto de variables y la mejor asignación de valores a dichas variables para describir dicho concepto.

En comparación con otros modelos genéticos para el aprendizaje de reglas, el modelo iterativo de SLAVE permite obtener respuestas rápidas debido a la reducción del espacio de búsqueda. A pesar de ello, el problema del tiempo de respuesta continúa cuando se tiene un número alto de variables antecedente posibles. En estos casos el espacio de búsqueda sigue siendo excesivamente grande.

El algoritmo genético de SLAVE elimina variables irrelevantes para un valor fijo de la variable de clasificación. Sin embargo, la codificación de las variables irrelevantes permanece en el cromosoma y está expuesta a modificaciones provocadas por los operadores genéticos. Con esta situación, al algoritmo le cuesta un gran esfuerzo evolutivo determinar que una variable es irrelevante. Por el contrario, el esfuerzo genético para interpretar una variable irrelevante como relevante es muy pequeño.

Esta diferencia de esfuerzo se incrementa exponencialmente con el número de valores que constituyen los dominios de las variables, y con el número de variables que intervienen en el problema.

En este capítulo nos planteamos una modificación del algoritmo genético de SLAVE para aumentar su capacidad para determinar variables irrelevantes y mejorar su tiempo de respuesta para encontrar un subconjunto de variables relevantes.

4.2 Modelos de selección de subconjuntos de características

Según la clasificación que propone Langley en [95], atendiendo a la estrategia utilizada para evaluar las distintas alternativas de subconjuntos de características, podemos considerar la existencia de dos modelos: los modelos por filtro (filter model) y los modelos por envoltura (wrapper model).

El modelo por filtro recibe este nombre porque el subconjunto de características irrelevantes para el problema es eliminado de la descripción del mismo antes de aplicar el algoritmo inductivo. Este tipo de modelo tiene una estructura semejante a la que se propone en la figura 4.1. De entre los algoritmos descritos en la literatura que siguen este modelo, podemos destacar a FOCUS y FOCUS-2 [1, 2].

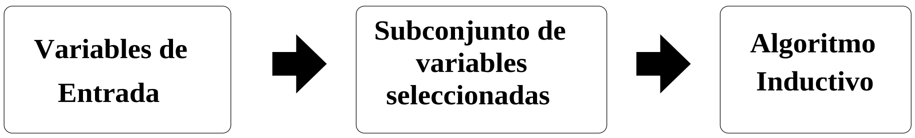


Figura 4.1: Esquema de un modelo por filtro

Frente al modelo por filtro, podemos encontrar otra estrategia de actuación que fija como objetivo del algoritmo inductivo la tarea adicional de determinar el subconjunto de características relevantes, de manera, que la evaluación de las variables seleccionadas se realiza de forma simultánea a la evaluación de la bondad de las soluciones que se desean extraer. El modelo asociado a la utilización de esta estrategia es conocido como modelo por envoltura y presenta una estructura semejante a la propuesta en la figura 4.2. Por tanto, los algoritmos de aprendizaje inductivo que siguen este modelo, son los que en [115] se clasifican como algoritmos que aprenden la relevancia parcial de los atributos, y ya comentamos en el capítulo primero de esta memoria. Por el contrario, los algoritmos que siguen el modelo por filtro son los algoritmos que aprenden la relevancia completa sobre todos los atributos que se proponen en el problema. Podemos destacar de entre los algoritmos que siguen este segundo modelo, OBLIVION [96] que combina este modelo con el método del vecino más cercano.

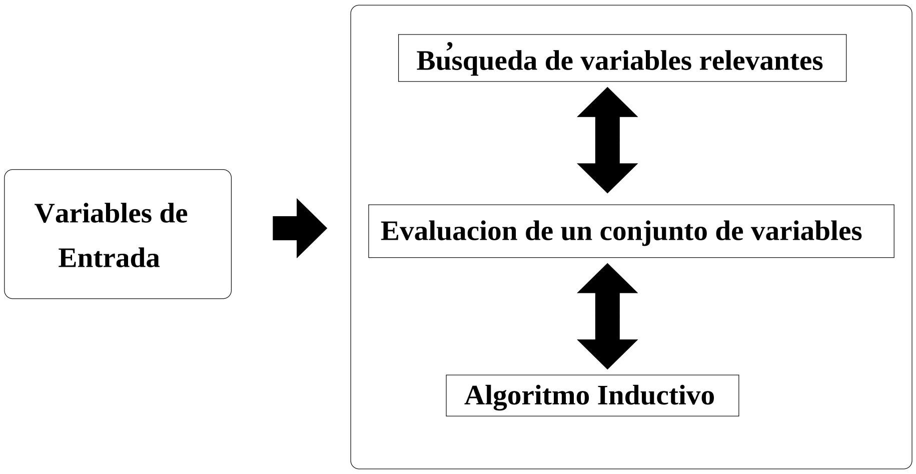


Figura 4.2: Esquema de un modelo por envoltura

Desde nuestro punto de vista, los algoritmos de refinamiento de teorías deberían incluirse como un modelo de selección de características, ya que algunos de ellos tienen como objetivo reducir el número de variables que intervienen en el proceso de clasificación de un concepto. En este sentido, el algoritmo de refinamiento de teorías difusas propuesto en el capítulo 2 de esta memoria, podría considerarse como un modelo de selección de características, ya que trata no sólo de mejorar el comportamiento del sistema en cuanto a su capacidad de predicción, sino que además intenta aumentar la comprensibilidad y simplicidad de las reglas, que son conceptos que involucran la selección de las características relevantes.

La estrategia utilizada por SLAVE en la selección de las variables relevantes para un determinado concepto, está claramente incluído dentro del modelo por envoltura. Debemos destacar, que a diferencia de otros algoritmos que siguen este modelo, SLAVE plantea la relevancia desde un punto de vista temporal, es decir, como una situación puntual que aparece en el proceso de aprendizaje, sin que ello implique nada definitivo para el resto del proceso evolutivo. Esto significa que en un momento dado de la evolución, una variable puede considerarse como irrelevante para un concepto en un individuo concreto de la población. Sin embargo, esta situación no implica que dicha variable sea considerada como irrelevante para el resto del proceso, ya que una simple aplicación de un operador genético puede considerar a dicha variable como relevante en las siguientes iteraciones.

Otra característica de SLAVE es que la bondad del subconjunto de características seleccionadas se valora conjuntamente con la asignación asociada a cada una de estas variables, aplicándose el criterio de simplicidad para tratar de reducir lo más posible dicho conjunto. Por tanto, la selección de características está perfectamente integrada con los criterios utilizados para la evaluación de las reglas.

4.3 Algoritmo genético de dos niveles

Conceptualmente, podemos considerar que SLAVE se propone resolver dos tareas de aprendizaje: determinar el subconjunto de variables que son necesarias para explicar un concepto y encontrar la mejor asignación de valores a dichas variables. Estas dos tareas que están muy relacionadas, en la definición de SLAVE se plantean conjuntamente, de manera, que se utiliza una misma representación y un mismo proceso para ambas. Este hecho conlleva que el proceso de selección de variables se entienda como un efecto colateral de la tarea de asignación de valores a variables, y por consiguiente, el algoritmo centra su esfuerzo en esta segunda tarea.

Como ya se dijo en este capítulo, por la forma de representación y por la conducta de SLAVE, determinar la irrelevancia de una variable requiere un costo evolutivo grande que crece exponencialmente con el número de etiquetas definidas para la variable, y con el número de variables que intervienen en el problema. Este hecho se pone de manifiesto cuando SLAVE se aplica a bases de datos con un número elevado de variables. Por ejemplo, en las pruebas experimentales realizadas hasta el momento, se puede comprobar que SLAVE obtiene un número excesivo de reglas para la base de datos IONOSPHERE, que provoca que la capacidad de predicción del conjunto de reglas obtenido no tenga un comportamiento tan bueno como podría esperarse.

Para resolver este problema, proponemos una nueva definición del algoritmo genético de SLAVE, donde los individuos estan constituidos por dos subestructuras que representan soluciones de una mayor complejidad. Cada subestructura se define mediante una representación apropiada a cada una de las tareas de aprendizaje. Por esta razón, denominaremos al algoritmo genético resultante, algoritmo genético con dos niveles.

Existen trabajos en la literatura que abordan el problema de la búsqueda de soluciones complejas a través de la descomposición de la representación en subestructuras. Sobre cada subestructura se plantea un algoritmo de búsqueda con una heurística propia, mientras que la evaluación de la bondad de la solución se realiza sobre la yuxtaposición de todas las subestructuras. Algunos investigadores han propuesto sistemas de búsqueda que siguen esta filosofía. En concreto, en el campo de los algoritmos genéticos podemos destacar los trabajos [122, 123] donde se describen las ventajas que proporciona esta forma de actuar en los procesos de búsqueda realizados a través de algoritmos genéticos, y en especial, la aplicación de este modelo al problema del aprendizaje. Cuando se utilizan algoritmos genéticos, o en general, estrategias evolutivas, a esta forma de actuar se la conoce como *coevolución*.

Por tanto, un algoritmo genético con dos niveles, mantiene dos representaciones diferentes: una para determinar el subconjunto de variables relevantes para un determinado concepto, y otro para encontrar la mejor asignación entre valores y variables para dicho concepto. Sobre cada representación se establece un proceso de coevolución genética, donde cada nivel tiene sus propios operadores genéticos, pero la bondad de la solución se establece en base a la colaboración entre los dos niveles que componen un individuo, como muestra la figura 4.3.

En la nueva definición del módulo de selección de reglas de SLAVE, distinguimos dos niveles diferentes en el proceso de búsqueda:

- El nivel de variable
- El nivel de valor

El primero de ellos, codifica qué variables de las que inicialmente se proponen en el problema se consideran relevantes para formar parte de la descripción de un determinado concepto. Esta información se va modificando durante el proceso evolutivo. El nivel de valor, codifica los valores particulares, tanto para las variables que se consideran relevantes como para las irrelevantes, aunque el código asociado a las variables no relevantes en este nivel no se considera para establecer la bondad de las reglas.

Esta separación entre ambas tareas de aprendizaje, que las sitúa a un mismo nivel de importancia durante el proceso, puede ser muy importante para problemas que involucran muchas variables, ya que cuando el algoritmo genético obtiene la mejor descripción para un concepto, puede seleccionar un subconjunto de variables y establecer la búsqueda usando esta información. Además, durante el proceso evolutivo los subconjuntos de variables seleccionadas pueden cambiar mediante la inclusión de nuevas variables o la eliminación de algunas de ellas que inicialmente fueron seleccionadas. De esta forma, el esfuerzo que requiere el sistema para seleccionar o eliminar una de las variables, es equivalente al esfuerzo que realiza el algoritmo para añadir o eliminar la asignación de un valor a una variable.

Para definir el algoritmo genético en dos niveles, es necesario describir las tres componentes siguientes:

1. *Una representación genética de las soluciones al problema,*
2. *una forma para crear la población inicial de soluciones, y*
3. *los operadores genéticos definidos para cada nivel.*

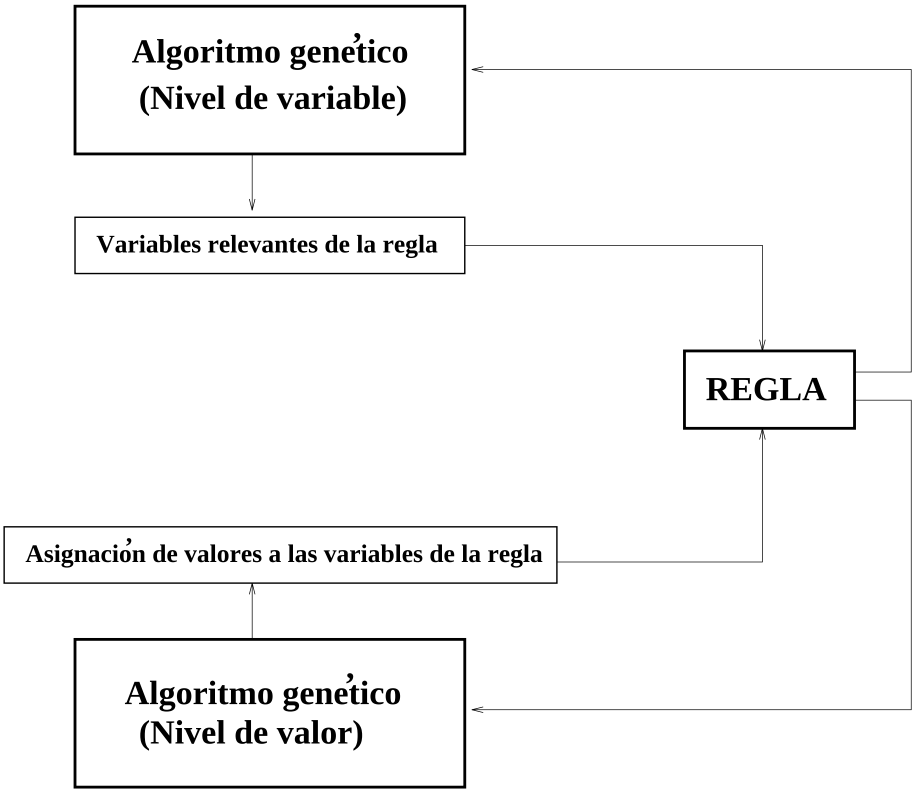


Figura 4.3: Esquema de SLAVE con dos niveles

Unicamente es necesario describir estas tres componente, ya que el modelo de coevolución mantiene la misma función descrita en el capítulo anterior para valorar la bondad de las reglas. Esta función de evalución es válida, ya que valora las mismas propiedades de las descripción propuestas por los individuos de la población, donde ahora, cada individuo esta compuesto por dos estructuras con significado diferente.

Lo mismo sucede con el operador de selección de la nueva población, ya que esta operación se realiza simultaneamente sobre las dos estructuras que componen cada individuo, esto es, la selección se realiza sobre individuos completos.

A continuación describimos más detalladamente las tres componentes que se modifican al realizar esta redefinición del algoritmo genético.

4.3.1 Representación de un elemento de la población

Supongamos que tenemos n posibles variables antecedente $X_1, \ldots, X_n$ y cada una de ellas tiene un dominio difuso asociado D_i con m_i componentes. La codificación genética necesita información sobre las variables seleccionadas y los valores asignados a estas variables. Así, necesitamos una codificación genética con las dos estructuras siguientes:

- Un cromosoma de variable
- Un cromosoma de valor

El cromosoma de variable (VAR) está constituido por un vector binario de longitud n (el número de variables) donde el valor uno indica que dicha variable se selecciona para la descripción de la regla, mientras que el valor cero indica que la variable no se selecciona.

El cromosoma de valor (VAL) usa la codificación binaria implementada en SLAVE, y ya mostrada en el capítulo 2.

$\boxed{\textbf{Ejemplo}}$ **4.1** *Supongamos que tenemos tres variables , X_1, X_2 y X_3, tales que sus dominios difusos asociados son los mostrados en la figura 2.3 del capítulo 2. En este caso el código:*

$$VAR\ 101\ \ VAL\ 1110010111$$

representa al siguiente antecedente:

$$X_1 \ is\ \{A_{11}, A_{12}, A_{13}\} \ and\ X_3 \ is\ \{A_{31}, A_{32}\}.$$

Como X_1 toma todos los valores de su dominio D_1, el antecedente anterior es equivalente a

$$X_3 \ is\ \{A_{31}, A_{32}\}.$$

La variable X_2 fué eliminada de la descripción en el nivel de variable, mientras que X_1 fué eliminada en el nivel de valor.

Un aspecto importante a destacar, es que cuando el estado de una variable cambia de ser seleccionada a no ser seleccionada, el código relativo a dicha variable permanece. La razón es que si este estado cambia de nuevo, la variable tiene un valor concreto, que además coindice con el valor que tenía dicha variable antes de su primer cambio de estado.

4.3.2 La generación de la población inicial

La generación de la población inicial se debe hacer a dos niveles. A nivel de variable, el cromosoma de variable se construye usando una *probabilidad de activación inicial* $p \in [0, 1]$, donde p es la probabilidad de que una variable se seleccione para la descripción de la regla. Para no obtener inicialmente un antecedente excesivamente general, es conveniente que esta probabilidad tome un valor cercano a uno, dejando que sea el proceso evolutivo el que determine la configuración de la estructura más apropiada para el antecedente.

En el nivel de valor, el cromosoma se obtiene seleccionando aleatoriamente un ejemplo del concepto que se desea aprender y obteniendo el antecedente más especifico que mejor lo representa, como ya se mostró en el capítulo 2.

$\boxed{\textbf{Ejemplo}}$ **4.2** *Sean X_1, X_2 y X_3 tres variables cuyos dominios asociados corresponden a los que aparecen el la figura 2.3, y sea (r1,r2,r3) un ejemplo selecionado aleatoriamente del conjunto de entrenamiento para un determinado concepto. El antecedente más específico que mejor lo recubre es*

$$X_1 \text{ is } A_{13} \text{ and } X_2 \text{ is } A_{23} \text{ and } X_3 \text{ is } A_{21}$$

cuya representación binaria es 0010010010. *Tomando 0.9 como valor de la probabilidad de activación inicial, supongamos que obtenemos la configuración 101 para el nivel de variable. Por tanto, la codificación genética es*

$$VAR\ 101\ \ VAL\ 0010010010.$$

Este proceso se repite para un número fijo de ejemplos seleccionados aleatoriamente para confeccionar la población inicial.

4.3.3 Operadores genéticos

El modelo de coevolución que se propone para resolver este problema, define un conjunto de operadores genéticos para cada una de las subestructuras que constituyen un individuo de la población. Por tanto, en este caso, nos encontramos con dos conjuntos de operadores genéticos, uno para el nivel de variable y otro para el nivel de valor.

Los operadores del nivel de valor coinciden con los propuestos en la versión de SLAVE presentada en los capítulos anteriores. Por consiguiente, únicamente nos queda describir los operadores del nivel de variable.

Este nivel mantiene una codificación binaria, para la cuál definimos los dos operadores genéticos estandar, es decir, la operación de mutación y la operación de cruce sobre dos puntos.

Estos conjuntos de operadores realizan su función paralelamente sobre ambas estructuras. Esto implica que la aplicación de los operadores genéticos se realiza de forma independiente entre los dos niveles.

4.4 Pruebas experimentales

Con la introducción de un algoritmo genético de dos niveles, tratamos de mejorar el tiempo de respuesta en la búsqueda de las soluciones parciales. Con este fin, probamos el comportamiento del algoritmo sobre las bases de datos crisp y los parámetros del algoritmo que se han empleado en las experimentaciones anteriores. La tabla 4.1 muestra los

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.47	95.80	4.2	50
MONK1	100	100	4.0	43
MONK2	94.67	83.19	15.2	109
MONK3	99.23	92.64	7.2	50
INFARTO	99.10	84.05	4.8	47
WINE	99.49	95.19	6	153
PIMA	85.45	76.56	17	2661
IONOSPHERE	97.22	90.91	9	1228
SOYBEAN	97.10	97.5	34	954

Tabla 4.1: Resultados con la nueva versión de SLAVE

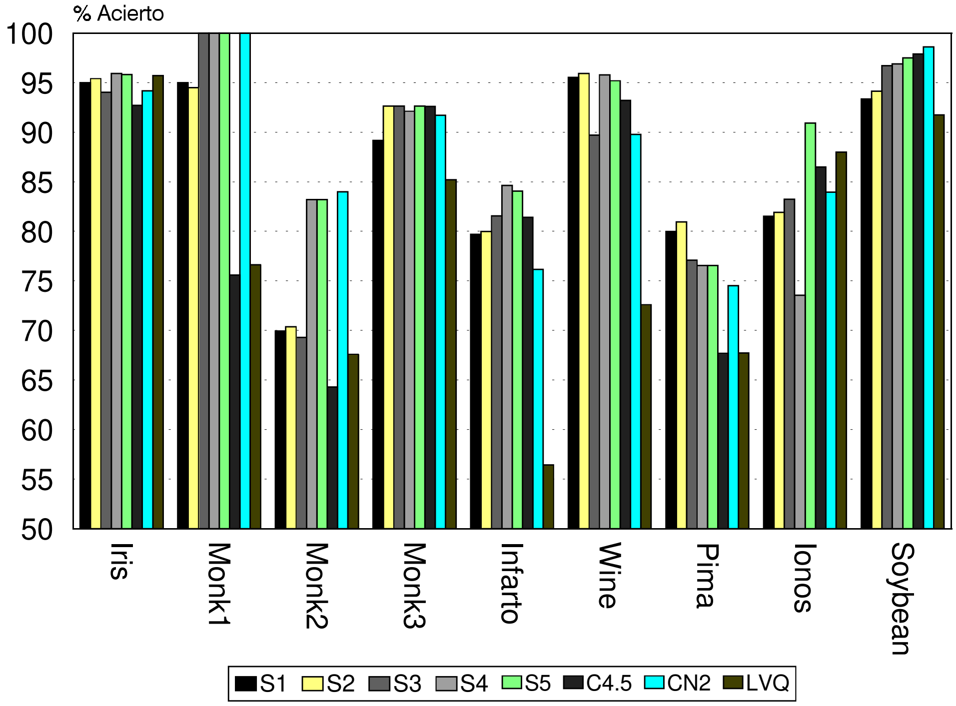
resultados que obtiene este algoritmo, asumiendo que la probabilidad de mutación y la probabilidad de cruce del nivel de variable coinciden con las definidas sobre el nivel de valor, que el valor de la probabilidad de activación definida para generar la población inicial es 0.9 y que el resto de parámetros, son los definidos para los anteriores experimentos.

De dicha tabla, podemos destacar que para las bases de datos que contienen pocas variables, los resultados obtenidos son semejantes a los que obtienen en la versión anterior de SLAVE, como muestra la figura 4.4, e incluso el tiempo de respuesta del algoritmo no varía sustancialmente. Sin embargo, para las bases de datos con un número mayor de variables, se producen mejoras significativas. Por ejemplo, se produce una reducción considerable en el tiempo de respuesta del algoritmo para las bases de datos WINE, PIMA e IONOSPHERE. Además, también se reduce el número de reglas obtenidas para las bases de datos PIMA e IONOSPHERE, que pasa en la primera de 31 a 17 y en la segunda de 53.6 a 9. Conviene destacar el aumento en el porcentaje de acierto obtenido sobre la base de datos IONOSPHERE que pasa de un 73.53 % a un 90.91%, que es la de mayor número de variables.

Por consiguiente, la descomposición de la búsqueda de las soluciones utilizando dos algoritmos de búsqueda que, de forma paralela, tratan de encontrar una solución parcial de segundo orden, parece una estrategia interesante que obtiene los mejores resultados cuando el problema planteado tiene un número elevado de variables.

4.5 Algoritmo genético de dos niveles con información

Los resultados experimentales de la sección anterior muestran que la aplicación de la coevolución a los dos niveles definidos para la representación de las soluciones provoca



S1	Algoritmo SLAVE descrito en el capítulo 2
S2	Algoritmo SLAVE descrito en el capítulo 2, incluyendo nuevos operadores genéticos
S3	Algoritmo SLAVE+R
S4	Algoritmo SLAVE descrito en el capítulo 3
S5	Algoritmo SLAVE dos niveles sin información

Figura 4.4: Comparativa del acierto en los conjuntos de prueba (5)

mejoras en el comportamiento del algoritmo de aprendizaje. Más que en la capacidad de predicción, esta mejora se produce en el tiempo de respuesta del algoritmo de aprendizaje. Además, dicho tiempo mejora en comparación con las definiciones anteriores del algoritmo cuanto mayor sea el número de variables implicadas en el proceso de aprendizaje. Por consiguiente, parece que este algoritmo es apropiado para aprender grandes bases de datos, donde muchas variables están involucradas en la descripción del problema, y posiblemente muchas no sean relevantes.

Si nos detenemos en el proceso de generación de la población inicial de soluciones, podemos observar que para la configuración del nivel de valor, el sistema selecciona un ejemplo del conjunto de entrenamiento. Por consiguiente, el sistema aprovecha información sobre el problema, para generar este nivel. Por el contrario, en la configuración del nivel de variable, el algoritmo no contempla información sobre el problema que intenta resolver, y el proceso se realiza de forma aleatoria, en base a una probabilidad de activación inicial. En este sentido, podemos entender que el algoritmo, al generar por primera vez el nivel de variable, no aprovecha cierto conocimiento que se puede extraer del conjunto de entrenamiento. La adquisición de este conocimiento puede ser importante, ya que desde el principio del proceso evolutivo, sitúa el conjunto de soluciones iniciales en un camino más cercano a las posibles soluciones deseables.

En un principio, la incorporación de información extraída del conjunto de entrenamiento, puede reducir aún más el tiempo de respuesta del algoritmo. Este hecho provoca que la convergencia hacia las soluciones deseables se realice más rápidamente.

También es posible que, en función del tipo de información utilizada, esta sea útil, no sólo para la generación de la población inicial, sino que sea aplicable durante el proceso evolutivo. Si la información establece una valoración a priori de cada variable para determinar si debe o no formar parte de la descripción de la regla para un determinado concepto, está puede considerarse como parte constituyente de cada individuo y evolucionar para determinar el subconjunto de variables necesarias en dicha descripción.

A continuación, proponemos la incorporación al algoritmo genético de dos niveles de información obtenida sobre el conjunto de entrenamiento. Esta información permitirá determinar la relevancia de las variables predictivas. La interpretación de la información nos lleva a dos modelos.

4.5.1 Medidas de información

Estamos interesados en usar una medida de la relevancia inicial de las variables para determinar un concepto particular. Esta medida determinará el grado de influencia de cada variable para representar el valor de cada clase. En un segundo paso, el proceso genético, usando un conjunto de operadores, alterará este valor, y por tanto, modificará el criterio para determinar el conjunto de variables que deben activarse en la representación de cada clase.

Usamos la siguiente medida de información[93, 161]

$$I(X,Y) = \sum_x \sum_y p(x,y)\ log_2(\frac{p(x,y)}{p(x)p(y)})$$

donde p es una medida de probabilidad.

Esta medida define un rango diferente para cada variable. Para facilitar la comparación de los resultados en todas las variables, es necesario que todas las medidas trabajen en un mismo rango. Para este propósito, usamos la siguiente normalización

$$\tau(X,Y) = \frac{I(X,Y)}{H(X,Y)}$$

donde H(X,Y) es la entropía de Shannon sobre dos variables, definida como

$$H(X,Y) = \sum_x \sum_y p(x,y)\ log_2 p(x,y).$$

La medida estima la dependencia entre las variables X e Y en el sentido siguiente: valores de $\tau(X,Y)$ cercanos a cero indican un alto grado de independencia entre ambas variables, mientras que valores cercanos a uno, muestran un alto grado de dependencia funcional entre ellas.

En el algoritmo de aprendizaje de SLAVE, se seleccionan reglas fijando un concepto concreto. Por tanto, necesitamos restringir la medida anterior para el concepto que se desea aprender, y definimos

$$\tau_C(X) = \frac{I(X, Y=C)}{H(X, Y=C)}$$

donde X es una variable predictiva e Y es la variable de clasificación, siendo C un concepto particular o clase.

De forma similar a $\tau(X,Y)$, $\tau_C(X)$ determina la dependencia o independencia de la variable X para representar el concepto C de la variable de clasificación. En el algoritmo genético interpretamos este valor como una probabilidad inicial de activación de la variable, es decir, usamos $\tau_C(X)$ como la probabilidad con la que la variable X debe ser considerada como relevante para el concepto C, y $1-\tau_C(X)$ como la probabilidad con la que la variable X no debe tenerse en cuenta para dicho concepto.

El hecho de que los dominios de las variables estén compuestos por etiquetas lingüísticas, obliga a establecer un modelo de cálculo para estas probabilidades. En [54] se propone una metodología general para el cálculo de creencias usando información difusa. Con esta formulación, la probabilidad de que la variable X tome un valor a_i de su dominio $\{a_1, a_2, \ldots, a_t\}$ se define como

$$p(X=a_i) = \frac{1}{m} \sum_{j=1}^{m} (\frac{\mu_{a_i}(e_j)}{\sum_t \mu_{a_t}(e_j)})$$

donde m es el número de ejemplos del conjunto de entrenamiento E, e_j es un ejemplo de dicho conjunto de entrenamiento y μ_w es la función de pertenencia a un conjunto difuso w. La fórmula anterior asume que todos los ejemplos del conjunto de entrenamiento son no difusos. La probabilidad bidimensional se define de forma similar a la fórmula anterior, pero requiere la información de dos variables que serán combinadas a través de una t-norma

$$p(X = a_i, Y = b_j) = \frac{1}{m} \sum_{k=1}^{m} \left(\frac{\mu_{a_i}(e_k) * \mu_{b_j}(e_k)}{\sum_{t,h} \mu_{a_t}(e_k) * \mu_{b_h}(e_k)} \right)$$

donde el dominio de la variable Y es $\{b_1, b_2, \ldots, b_r\}$. Desde ahora, asumiremos que la t-norma utilizada para las fórmulas anteriores es el mínimo.

4.5.2 Añadir las medidas de información a SLAVE

Como anteriormente describimos, la modificación consiste en encontrar una forma de asignar los valores iniciales de relevancia a cada una de las variables predictivas, dependiendo de la información que proporciona el conjunto de entrenamiento. El proceso evolutivo determinará los valores de relevancia más apropiados para cada variable. En la sección anterior, se definieron las medidas de información que se usarán como probabilidades de activación.

La representación de los individuos de la población del nuevo algoritmo genético es similar a la que ya se propuso anteriormente. Por tanto, la representación consta de dos niveles (un nivel de variable y un nivel de valor) para todos los individuos de la población. Sin embargo, el nivel de variable de un elemento de la población estaba compuesto por un vector binario, donde el valor 1 en la variable X implicaba la activación de dicha variable y 0 la no consideración de la variable. Ahora, usaremos un nivel con codificación real en el cuál el nivel de variable contiene los valores τ_C, que se mueven en el intervalo [0,1]. Este valor se interpretará como la probabilidad inicial de activación de cada variable.

Podríamos transformar la probabilidad de activación obtenida en base a la información contenida en el conjunto de entrenamiento en un vector binario, y usarlo en la misma forma que se propuso en la versión anterior. Sin embargo, deseamos que los cambios de relevancia o irrelevancia de una variable se realicen de forma suave, es decir, durante la ejecución, el proceso evolutivo debe definir la tendencia hacia la relevancia o irrelevancia de cada variable.

Además, debemos destacar una ventaja fundamental que aporta la obtención de las medidas de información sobre el nivel de valor. Supongamos que X es una variable y x es uno de los valores que puede tomar dicha variable, y sean Y e y la variable de clasificación y un valor de dicha variable respectivamente. Si al observar el conjunto de entrenamiento resulta que no aparece ninguna ocurrencia donde simultáneamente aparezca que $X = x$ e $Y = y$, esto nos indica que el valor x de la variable X no es útil para determinar el concepto y. En este sentido, utilizamos la probabilidad bidimensional para determinar los valores de cada variable que no aparecen para los conceptos particulares que se desean aprender. Los valores de las variables que verifican tener probabilidad conjunta cero sobre algún concepto de la variable de clasificación, pueden ser eliminados y por tanto no se tienen en cuenta para encontrar la descripción de dicho concepto. El hecho de no considerar estos valores implica una reducción del nivel de valor, y por consiguiente una reducción del espacio de búsqueda.

Al optar por definir un nivel de variable con codificación real que mantiene las probabilidades de activación de cada variable, nos encontramos con la necesidad de articular algún mecanismo para determinar si una variable se considera activa o no en la evolución de una regla. De entre las diversas posibilidades que se pueden sugerir, optamos por definir dos formas alternativas de manejo de dicha información para su estudio. A estas dos alternativas las denominamos:

- **Aproximación por copia**

  En la primera alternativa, la relevancia de una variable para determinar su utilidad en la representación de un concepto depende tan solo de su probabilidad de activación. En la generación de la población inicial, el nivel de variable se inicializa con los valores τ_C para cada variable. El nivel de valor se genera al igual que las versiones anteriores, tomando aleatoriamente un ejemplo de dicho concepto del conjunto de entrenamiento.

  La diferencia fundamental se produce en la evaluación del antecedente. En este caso, dependiendo de la probabilidad individual de activación, algunas variables se seleccionan para la descripción de la regla, y se crea un nivel virtual con un nivel de variable binario, es decir, cada individuo, mantiene sus dos niveles, y durante el proceso de evaluación se le añade un nivel más que almacena qué variables fueron consideradas como relevantes y cuales no. El proceso para determinar si una variable X_i estará en la descripción de la regla para el concepto C se realiza de la siguiente forma:

  Esquema 4.1 *Proceso de selección de variables activas en la aproximación por copia*

  1. *Se genera aleatoriamente un valor t en el intervalo [0,1]*
  2. *Si $t \leq \tau_C(X_i)$ entonces la variable X_i se considerará para la evaluación de la regla. En otro caso, esta variable no se tendrá en cuenta en la evaluación de la regla.*

  Sin embargo, la codificación real original del nivel de variable se conserva, y es esta la utilizada para el proceso evolutivo, es decir, la copia binaria que se genera sobre el nivel de variable únicamente se usa para la evaluación de la descripción.

Ejemplo **4.3** *Supongamos que tenemos tres variables, X_1, X_2 y X_3; los dominios difusos asociados con cada una de ellas se muestran en la figura 2.3. Supongamos además que los valores de información obtenidos para el concepto C de la variable de clasificación son:*

$$\tau_C(X_1) = 0.5,\ \tau_C(X_2) = 0.7,\ \tau_C(X_3) = 0.1$$

donde además, se ha calculado que

$$p(X_1 = A_{11}, Y = C) = 0,\ p(X_2 = A_{24}, Y = C) = 0,$$

$$p(X_3 = A_{25}, Y = C) = 0.$$

El proceso de evaluación necesita un nivel virtual de variable binario. Supongamos que, aplicando el esquema anterior, se ha obtenido la configuración 110. Con esta información, tenemos la siguiente configuración:

$$VAR\ (0.5, 0.7, 0.1)\ VAR(110)\ VAL\ ((01)(110)(11))$$

y para la evaluación consideraremos la siguiente descripción:

$$X_1\ is\ \{A_{13}\}\ and\ X_2\ is\ \{A_{21}, A_{22}\}.$$

- **Aproximación por umbral de activación**

  Esta segunda aproximación propone la codificación de un umbral de activación dentro del nivel de variable. Con esta nueva representación de los elementos de la población, una variable X_i se considerará como relevante para la construcción de la descripción de una regla, si $\tau_C(X_i) \geq \tau_{umbral}$, donde τ_{umbral} representa el umbral de activación de las variables del j-ésimo elemento de la población. En otro caso, la variable no se tendrá en cuenta en la descripción de la regla.

  Inicialmente, el umbral de activación se define aleatoriamente para cada elemento de la población, donde τ_{umbral} toma un valor aleatorio en el intervalo

  $$[\min_i \tau_C(X_i), \max_i \tau_C(X_i)]$$

  Los valores τ_{umbral} son modificados por los operadores genéticos durante el proceso evolutivo como el resto de los componentes del nivel de variable.

Ejemplo **4.4** *Utilizando las mismas variables del ejemplo anterior, el código:*

$$VAR\ (0.5, 0.7, 0.1)\ \tau_{umbral}\ 0.6\ VAR(110)\ VAL\ ((01)(110)(11))$$

representa el siguiente antecedente

$$X_2 \ is \ \{A_{21}, A_{22}\}.$$

La modificación del nivel de variable que pasa de tener codificación binaria a tener codificación real, obliga a la inclusión de nuevos operadores genéticos. Para este nivel incluimos dos operadores genéticos, uno para la mutación y otro para el cruce, que han sido ampliamente utilizados en la construcción de algoritmos genéticos con codificación real. Los operadores genéticos son la mutación no uniforme [106], que altera un gen con una cierta probabilidad que varía a medida que la evoluación del genético avanza, y el operador de cruce BLX_α [35], que permite la combinación de dos cromosomas para generar un nuevo cromosoma.

Como consideración final, diremos que los valores de τ_C se recalculan cada vez que el algoritmo de aprendizaje lanza un algoritmo genético para encontrar la mejor descripción, ya que debido al comportamiento de SLAVE, cuando se encuentra una regla para un concepto, los ejemplos del conjunto de entrenamiento que son cubiertos por dicha regla, se eliminan del mismo y por consiguiente la información que contiene el conjunto de entrenamiento después de este proceso cambia.

4.6 Pruebas experimentales

La inclusión en el nivel de variable de la información que aportan los ejemplos del conjunto de entrenamiento sobre la incidencia de cada una de las variables para representar un determinado concepto, permite la generación de una población inicial que se encuentra más cerca de las soluciones que se desean obtener. Este hecho, combinado con la reducción del espacio de búsqueda en el nivel de valor debido a la eliminación de aquellos valores de las variables que no tienen valor cero de probabilidad conjunta con el concepto que se aprende, debería tener como consecuencia una mejora en el tiempo de respuesta del algoritmo de aprendizaje. En esta parte experimental, estudiaremos el comportamiento de las dos aproximaciones definidas anteriormente que manejan dicha información para comprobar que se mejora el tiempo necesario en el aprendizaje de la base de reglas.

En la primera parte de esta sección nos centraremos exclusivamente sobre las bases de datos con consecuente crisp, ya que las bases de datos con consecuente difuso tiene un número pequeño de variables antecedente, y los resultados obtenidos por estas nuevas versiones son semejantes a los obtenidos en el capítulo anterior. Utilizando como parámetros de entrada los ya definidos en los anteriores experimentos. Además, en esta experimentación debemos incluir dos parámetros adicionales que son $\alpha = 0.05$ para el cruce BLX_α y una probabilidad de mutación sobre el nivel de variable que será de 0.001. En la tabla 4.2 se muestran los resultados obtenidos utilizando la aproximación por copia, mientras que la tabla 4.3 muestra los resultados obtenidos con la aproximación por umbral de activación.

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.82	96.12	4.2	40
MONK1	100	100	4.0	37
MONK2	94.67	83.19	15.2	109
MONK3	99.23	92.63	7.2	50
INFARTO	99.10	84.05	4.8	47
WINE	99.62	95.55	6	104
PIMA	85.69	75.00	21	2064
IONOSPHERE	96.22	93.12	9	791
SOYBEAN	96.52	99.45	33	1056

Tabla 4.2: Resultados de SLAVE dos niveles con copia

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	97.51	96.21	4	27
MONK1	100	100	4.0	30
MONK2	94.67	82.73	14.8	95
MONK3	99.50	95.32	7.6	46
INFARTO	99.72	86.12	3.7	27
WINE	98.50	96.76	5.2	41
PIMA	82.81	76.87	17	1107
IONOSPHERE	94.89	92.88	4	485
SOYBEAN	97.50	99.65	27	116

Tabla 4.3: Resultados de SLAVE dos niveles con umbral de activación

Estos resultados prueban de forma efectiva que las suposiciones descritas anteriormente son ciertas, ya que para ambas aproximaciones se puede observar una considerable reducción del tiempo de respuesta del sistema, aunque está es más patente en las resultados de la aproximación por umbral de activación.

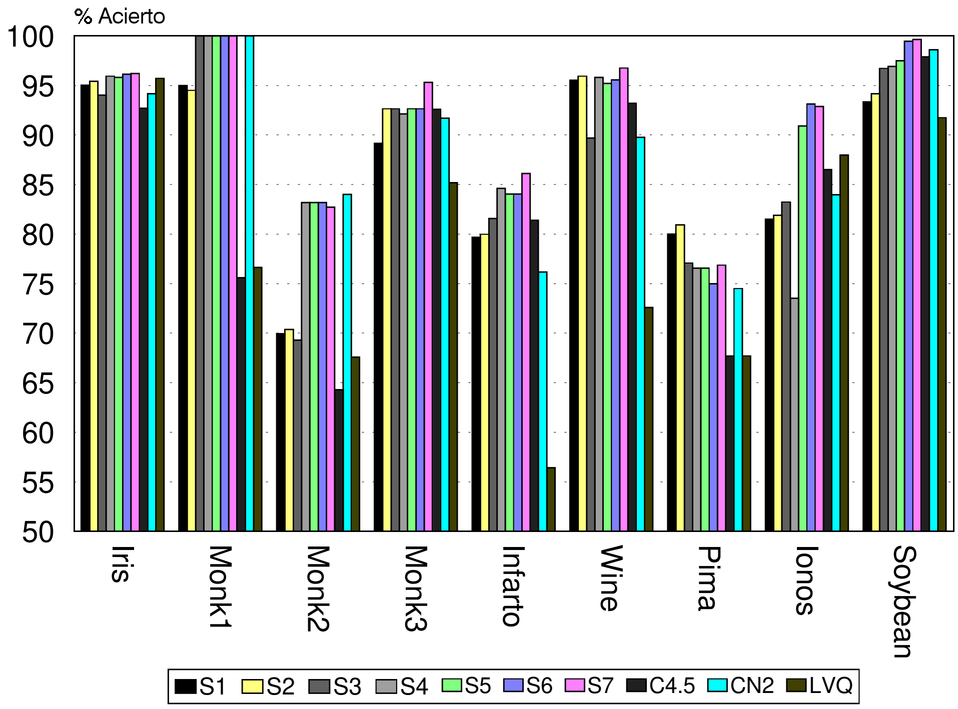
Sobre la capacidad de predicción, podemos observar que aparece un ligero empeoramiento sobre los conjuntos de entrenamiento. Sin embargo, los resultados sobre los conjuntos de prueba son muy parecidos a los mejores obtenidos por las diferentes versiones de SLAVE, e incluso, en algunos casos estos son mejores. Además, tampoco podemos observar diferencias apreciables entre el número de reglas que estas dos aproximaciones obtienen sobre la anterior experimentación con SLAVE. Sin embargo, son destacables los resultados obtenidos sobre la base de datos IONOSPHERE, ya que para ambas alternativas se mejora la predicción sobre el conjunto de prueba y el número de reglas obtenido por la aproximación por umbral de activación es sorprendentemente pequeño en comparación con los resultados anteriores.

Con la prueba experimental que se ha desarrollado en este apartado, no existe una evidencia clara de que una aproximación se comporte mejor con respecto a la otra en términos del porcentaje de aciertos sobre los conjuntos de entrenamiento y prueba. Sin embargo, con respecto al tiempo de respuesta, la aproximación por umbral de activación sí presenta un comportamiento claramente mejor. Por esta razón, incluimos esta aproximación como componente de la nueva versión de SLAVE.

En la figura 4.5 se muestran los resultados medios obtenidos por las distintas versiones de SLAVE propuestas en esta memoria, así como los resultados de los tres algoritmos de aprendizaje seleccionados para compara su comportamiento. Podemos observar que la versión de SLAVE dos niveles por umbral de activación obtiene los mejores resultados en porcentaje de acierto sobre los conjuntos de prueba de todas las bases, excepto sobre la base de datos IONOSPHERE.si los comparamos con las versiones anteriores de SLAVE propuestas en esta memoria. Además, si la comparamos con los otros algoritmos de aprendizaje, también podemos observar que obtiene el mejor resultado para todas las bases de datos, siendo únicamente superada por el algoritmo CN2 en la base de datos MONK2.

Los resultados presentados ponen de manifiesto que la utilización de la coevolución como planteamiento general para los procesos de búsqueda en los algoritmos de aprendizaje son una alternativa válida, en el sentido de que mantienen la capacidad de predicción y mejoran sustancialmente el tiempo de respuesta del algoritmo. Además, este tiempo de respuesta del algoritmo mejora aún más cuando se utiliza la información que aporta el conjunto de ejemplos de entrenamiento.

Si bien SLAVE obtiene prácticamente el mejor resultado para todas las bases de datos propuestas, ahora estamos interesados en comprobar si este buen comportamiento se mantiene ante bases de datos que contienen un número elevado de variables donde muchas de ellas son irrelevantes. Dado que es difícil encontrar bases de datos con estas características, planteamos una nueva experimentación sobre algunas de las bases de datos ya



S1	Algoritmo SLAVE descrito en el capítulo 2
S2	Algoritmo SLAVE descrito en el capítulo 2, incluyendo nuevos operadores genéticos
S3	Algoritmo SLAVE+R
S4	Algoritmo SLAVE descrito en el capítulo 3
S5	Algoritmo SLAVE dos niveles sin información
S6	Algoritmo SLAVE dos niveles por copia
S7	Algoritmo SLAVE dos niveles por umbral de activación

Figura 4.5: Comparativa del acierto en los conjuntos de prueba (6)

		IRIS+	SOYBEAN+
SLAVE	Prueba	93.58	99.53
	N. Reglas	7.8	28.7
	Tiempo	132	1553
CN2	Prueba	83,04	92.13
	N. Reglas	80	173
	Tiempo	272	5660
C4.5	Prueba	92.56	97.9
	Tiempo	19	155
LVQ	Prueba	92,89	90.68
	Tiempo	175	2477

Tabla 4.4: Resultados obtenidos sobre las nuevas bases de datos con consecuente crisp

utilizadas, incluyendo un número elevado de variables irrelevantes.

Con este proposito, seleccionaremos dos bases de datos con consecuente crisp (IRIS y SOYBEAN) y una con consecuente difuso (MONTE) y generaremos las tres siguientes base de datos:

- **IRIS+**: Esta base de datos contiene las cuatro variables predictivas originales de la base de datos IRIS a las que se le añaden 95 variables. La nuevas variables son continuas y toman valores aleatorios en el intervalo [0,1]. Los conjuntos de ejemplos se generan de la siguiente forma: por cada ejemplo que aparece en la base de datos original se generan diez copias del mismo en la nueva base de datos de manera que, las variables originales conservan sus valores originales mientras que el resto de variables toman un valor aleatorio para cada una de las copias. De esta forma, la base de datos IRIS+ contiene 99 variables predictivas, y diez veces más ejemplos que la base de datos IRIS.

- **SOYBEAN+**: Esta base de datos se genera de forma semejante a la base de datos IRIS+ pero incluyendo 50 variables irrelevantes más sobre las 35 que posee el conjunto de ejemplos original. Por consiguiente, SOYBEAN+ contiene 85 variables predictivas y diez veces más ejemplos que SOYBEAN.

- **MONTE+**: La generación de esta base de datos es semejante a la anteriormente propuesta, incluyen en este caso 10 nuevas variables y generando tres copias por cada ejemplo del conjunto original.

Además, debemos especificar que la generación de los conjuntos de entrenamiento y prueba de estas nuevas bases de datos se generan a partir de los conjuntos de entrenamiento y prueba ya definidos sobre cada una de las bases de datos originales y utilizados en la parte experimental de esta memoria.

En la tabla 4.4 se muestran los resultados obtenidos en esta experimentación para las bases de datos IRIS+ y SOYBEAN+. Podemos observar que SLAVE continua obteniendo los mejores resultados en porcentaje de acierto sobre los conjuntos de prueba. Este hecho pone de manifiesto que el algoritmo SLAVE dos niveles por umbral de activación tiene un comportamiento robusto ante la presencia de variables irrelevantes. Por ejemplo, en una de las ejecuciones realizadas sobre la base de datos IRIS+ se obtuvo el siguiente conjunto de reglas:

```
 SI
       NS55 es {Alto}
       NS58 es {Bajo,Medio}
       Sepal_l es {Alta}
       Sepal_w es {Baja,Media}
       Petal_l es {Media,Alta}
       Petal_w es {Media}
   ENTONCES
       Clase es Versicoluor
--------------------------------------

 SI
       NS06 es {Bajo,Medio}
       Petal_w es {Media}
   ENTONCES
       Clase es Versicoluor
--------------------------------------

 SI
       Petal_l es {Muy_Baja,Baja}
       Petal_w es {Muy_Baja}
   ENTONCES
       Clase es Setosa
--------------------------------------
  SI
       Sepal_w es Menor o Igual que {Alta}
       Petal_l es {Alta,Muy_Alta}
       Petal_w es Mayor o Igual que {Media}
   ENTONCES
       Clase es Virginica
--------------------------------------

 SI
       Sepal_w es Menor o Igual que {Alta}
```

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
MONTE	0.0019	0.0022	17	2401

Tabla 4.5: Resultados obtenidos sobre la base de datos MONTE+

```
    Petal_l es {Baja,Media}
    Petal_w es {Baja,Media}
  ENTONCES
    Clase es Versicoluor
--------------------------------------
```

En esta base de reglas, únicamente aparecen variables irrelevantes afectando en un grado mímino a las dos reglas más especificas (las variables irrelevantes incluidas artificialmente son NS06 NS55 y NS58), lo que refleja el buen comportamiento del proceso de selección de características incluido en SLAVE.

Otro hecho que pone en evidencia la robusted de SLAVE en la selección de variables con respecto a CN2 lo refleja el número de reglas. La diferencia entre el número de reglas obtenidos por estos dos algoritmos es significativa. Sin embargo, el hecho más significativo surge en la comparación de los tiempos de respuesta. Si bien, para todas las versiones anteriores, el tiempo de respuesta de SLAVE era ampliamente peor que los que proporcionan C4.5, CN2 y LVQ, cuando el número de variables es muy elevado el tiempo de respuesta de SLAVE mejora en comparación con estos algoritmos. En concreto, en las bases de datos propuestas, SLAVE obtiene mejor tiempo de respuesta que CN2 y que LVQ.

Este buen comportamiento de SLAVE es aplicable también a problemas con consecuente difuso, como muestra el resultado obtenido sobre la base de datos MONTE+ en la tabla 4.5 [1].

Como conclusión final de este capítulo podemos afirmar, en base a las pruebas experimentales realizadas, que con esta versión de SLAVE se satisfacen los objetivos propuestos en este trabajo. En el siguiente capítulo nos planteamos dotar a SLAVE de una mayor capacidad de aproximación al comportamiento real de los sistemas. Con tal objetivo, se propone un nuevo modelo de regla que incluye modificadores semánticos que permiten modificar la semántica de las etiquetas iniciales, de manera que las descripciones que se obtengan se acerquen lo más posible a la información que aportan los ejemplos.

[1] No comparamos estos resultados con los obtenidos por el algoritmo WM ya que la implementación que tenemos de dicho algoritmo no soporta el número de variables antecedentes que posee esta base de datos.

Capítulo 5

Aprendizaje de reglas con modificadores semánticos

5.1 Modelos difusos descriptivos y aproximativos

Uno de los principales problemas con los que nos encontramos cuando utilizamos un algoritmo de aprendizaje consiste en establecer una discretización correcta de las variables que se mueven sobre referenciales continuos. Cuando esta discretización de los referenciales continuos de las variables se realiza mediante conjuntos difusos, diremos que el algoritmo de aprendizaje trabaja con modelos difusos.

Con respecto a la forma en la que se construyen los dominios de estas variables, podemos considerar dos líneas de actuación diferentes en los algoritmos de aprendizaje sobre modelos difusos:

i) aquellos que requieren que los dominios de las variables estén previamente definidos, es decir, que se conozca a priori el conjunto discreto de valores que puede tomar una variable, y

ii) aquellos que durante el proceso de aprendizaje establecen el dominio de las variables.

Los primeros son conocidos como modelos difusos descriptivos (MDD), ya que el conocimiento que se extrae de un determinado sistema se expresa exclusivamente en términos de los valores fijados a los dominios previamente a la ejecución del proceso de aprendizaje. Los segundos, son conocidos como modelos difusos aproximativos (MDA), donde el algoritmo de aprendizaje tiene dos tareas: encontrar los valores apropiados que deben tomar las variables y en función de estos valores extraer el conocimiento que permita describir al sistema.

Los MDD trabajan sobre espacios de búsqueda finitos, lo que conlleva que la capacidad de representación del sistema que se desea aprender esté limitada por los dominios definidos sobre las variables. Además, si se desea que el conocimiento extraído de un sistema sea fácilmente interpretable por una persona que trabaje con dicho sistema, es necesario que los dominios sean definidos por un experto. Por el contrario, los MDA no requieren la participación de un experto, y en este sentido, los algoritmos son más autónomos. Esta automonía del experto provoca que el conocimiento extraído sea difícil de interpretar desde un punto de vista humano, en la mayoría de los casos. Además, los MDA trabajan sobre espacios de búsqueda que en principio son infinitos, lo cuál requiere un esfuerzo mayor durante el proceso de aprendizaje.

En base a las características anteriores, podemos considerar que los MDD son apropiados para problemas donde se requiere que el conocimiento obtenido sea fácilmente interpretable, mientras que los MDA son apropiados para problemas donde lo fundamental es encontrar una buena representación del comportamiento del sistema, aún a pesar de que el resultado sea difícil de interpretar.

Una de las premisas que inicialmente se estableció en el desarrollo del sistema de aprendizaje SLAVE, consistía en que el conocimiento extraído por él, debía ser fácilmente interpretable desde un punto de vista humano, y por esta razón, como hemos visto en capítulos anteriores, se han introducido en su diseño factores que fuerzan al algoritmo a obtener este tipo de conocimiento. En este sentido, SLAVE es un MDD.

El hecho de que SLAVE sea un MDD, implica que tiene una fuerte dependencia de las discretizaciones propuestas por el experto para las variables que intervienen en el problema. Con los siguientes ejemplos, ilustramos esta dependencia.

Ejemplo 5.1 *Supongamos que se desea aprender un sistema que tiene una única variable antecedente X que sabemos que se mueve en el rango [0,3] y una variable consecuente Y que puede tomar dos valores {ClaseA,ClaseB}. Además, conocemos que el sistema tiene el siguiente comportamiento:*

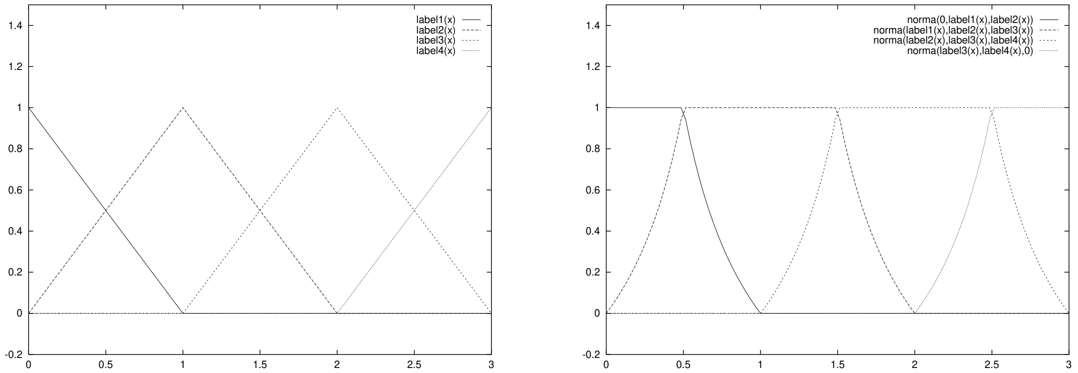
Problema 1: *SI $X \in$ [0.5,1.5] ENTONCES Y es Clase-A*
EN OTRO CASO Y es Clase-B

Para el aprendizaje de dicho sistema, el experto propone la discretización sobre el referencial de la variable X utilizando cuatro etiquetas difusas triangulares, como muestra la figura 5.1(a).

La figura 5.1(b) muestra la alteración de las etiquetas del dominio al aplicar la función de adaptación U de SLAVE

Como solución, SLAVE devuelve las dos reglas que muestran las figuras 5.2(a) y 5.2(b) respectivamente.

En la figura 5.2(c) podemos observar como las dos reglas discriminan sobre un conjunto de ejemplos generados a partir de la discretización propuesta para la variable. Por tanto,



(a) Dominio inicial de la variable X (b) Dominio modificado por la función U

Figura 5.1: Dominio de la variable X

SLAVE consigue una representación correcta del comportamiento del sistema, en base al dominio definido por el experto.

Sin embargo, una inadecuada discretización de las variables continuas conlleva en general un incremento del error en la capacidad de predicción del algoritmo de aprendizaje. Para ilustrar este problema proponemos el siguiente ejemplo.

$\boxed{\textbf{Ejemplo}}$ **5.2** *Supongamos que se desea aprender el comportamiento de los siguientes dos problemas:*

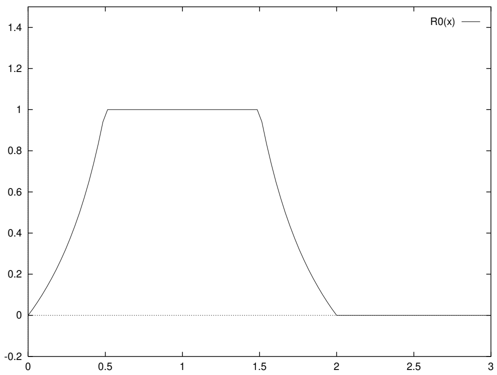
Problema 2: *SI* $X \in$ *[0.7,1.3] ENTONCES* Y *es Clase-A*
EN OTRO CASO Y *es Clase-B*

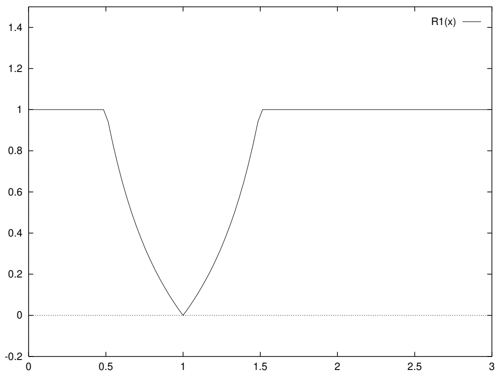
Problema 3: *SI* $X \in$ *[0.9,1.8] ENTONCES* Y *es Clase-A*
EN OTRO CASO Y *es Clase-B*

donde ambos tienen una variable predictiva $X \in$ *[0,3] y una variable de clasificación* Y*. El dominio definido sobre la variable* X *para los dos problemas es igual que el propuesto para el ejemplo 5.1 y se muestra en la figura 5.1 y los valores que puede tomar la variable consecuente* Y*, son al igual que en el ejemplo anterior dos* $\{ClaseA, ClaseB\}$*.*

Aplicando SLAVE a ambos problemas, obtenemos el mismo conjunto de reglas, que además coinciden con el conjunto de reglas que se obtuvieron para el problema del ejemplo anterior.

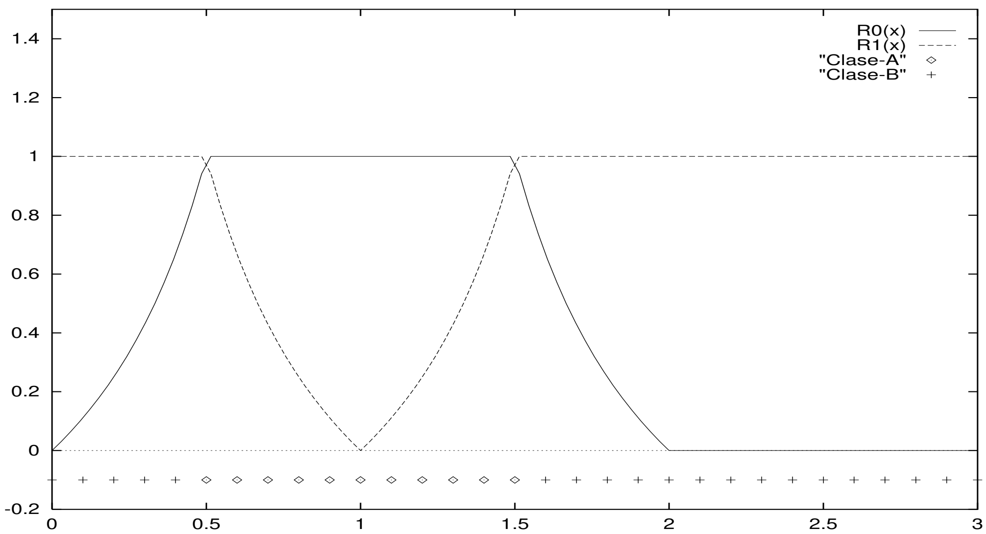
Como podemos observar en la figura 5.3(a) y en la figura 5.3(b), las reglas obtenidas no permiten realizar una correcta clasificación de los ejemplos que se le proponen. En concreto, en ambos casos, el porcentaje de acierto es del 77.42%, mientras en el problema 1, el porcentaje de acierto era del 100%.





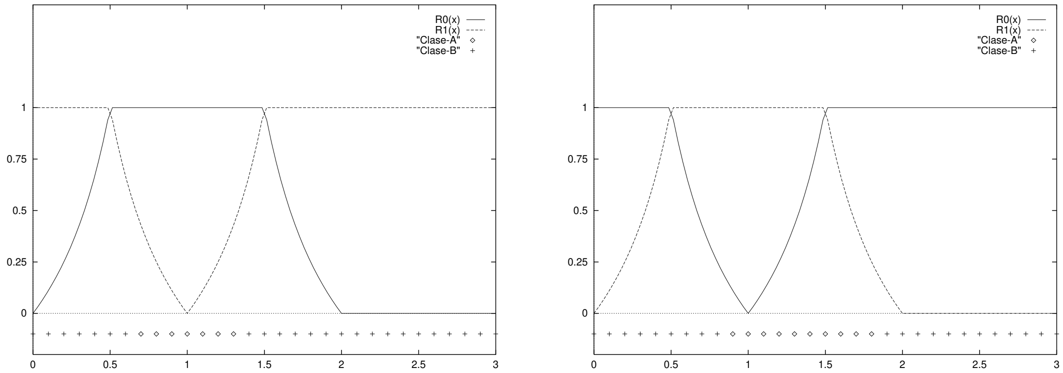
(a) SI X es label2
ENTONCES Y es Clase-A

(b) SI X es {label1, label3, label4 }
ENTONCES Y es Clase-B



(c) Interacción entre las reglas para resolver el problema de clasificación

Figura 5.2: Reglas obtenidas por SLAVE



(a) Problema 2 (b) Problema 3

Figura 5.3: Reglas obtenidas por SLAVE para los dos problemas

Los ejemplos anteriores ponen de manifiesto que existe una fuerte dependencia entre la discretización que se realiza sobre los dominios de las variables involucradas y la capacidad de clasificación del algoritmo empleado. Por tanto, si se desea incrementar la capacidad de aprendizaje de los algoritmos que se basan en el MDD, normalmente se utiliza un algoritmo de ajuste final del conocimiento obtenido. Por consiguiente, un algoritmo que sigue esta línea de actuación, parte de una discretización inicial del rango de las variables continuas para obtener un conocimiento inicial. En un proceso posterior, se modifica la discretización inicial para refinar este conocimiento utilizando un algoritmo de ajuste. Estos algoritmos de ajuste (*tunning*) se han usado ampliamente en el campo de las reglas difusas aplicadas a la modelización cualitativa de sistemas. Algunos de estos métodos de ajuste que podemos encontrar en la bibliografía son [69].

En principio, podría parecer que añadir un algoritmo de ajuste a aquellos algoritmos que no alteran la discretización de los dominios es una buena solución para resolver este problema. De hecho, algunos algoritmos basados el modelo descriptivo o aproximativo de Mamdani o que trabajan con el modelo de regla TSK, tales como [22, 23], lo añaden como una componente más del proceso de aprendizaje. Sin embargo, la mayoría de estos algoritmos de ajuste presentan algunos inconvenientes, tales como:

1.) Estos algoritmos se desarrollan para refinar el conocimiento extraido, que normalmente esta representado en forma de reglas, en los problemas de la modelización cualitativa de sistemas. Normalmente, el modelo de regla empleado en este campo implica que todas las variables de entrada toman uno y sólo un valor de su dominio (modelo de regla de Mamdani [34]). Por tanto, para algoritmos que utilicen otros modelos de regla, el proceso de ajuste requiere una adaptación, que no es fácil en todos los casos.

2.) Los algoritmos de ajuste tienen como objetivo optimizar el conocimiento para incrementar la capacidad de representación del comportamiento del sistema que desean aprender. Por consiguiente, es posible que las discretizaciones resultantes no tengan nada que ver (en cuanto a similaridad) con la discretización inicialmente propuesta. Esto será un problema, cuando uno de los objetivos del algoritmo de aprendizaje sea extraer conocimiento que sea fácilmente interpretable desde el punto de vista humano.

3.) Una mala discretización no sólo implica una limitación en la capacidad de aprendizaje del algoritmo, sino que además puede incluir conocimiento para tratar de compensar la falta de discriminación de las discretizaciones propuestas. Si trabajamos con un algoritmo de aprendizaje que representa su conocimiento con reglas, este problema conlleva que:

   i) variables que son irrelevantes sean consideradas como relevantes en la descripción de las reglas, y que

   ii) se incluya un conjunto de reglas donde sólo sería necesaria una única regla, si la discretización fuera correcta.

   Normalmente, los algoritmos de ajuste asumen que la estructura del conocimiento es válida, y tratan de buscar la mejor discretización, que, manteniendo la estructura de este conocimiento, mejore su comportamiento y se adapte todo lo posible al sistema que se desea aprender. Con esta forma de actuación, el conocimiento resultante está expuesto a la redundancia y/o a la limitación en la capacidad de predicción y/o al incremento del espacio de búsqueda.

4.) En general, son algoritmos con alto costo computacional.

Por las razones expuestas anteriormente, descartamos la definición de un algoritmo de ajuste, que asociado a SLAVE, permita mejorar su capacidad de predicción.

Otra posibilidad para aumentar la capacidad de representación del conocimiento consiste en la definición de un modelo difuso que combine las ventajas tanto de los MDDs como de los MDAs. En este sentido, un modelo difuso hibrido (MDH) parte de una discretización inicial de las variables propuestas por un experto, al igual que los MDDs, pero que durante el proceso de aprendizaje sea capaz de modificar la semántica de las etiquetas propuestas inicialmente. En cuanto a esta segunda capacidad, el sistema tiene un comportamiento parecido a los MDAs. Sin embargo, a diferencia de éste, los valores que finalmente toman las variables están conectados con los que el experto propuso y maneja habitualmente para reconocer el comportamiento del sistema, Es decir, la idea es transformar las etiquetas tan sólo cuando sea preciso, lo mínimo necesario y asociar los cambios a las etiquetas iniciales.

Por consiguiente, con un MDH conseguimos que el conocimiento obtenido sea fácilmente interpretable, ya que se define en base a pequeñas alteraciones de las etiquetas lingüísticas que propone inicialmente el experto, y permite aumentar la capacidad de representación del comportamiento del sistema gracias a la flexibilidad que se le otorga al algoritmo de poder modificar la semántica de las etiquetas.

En este capítulo, nos planteamos la posibilidad de convertir SLAVE en un MDH. Para lo cuál, estudiamos enriquecer el lenguaje para aumentar la capacidad de representación del conocimiento, conservando las propiedades expuestas en los capítulos anteriores.

El enriquecimiento del lenguaje pasa por la inclusión en el modelo de regla de **modificadores semánticos** [172]. Los modificadores semánticos permiten pequeñas alteraciones en la semántica de las etiquetas definidas inicialmente, es decir, permiten la definición de nuevas etiquetas durante el proceso de aprendizaje, matizando el significado de las etiquetas originales.

En la siguiente sección proponemos un nuevo modelo de regla, con la inclusión de modificadores semánticos.

5.2 Modificadores semánticos

Cuando un experto en un campo determinado realiza la discretización de una variable que se mueve sobre un referencial continuo, puede conocer de forma aproximada el significado que para él tiene una determinada etiqueta lingüística, pero difícilmente la conocerá de forma exacta. Por ejemplo, en la figura 5.4(a) aparece una posible representación de la etiqueta *aproximadamente 3*. Esta etiqueta puede responder al significado que dicho valor lingüístico tiene para un experto y para una variable concreta. Sin embargo, si a dicho experto le proponemos para el mismo concepto lingüístico alguna de las representaciones (b), (c) o (d) que aparecen en las figuras 5.4, seguramente, algunas de ellas podrían parecerle tan válidas como la que él propone. Aún más, es posible que para otro experto la semántica de dicho concepto quede reflejada mejor con alguna de las alternativas sugeridas al primer experto.

En la literatura sobre la modelización de etiquetas lingüísticas podemos encontrar resultados y discusiones sobre la forma más apropiada de representar un concepto lingüístico. Este hecho ponen de manifiesto que diferentes personas, con un mismo nivel de conocimiento sobre el tema, tiene una percepción diferente para una misma etiqueta lingüística.

Sin embargo, si se toman dos expertos que han realizado diferentes representaciones de un determinado concepto, y se establece una comunicación entre ellos, donde ese concepto es mencionado directamente, ambos colegas son capaces de comprenderse. Esto indica que la representación mental sobre un concepto lingüístico no es tan importante como la noción aproximada de dicho concepto. En este sentido, todas las representaciones

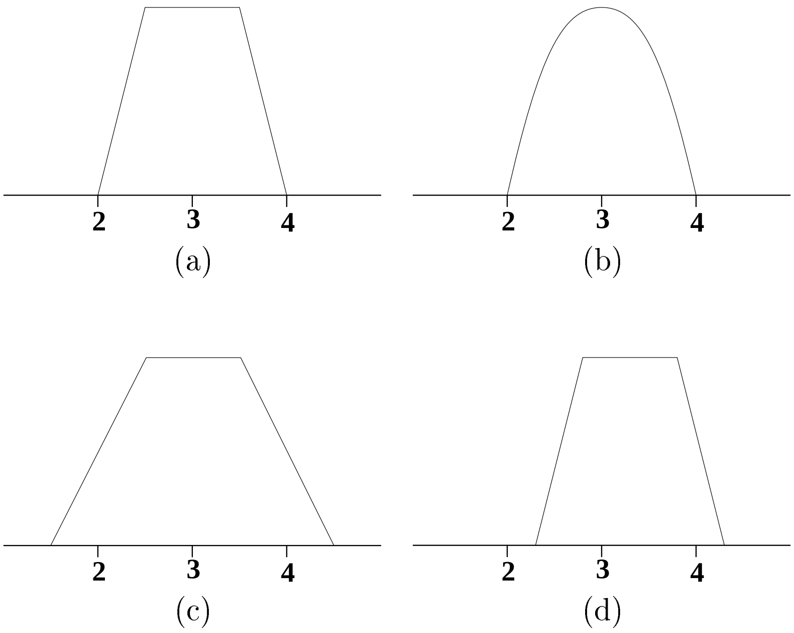


Figura 5.4: Etiqueta lingüística *aproximadamente 3*

que aparecen en la figura 5.4 y muchas más serían igualmente válidas para la etiqueta lingüística *aproximadamente 3* en un proceso de comunicación entre expertos.

Por el contrario, en procesos de extracción automática de conocimiento, la representación semántica de un concepto lingüístico tiene un papel fundamental, ya que para ese concepto se establece una valoración numérica sobre su idoneidad. En este caso, ya no todas las representaciones posibles de un concepto lingüístico, aunque todas representen una noción similar del concepto, son igualmente apropiadas para formar parte del conocimiento que se desea extraer. En este sentido, los MDD encuentran una fuerte restricción para mejorar su capacidad de representar el comportamiento de un determinado sistema.

Por esta razón, parece conveniente dotar a los algoritmos de aprendizaje basados en modelos difusos de la capacidad de alterar la semántica de las etiquetas que componen los dominios de las variables, con el fin de determinar el significado más correcto de la descripción del sistema.

En el campo de la lingüística, los encargados de matizar el significado de un determinado concepto son los modificadores semánticos. Los modificadores semánticos son elementos que aparecen habitualmente en los procesos de comunicación estructurada de alto nivel que permiten ampliar o restringir el significado de una etiqueta lingüística. Estos elementos juegan un papel importante en la comunicación humana, y ya Zadeh en [172] destaca su importancia y su utilidad en la representación del conocimiento en los procesos de razonamiento aproximado.

En modelos de representación del conocimiento basados en reglas difusas, los modificadores semánticos, en muchos casos, aparecen como funciones que permiten alterar la definición de las funciones de pertenencia de las etiquetas difusas. Ya que las etiquetas difusas establecen una distribución de posibilidad de pertenencia al mismo sobre el referencial, los modificadores alteran dicha distribución.

Por tanto, consideraremos un modificador semántico como una función β definida del conjunto de etiquetas al intervalo [0,1], es decir,

$$\begin{array}{cccc} \beta: & D & \longrightarrow & [0,1] \\ & \mathrm{L(x)} & \longrightarrow & \beta(L(x)) \end{array}$$

En consecuencia, un MDH parte de un conjunto inicial de etiquetas lingüísticas T también denominados *términos atómicos* que establecen una primera discretización del referencial de las variables. Además, se tiene un conjunto de modificadores semánticos $S = \{\beta_1, \beta_2, \ldots, \beta_m\}$ que permiten alterar la distribución de posibilidad asociada a los términos atómicos, de manera que el modelo de regla asociado es

$$\text{IF } X \text{ IS } \phi(A) \text{ THEN } Y \text{ IS } B$$

donde $\phi(A)$ representa la composición de funciones $\beta_1 \circ \beta_2 \circ \ldots \circ \beta_t$ que son aplicables a las etiquetas de la variable X.

Ejemplo **5.3** *Supongamos que EDAD es una variable lingüística que tiene asociada los siguientes conjuntos de términos atómicos y de modificadores semánticos:*

$$T = \{\mathit{JOVEN,\ ADULTO,\ VIEJO}\}$$

$$S = \{\mathit{MUY,\ MAS\ O\ MENOS,\ ALGO\ MAS\ QUE,\ ALGO\ MENOS\ QUE}\}$$

En base a los conjunto anteriores, es posible construir nuevas etiquetas lingüísticas como:

MUY JOVEN	*MAS O MENOS JOVEN*
ALGO MAS QUE ADULTO	*ALGO MENOS QUE VIEJO*
ALGO MAS QUE MUY VIEJO	*MUY MUY VIEJO*

Del ejemplo anterior, se deduce que la inclusión de los modificadores semánticos amplian el conjunto de valores que puede tomar una variable lingüística y por consiguiente, incrementa la potencia de descripción de los MDH sobre los MDD.

Sin embargo, es posible que algunas combinaciones entre modificadores semánticos y términos atómicos no sean aplicables o interesantes. Por esta razón, asociado a ellos se han de incluir *reglas sintácticas* que establezcan qué modificadores y en que orden pueden ser aplicados a los términos atómicos y *reglas semánticas* que determinen el significado de las diferentes combinaciones.

En función de la complejidad que permitan las reglas sintácticas para la generación de nuevas etiquetas, podemos clasificar los MDH en las siguientes categorías:

Simple: Sólo permite la aplicación de un único modificador semántico para cada término atómico.

MUY JOVEN
ALGO MAS QUE ADULTO
ALGO MENOS QUE VIEJO
MAS O MENOS JOVEN

Simple redundante: Sólo permite la aplicación de un único modificador semántico para cada término atómico, pero este puede ser aplicado un número finito de veces.

MUY MUY JOVEN
ALGO MAS QUE ALGO MAS QUE ADULTO
MUY MUY MUY VIEJO

Complejo con orden fijo: Permite la aplicación de todos los modificadores semánticos sobre cada etiqueta, pero como máximo, éstos sólo pueden aparecer una única vez. Además, el orden en que los modificadores se componen para obtener la nueva etiqueta es fijo.

ALGO MAS QUE MUY JOVEN
ALGO MENOS QUE MAS O MENOS ADULTO
MAS O MENOS MUY VIEJO

Complejo redundante con orden fijo: Permite la aplicación de todos los modificadores semánticos sobre cada etiqueta, y cada uno de ellos puede aparecer un número finito de veces, aunque el orden de composición de los operadores es fijo.

ALGO MAS QUE MUY MUY JOVEN
ALGO MENOS QUE MAS O MENOS MAS O MENOS ADULTO
MAS O MENOS MAS O MENOS MUY VIEJO

Complejo redundante con orden variable: Permite la aplicación de todos los modificadores semánticos sobre cada etiqueta, y cada uno de ellos puede aparecer un número finito de veces, y no existe un orden predefinido para la composición de los modificadores.

MUY MAS O MENOS MUY JOVEN

MAS O MENOS MUY MUY JOVEN

MAS O MENOS ALGO MAS QUE MUY MAS O MENOS MUY VIEJO

Por supuesto, en la clasificación anterior, no se considera la aparición de reglas sintácticas que determinen que ciertos modificadores semánticos no pueden componerse, y por consiguiente, no son aplicables simultáneamente sobre una etiqueta. Por ejemplo, en el caso anterior, los modificadores ALGO MAS QUE y ALGO MENOS QUE, tiene significados opuestos. Tampoco se considera la existencia de reglas que determinen que ciertos modificadores semánticos no son aplicables a ciertas etiquetas.

La clasificación propuesta anteriormente, establece diferentes categorías en función del grado de comprensibilidad de las descripciones resultantes por la incorporación de modificadores semánticos. Por consiguiente, podemos entender que esta clasificación establece diferentes restricciones en la capacidad de construcción de descripciones que permiten la variables lingüísticas booleanas (VLBs) [172]. Por tanto, el concepto de VLB establece el marco más general, y por consiguiente, con mayor grado de libertad para la combinación de modificadores semánticos en las descripciones de las reglas. Debemos destacar que el modelo de VLB no puede incluirse directamente en SLAVE, debido que el modelo de regla definido en este algoritmo no permite algunas de las combinaciones que incluye el modelo VLB.

Esta clasificación determina la capacidad de generación de nuevas etiquetas en base a las inicialmente propuestas. Sin embargo, a medida que la combinación de modificadores es más libre, la comprensión del significado de la misma es más compleja desde un punto de vista humano. En este sentido, reglas sintácticas más restrictivas conllevan la generación de nuevas etiquetas más comprensibles, y en muchos sentidos, estas son más convenientes para su utilización en los MDH.

Por otra parte, las reglas semánticas asociadas a la variables lingüísticas establecen el significado de los términos atómicos y de los modificadores semánticos. Como se dijo anteriormente, en este trabajo los modificadores semánticos son funciones que se aplican sobre los términos atómicos que alteran su distribución de posibilidad sobre el referencial de una variable. Podemos clasificar los modificadores en función de la naturaleza de su regla semántica asociada en:

Modelización fija: Son los modificadores que tienen asociada una función determinada que no admite ser alterada. Por ejemplo, si tratamos de modelizar el modificador MUY, podemos asociarle una regla semántica del tipo:

$$MUY\,L(x) = L(x)^2$$

de manera, que la representación semántica asociada a MUY VIEJO es

$$(\mu_{VIEJO}(x))^2$$

Modelización variable: Son los modificadores que tienen asociada una función determinada que admite ser alterada mediante la inclusión de argumentos. Así, podemos modelizar el operador MUY como:

$$MUY^a L(x) = L(x)^a$$

donde a es un valor fijo definido entre $]0,\infty]$.

Con estos modificadores, la representación semántica de $MUY^{1.5}$ VIEJO es

$$(\mu_{VIEJO}(x))^{1.5}$$

Los modificadores semánticos variables aportan dos ventajas fundamentales: por una parte, permiten trabajar con diferentes significados de un modificador semántico, ya que alteran la semántica del mismo. Por otro lado, pueden permitir eliminar la redundancia de aplicación de los modificadores sobre los términos atómicos, ya que por ejemplo MUY MUY ALTO con la semántica fija descrita anteriormente es $(\mu_{ALTO}(x))^4$ que coincide con MUY^4 ALTO.

Sin embargo, estos modificadores presentan un inconveniente importante, restan comprensibilidad al significado de la etiqueta resultante. Este inconveniente puede paliarse en cierta medida, si la ponderación asociada a la alteración del modificador semántico no cambia bruscamente el significado del mismo. Como se comentó al principio de esta sección, para que el concepto lingüístico que se desea comunicar sea comprendido por el otro interlocutor no es tan importante la representación mental del concepto, como que la noción del mismo sea semejante para ambos interlocutores.

Por consiguiente, en este sentido, los modificadores semánticos con modelización variables incrementan la capacidad de representación desde el punto de vista de la extracción del conocimiento, y no alteran de forma brusca el significado original del modificador (si son definidos correctamente) desde el punto de vista de su comprensibilidad. Además, estos modificadores, proporcionan un número potencialmente infinito de cambios en las etiquetas originales.

5.3 Redefinir SLAVE como un modelo difuso híbrido

En este capítulo proponemos la ampliación de la capacidad de representación del comportamiento de un sistema, redefiniendo SLAVE como un MDH. En concreto, deseamos que SLAVE sea un MDH complejo con orden fijo, ya que proporciona un modelo de regla simple que permite al algoritmo de aprendizaje obtener conocimiento más cercano al comportamiento real del sistema. Además, dentro de este modelo de regla, incluímos los operadores semánticos variables. Los modificadores semánticos incluidos en SLAVE se estudian en el siguiente apartado.

5.3.1 Los modificadores semánticos de SLAVE

SLAVE es un sistema de aprendizaje de propósito general. En este sentido, se hace recomendable establecer un conjunto de modificadores semánticos que no siendo muy numeroso, para no incrementar excesivamente el espacio de búsqueda, permita un ajuste correcto de las etiquetas iniciales. Por consiguiente, para la definición de los mismos, atenderemos a las modificaciones básicas que permitirán mejorar la capacidad de aprendizaje del sistema. Así, si suponemos que la etiqueta inicialmente propuesta por un experto para el concepto *aproximadamente 3* es la representada en la figura 5.4(a), las etiquetas que aparecen en las figuras 5.4(b), 5.4(c) y 5.4(d) pueden obtenerse como alteraciones elementales provocadas por modificadores semánticos. Estas tres alteraciones elementales son el **cambio de la forma de la etiqueta**, el **cambio de amplitud de la etiqueta** y el **desplazamiento de la etiqueta** respectivamente.

Existen muchas más alteraciones elementales que las que se proponen en las figuras anteriores, pero en nuestro estudio solo consideraremos las tres anteriormente mencionadas.

5.3.1.1 Modificación de la forma de una etiqueta

Estos operadores permiten modificar la distribución de posibilidad que refleja la función de pertenencia de los elementos del referencial sobre la etiqueta. De una forma más general, podemos realizar la siguiente caracterización de estos operadores:

Definición 5.1 *Dadas dos etiquetas L_1 y L_2, diremos que L_2 es una etiqueta resultante de una alteración de cambio de forma de la etiqueta L_1, que notaremos por*

$$L_2 = ACD(L_1)$$

si el soporte de L_2 coincide con el soporte de L_1, es decir,

$$Sop(L_1) = Sop(L_2).$$

De entre el conjunto de familias de funciones que satisfacen la anterior definición, estamos interesados en aquellas funciones que alteren la distribución de la etiqueta manteniendo la ordenación relativa en la valoración de posibilidad de los elementos pertenecientes al soporte de la misma,. Por ejemplo, si en la etiqueta original L, dados cualesquiera dos elementos del soporte de la misma a y b, se verifica que

$$\mu_L(a) \geq \mu_L(b)$$

entonces un operador β aplicado a dicha etiqueta debe satisfacer que

$$\beta(\mu_L(a)) \geq \beta(\mu_L(b))$$

Estos operadores de alteración de la distribución de posibilidad de la etiqueta son interesantes desde nuestro punto de vista, porque preservan la ordenación de los elementos del soporte de la etiqueta original con respecto a su posibilidad relativa de pertenencia a la misma, y en consecuencia, la etiqueta resultante de la aplicación del operador presenta una semántica ligeramente diferente a la propuesta inicialmente por el experto.

El operador $\beta(L(x), a) = \mu_L(x)^a$ es un operador que representa a una familia de funciones que verifica la definición del operador ACD y además preserva la propiedad anterior. Un caso particular de este operador es aquel que transforma la etiqueta original mediante $\mu_L(x)^2$ que fué inicialmente propuesto en [172] para representar al modificador semántico **MUY**.

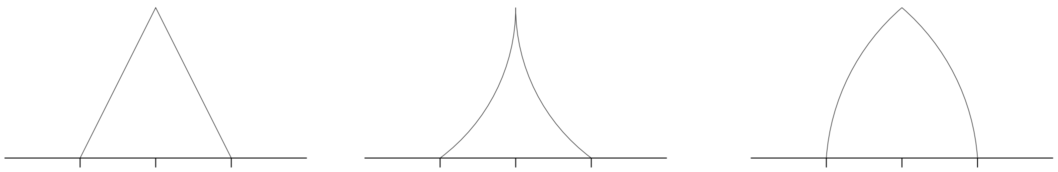
Como se muestra en la figura 5.5, este operador tiene dos interpretaciones diferentes en función del valor del parámetro a. Para valores de $a \in]1, \infty[$, se satisface que $\mu_L(x) \geq \mu_L(x)^a$ para todos los elementos que pertenecen al soporte de la etiqueta. En consecuencia, tras aplicar el operador, aquellos elementos con valores de posibilidad cercanos a uno en la etiqueta original mantienen o decrementan poco su posibilidad de pertenencia a la etiqueta. Por el contrario, los valores de posibilidad cercanos a cero, sufren un decremento más pronunciado en la etiqueta resultante. Por consiguiente, este operador puede entenderse como una modificación en la imprecisión de la etiqueta, ya que los primeros (los que poseen valores cercanos a uno) tras la aplicación del operador, aumentan la posibilidad de pertenecer a la nueva etiqueta en relación a los que poseen valores cercanos a cero en la etiqueta original.

Sin embargo, para valores de $a \in]0, 1[$, se produce el efecto contrario, es decir, la etiqueta resultante aumenta su grado de imprecisión, ya que todos los valores del soporte de la etiqueta mantienen o aumentan su posibilidad de pertenencia a la etiqueta, y este cambio es más evidente sobre aquellos elementos que poseían valores de posibilidad cercanos a cero en la etiqueta original.

Dada un etiqueta cualesquiera L, notaremos este operador de modificación semántica como L^a, donde $a \in]0, \infty[$.

5.3.1.2 Modificación de la anchura de una etiqueta

Este segundo operador elemental de alteración de la etiqueta se puede formular formalmente de la siguiente manera:



Etiqueta Original — Transformación con $a = 2$ — Transformación con $a = \frac{1}{2}$

Figura 5.5: Operador $\mu_L(x)^a$

Definición 5.2 *Dadas dos etiquetas L_1 y L_2, diremos que L_2 es una etiqueta resultante de una alteración de cambio en anchura de la etiqueta L_1, que notaremos por $L_2 = ACA(L_1)$, si se verifica que*

$$Sop(L_1) \subseteq Sop(L_2) \qquad o \qquad Sop(L_2) \subseteq Sop(L_1)$$

es decir, el soporte de la etiqueta L_1 está contenido en el soporte de la etiqueta L_2, o es el soporte de la etiqueta L_2 el que está contenido en la etiqueta L_1.

En base a esta definición, la etiqueta resultante al aplicar un operador ACA provoca modificaciones en la distribución de posibilidad de la etiqueta original al igual que en los operadores ACD, pero a diferencia de éstos, el cambio en la distribución de posibilidad viene determinado por la ampliación o reducción del soporte de la etiqueta. Por consiguiente, al igual que el operador anterior, éste también se puede considerar como una modificación que permite aumentar o disminuir la imprecisión de la etiqueta, mediante la inclusión o exclusión de nuevos elementos del referencial en su soporte.

Antes de fijar el operador ACA que utilizará SLAVE, es necesario establecer la siguientes definiciones.

Definición 5.3 *Dada una etiqueta L perteneciente al dominio D de una determinada variable X, que toma valores en un referencial U, cuyos elementos están bien ordenados, definimos*

i) el extremo inferior de la etiqueta, que notaremos por $L^{\triangleleft}$,

$$L^{\triangleleft} = \inf Sop(L)$$

ii) el extremo superior de la etiqueta, que notaremos por $L^{\triangleright}$,

$$L^{\triangleright} = \sup Sop(L)$$

iii) el centro de la etiqueta, que notaremos por $L^{\triangledown}$, como el elemento que se encuentra a igual distancia del extremo superior y del extremos inferior, es decir,

$$L^{\triangledown} = \frac{L^{\triangleright} + L^{\triangleleft}}{2}$$

En base a estos tres puntos que se extraen de una etiqueta, definimos la siguiente familia de funciones que permiten modificar la amplitud de una etiqueta:

Definición 5.4 *Dada una etiqueta L, perteneciente al dominio D de una determinada variable X, que toma valores en un referencial U, cuyo elementos están bien ordenados, y dados los puntos $L^{\triangleleft}$, $L^{\triangleright}$ y L^{∇}, el operador ϕ_1^b modifica la etiqueta original de la siguiente forma:*

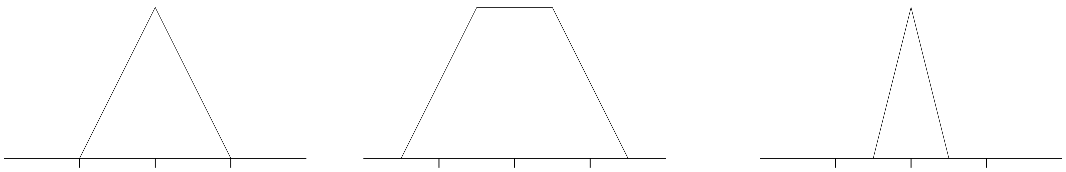
$$\phi_1^b(\mu_L(x)) = \begin{cases} \mu_L(x+b) & si\ x \in [L^{\triangleleft} - b, \min(L^{\nabla}, L^{\nabla} - b)] \\ \mu_L(x-b) & si\ x \in [\max(L^{\nabla}, L^{\nabla} - b), L^{\triangleright} + b] \\ \mu_L(L^{\nabla}) & si\ x \in]\min(L^{\nabla}, L^{\nabla} - b), \max(L^{\nabla}, L^{\nabla} + b)[\\ 0 & en\ otro\ caso \end{cases}$$

En la definición anterior, los valores positivos para el parámetro b provocan una ampliación del soporte de la etiqueta, mientras que los valores negativos para dicho parámetro disminuyen el soporte de la etiqueta original. En esta definición podemos encontrar dos inconvenientes:

a) Mientras que las etiquetas resultantes de la aplicación de este operador, que implican una ampliación del soporte de la etiqueta original, son una distribución de posibilidad, las etiquetas obtenidas tras la aplicación de este operador para reducir el soporte de la etiqueta, no satisfacen en general esta propiedad aún siendo la original una distribución de posibilidad. Por esta razón, modificamos la definición del operador anterior. Por consiguiente, redefinimos el operador de alteración del soporte de la etiqueta, introduciendo una operación de normalización de la siguiente forma:
$$\beta_2^b(\mu_L(x)) = \frac{\phi_1^b(\mu_L(x))}{\max_{u \in U} \phi_1^b(\mu_L(u))}$$

b) Es necesario acotar el máximo grado de modificación si deseamos que las etiquetas resultantes mantengan un significado semejante al de la etiqueta original. En cuanto a la reducción del soporte de la etiqueta, ésta se encuentra acotada, ya que el parámetro b no puede tomar un valor inferior a $L^{\triangleleft} - L^{\nabla}$. Sin embargo, el proceso de ampliación del soporte, solamente está acotado por el tamaño del referencial. En nuestro modelo deseamos que la etiqueta original pueda reducir su soporte a un único valor, dando la posibilidad al sistema de reducir la imprecisión, pero como máximo que el operador amplie el soporte al doble del de la etiqueta original. Por tanto, establecemos la siguiente redefinición del operador β_2^b como:

$$\beta_2^b(\mu_L(x)) = \frac{\phi^b(\mu_L(x))}{\max_{u \in U} \phi^b(\mu_L(u))}$$



Etiqueta Original Transformación con $b = \frac{1}{2}$ Transformación con $b = -\frac{1}{2}$

Figura 5.6: Operador L_b

donde ahora

$$\phi^b(\mu_L(x)) = \begin{cases} \mu_L(x + (b \times \delta)) & \text{si } x \in [L^{\triangleleft} - (b \times \delta), \min(L^{\triangledown}, L^{\triangledown} - (b \times \delta))] \\ \mu_L(x - (b \times \delta)) & \text{si } x \in [\max(L^{\triangledown}, L^{\triangledown} - (b \times \delta)), L^{\triangleright} + (b \times \delta)] \\ \mu_L(L^{\triangledown}) & \text{si } x \in]\min(L^{\triangledown}, L^{\triangledown} - (b \times \delta)), \max(L^{\triangledown}, L^{\triangledown} + (b \times \delta))[\\ 0 & \text{en otro caso} \end{cases}$$

siendo $b \in [-1, 1]$ y $\delta = L^{\triangledown} - L^{\triangleleft}$.

Con esta nueva definición, los valores del parámetro $b \in [-1, 0[$ implican una reducción, mientras que para $b \in]0, 1]$ éstos implican una ampliación del soporte de la etiqueta original.

Con las dos matizaciones anteriores, conseguimos restringir las modificaciones permitidas al operador, así como, que cualquier modificación sobre la etiqueta original satisfaga ser una distribución de posibilidad sobre el referencial. En la figura 5.6 mostramos el comportamiento de este operador sobre una etiqueta original de forma triangular.

5.3.1.3 Modificación por desplazamiento de una etiqueta

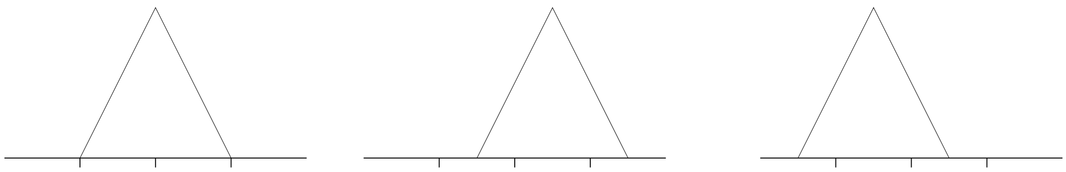
El tercer tipo de modificación básica que deseamos incorporar a SLAVE es la capacidad de desplazar ligeramente la etiqueta de su ubicación original sobre el referencial. Formalmente, este tipo de operación se puede definir en los siguientes términos:

Definición 5.5 *Dadas dos etiquetas L_1 y L_2, diremos que L_2 es una etiqueta resultante de una alteración de desplazamiento de la etiqueta L_1, que notaremos por $L_2 = AD(L_1)$, si se verifica que*

$$\mu_{L_1}(x) = \mu_{L_2}(x + c) \qquad \forall x \in U$$

donde μ_L representa la función de pertenencia de un elemento x del referencial U a la etiqueta L, y c es un valor real fijo.

En esta definición está incluido cualquier desplazamiento de la etiqueta sobre el referencial, pero al igual que se propuso en el operador anterior, estamos interesados en acotar



Etiqueta Original   Transformación con $c = \frac{1}{2}$ Transformación con $c = -\frac{1}{2}$

Figura 5.7: Operador $\beta_3(L, c)$

los máximos desplazamientos permitidos. En este sentido, sólo permitiremos desplazamientos de la etiqueta que mantengan en su soporte al menos la mitad de los elementos que pertenencían al soporte de la etiqueta original. Para respetar este criterio, realizamos la siguiente definición de este operador de desplazamiento $\beta_3(L, c)$

$$\beta_3(L, c) = \mu_L(u + c * \delta) \qquad \forall u \in U$$

donde $c \in [-1, 1]$ y $\delta = L^{\triangledown} - L^{\triangleleft}$.

Por tanto, δ representa el máximo desplazamiento permitido de la etiqueta, que coincide con la mitad del soporte de la etiqueta. Los valores positivos para el parámetro c indican desplazamientos hacia la derecha de la etiqueta, mientras que los valores negativos implican desplazamientos hacia la izquierda.

En la siguiente figura, mostramos un ejemplo del comportamiento de este operador sobre una etiqueta original de forma triangular.

5.3.2 Composición de los modificadores semánticos

La inclusión de uno de los anteriores operadores semánticos aumenta considerablemente la potencia de representación del comportamiento de los sistemas. Por ejemplo, si el algoritmo incluyera el operador $\beta_2(L, b)$, podría encontrar una descripción correcta del problema 2 planteado en el ejemplo 5.2 a partir de las etiquetas inicialmente propuestas. Para el valor $b = -0.6$ aplicado a la etiqueta $label2$ en la regla propuesta para la Clase-A, permite una discriminación correcta de las clases involucradas en el problema como muestra la figura 5.8.

Sin embargo, este operador semántico no sería suficiente para discriminar las clases propuestas en el problema 3 del mismo ejemplo. Es más, para este problema, la aplicación de forma aislada de cada uno de los operadores propuestos, no proporcionaría una discriminación correcta.

Este hecho pone de manifiesto que en problemas complejos es necesaria la aplicación de más de uno de los anteriores operadores semánticos para encontrar un correcta descripción

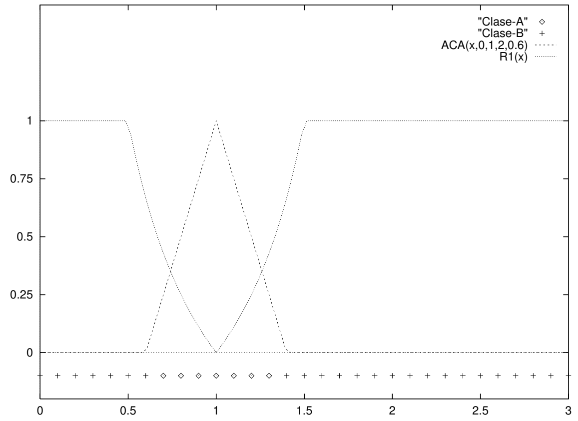


Figura 5.8: Solución al problema 2 usando el operador $\beta_2(label2, -0.6)$

del comportamiento del sistema.

Como ya se ha dicho en este capítulo, cuando se tiene un conjunto de modificadores semánticos aplicables a las etiquetas, se pueden tomar dos opciones: establecer una sintaxis fija en la aplicación de los modificadores sobre las etiquetas, es decir, los modificadores siempre se aplican sobre las etiquetas en un orden previamente fijado, o establecer una sintaxis libre donde esté permitida cualquier combinación en la composición de los modificadores. Antes de tomar partida por una de estas opciones, es necesario estudiar algunas propiedades de los modificadores definidos.

5.3.3 Aproximación genética a la alteración de las etiquetas

La introducción de los operadores semánticos anteriores implica una modificación en la representación del conocimiento extraído, y por consiguiente, una nueva propuesta de modelo de regla. Así, proponemos un modelo de regla que incluye el anterior conjunto de operadores semánticos:

$$\text{IF } X_1 \textit{ is } A_1(a_1, b_1, c_1) \textit{ and } \ldots \textit{ and } X_p \textit{ is } A_p(a_p, b_p, c_p) \text{ THEN } Y \textit{ is } B$$

donde cada variable X_i tiene un conjunto de U_i y toma valores en un dominio finito D_i, para $i \in \{1, \ldots, p\}$. El conjunto de referencia para Y es V y su dominio es F. El valor de la variable Y es B, donde $B \in F$ y el valor de la variable X_i es $A_i(a_i, b_i, c_i)$, donde $A_i \in P(D_i)$ y $P(D_i)$ denota el conjunto de las partes D_i, y a_i, b_i y c_i son los parámetros de los modificadores semánticos β_1, β_2 y β_3 respectivamente. Así,

$$A_i(a_i, b_i, c_i) \equiv \beta_3(\beta_2(\beta_1(A_i, a_i), b_i), c_i)$$

Readaptar SLAVE como un MDH capaz de extraer el conocimiento usando como herramienta el anterior modelo de regla, implica una redefinición del módulo de extracción

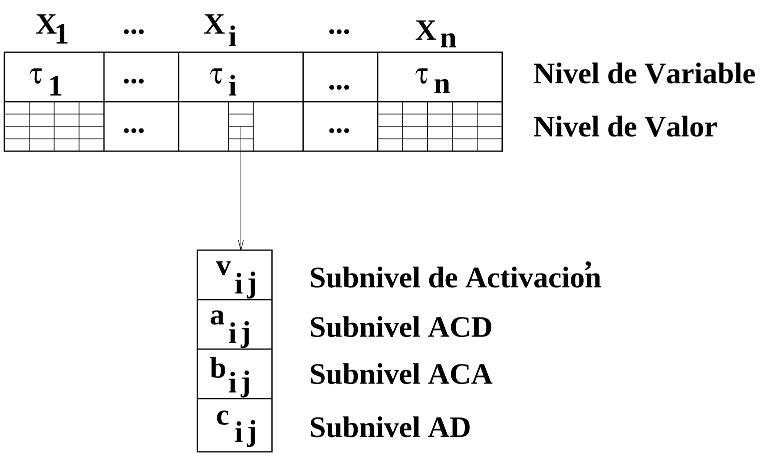


Figura 5.9: Representación del conjunto de soluciones en el AG

de reglas del sistema que viene implementado mediante un algoritmo genético. En esta redefinición, se incorporan como nuevos elementos a buscar, no sólo las variables relevantes para la descripción del concepto y la asignación más apropiada de valores de su dominio, sino que además, forma parte importante del proceso la búsqueda de los valores más apropiados para los parámetros de los modificadores semánticos que se proponen. Por tanto, es necesario incluir en la codificación de los cromosomas la información relativa a los operadores semánticos.

Como se describió en el capitulo anterior, SLAVE usa un algoritmo genético con dos niveles, donde en el nivel de variable se busca el conjunto de variables que son relevantes para describir el concepto, mientras que en el nivel de valor se busca la mejor asignación de valores para las variables relevantes. La mejor solución se busca mediante la coevolución de los dos niveles.

Como los modificadores semánticos afectan directamente al nivel de valor, proponemos la modificación de este nivel, incorporando tres subniveles como muestra la figura 5.9 para buscar la mejor configuración de los modificadores semánticos para las etiquetas asociadas a las variables relevantes.

Con esta representación en dos niveles proponemos nuevamente un modelo basado en la coevolución para encontrar la mejor solución. A diferencia del modelo propuesto en el capítulo anterior, para esta representación aplicamos un modelo basado en la coevolución de cinco algoritmos genéticos, donde la representación de cada nivel o subnivel evoluciona de forma independiente, pero la valoración de una solución se establece en función de la colaboración entre ellos.

Cada subnivel del nivel de valor hace referencia a cada uno de los modificadores semánticos descritos anteriormente. El tamaño de cada subnivel es el mismo y coincide con el número total de etiquetas implicadas en el problema, ya que cada modificador se aplica para todas y cada una de las etiquetas que constituyen los dominios de las variables. Por tanto, todos los subniveles son de longitud fija.

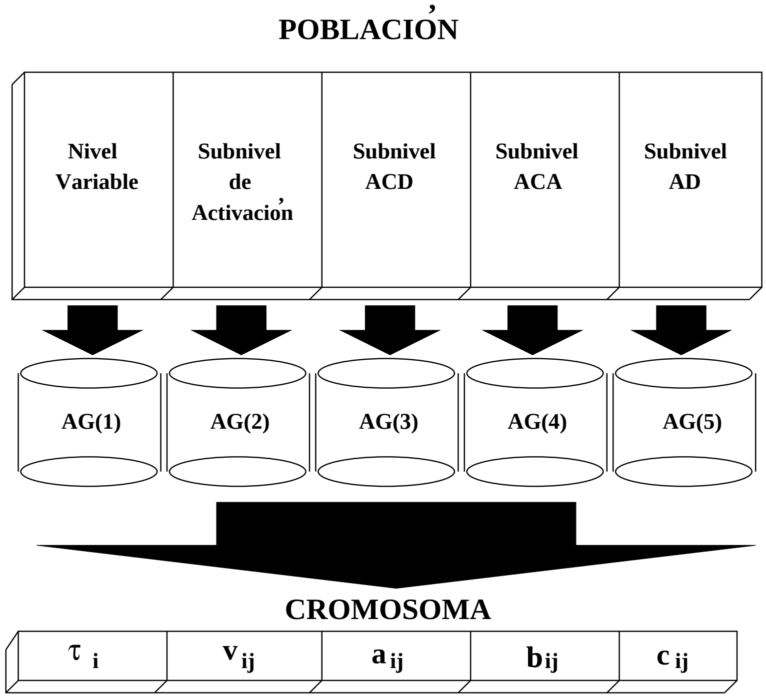


Figura 5.10: Coevolución sobre cinco algoritmos genéticos

El subnivel de activación representa la información relativa a la presencia o ausencia de cada valor en la descripción de la regla. En este sentido, mantiene el significado del anterior nivel de valor, y al igual que esté, conserva la codificación binaria, de manera que el valor uno indica presencia, mientras que el valor cero indica la no consideración de dicha etiqueta.

El subnivel ACD está formado por el trozo de cromosoma que hace referencia a los parámetros del modificador semántico β_1. El rango de valores que puede tomar este parámetro se mueve en el intervalo [0,∞]. Un valor en [0,1[establece un crecimiento de la imprecisión con respecto a la etiqueta original, un valor en]1, ∞], una reducción de la imprecisión, mientras que el valor uno preserva la distribución de la etiqueta original. Es importante destacar que este operador es aplicable únicamenta sobre etiquetas definidas como conjuntos difusos, ya que no altera la semántica de etiquetas definidas como intervalos, es decir, etiquetas donde todos los elementos del soporte de la etiqueta tienen valoración uno.

El subnivel ACA mantiene la información relativa a los parámetros del modificador semántico β_2. Al igual que el modificador β_3, estos parámetros se mueven en el intervalo [-1, 1]. Un valor del parámetro entre [-1,0[o entre]0,1] indican una ampliación o una reducción de la imprecisión con respecto a la etiqueta original respectivamente. No se produce alteración de la etiqueta original si el parámetro toma valor cero.

El subnivel AD codifica los parámetros asociados al modificador β_3 para cada etiqueta. El rango de variación de cada parámetro está en el intervalo [-1,1], donde un valor en

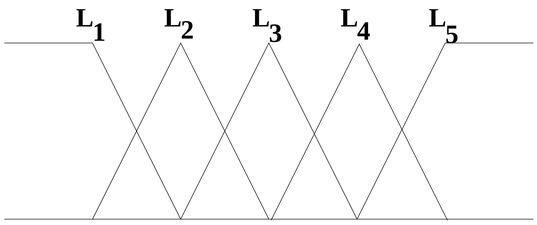


Figura 5.11: Dominio de la variable X

el rango [-1,0[implica un desplazamiento hacia la izquierda con respecto a la etiqueta original, un valor en el rango]0,1] un desplazamiento hacia la derecha, mientras que el cero indica que no se produce ninguna modificación por desplazamiento de la etiqueta original.

Por consiguiente, la semántica de una etiqueta que aparece en la descripción de una regla viene determinada por un tupla de tres componentes (a_i, b_i, c_i), donde a_i, b_i y c_i son los parámetros de los modificadores β_1, β_2 y β_3 respectivamente.

Ejemplo 5.4 *Supongamos que X es una variable considerada relevante para un elemento de la población en la descripción de un concepto. Supongamos además que el dominio asociado a esta variable está compuesto por cinco etiquetas $\{L_1, L_2, L_3, L_4, L_5\}$, siendo la configuración del elemento para esta variable la siguiente:*

	L_1	L_2	L_3	L_4	L_5
v_i	*0*	*1*	*1*	*1*	*0*
a_i	*1*	*2*	*1*	*1*	*0.5*
b_i	*-0.5*	*0*	*0*	*0.25*	*1*
c_i	*0.25*	*0.5*	*0*	*0*	*-0.1*

Los valores del subnivel de activación nos indican que las etiquetas que toma como valor la variable X son L_2, L_3 y L_4. Por consiguiente, las otras dos etiquetas no son consideradas y los valores de los parámetros de los modificadores semánticos asociados a ellas no se consideran tampoco. La etiqueta L_3(1,1,0,0) conserva su definición original, ya que los tres parámetros asociados a los modificadores semánticos toman los valores que indican la no alteración de la etiqueta. La etiqueta L_4(1,1,0.25,0) sólamente sufre una alteración provocada por el modificador β_2 que provoca una ampliación del soporte de la etiqueta original. Por el contrario, la etiqueta L_2(1,2,-0.5,0.25) se ve afectada por los tres modificadores semánticos. Así, la configuración de la variable X para este caso concreto es:

$$X \textit{ IS } L_1\textit{(0,1,-0.5,0.25) or } L_2\textit{(1,2,-0.5,0.25) or } L_3\textit{(1,1,0,0) or } L_4\textit{(1,1,0.25,0) or } L_5\textit{(0,0.5,1,-0.1)}$$

(a) Modificaciones aplicadas a las etiquetas (b) Etiqueta resultante normalizada

Figura 5.12: Representación del valor asignado a la variable X

y se puede expresar como:

$$X \textit{ IS } L_2\textit{(2,-0.5,0.25) or } L_3\textit{(1,0,0) or } L_4\textit{(1,0.25,0)}$$

o de forma más simple como:

$$X \textit{ IS } L_2\textit{(2,-0.5,0.25) or } L_3 \textit{ or } L_4\textit{(-,0.25,-)}$$

De manera que si la figura 5.11 representa el dominio de la variable X, la figura 5.12(a) presenta las modificaciones aplicadas a cada etiqueta activa, mientras que la figura 5.12(b) representa el valor normalizado que toma dicha variable para esta configuración de los parámetros del nivel de valor.

En cuanto a la generación de la población inicial del proceso evolutivo, en el nivel de variable y en el subnivel de activación, este proceso se realiza de forma semejante al descrito en el capítulo anterior, ya que estos dos niveles se heredan directamente de la anterior representación. Con respecto al resto de los subniveles, tomamos el criterio de no alterar inicialmente la definición de las etiquetas de los dominios. Por esta razón, todos los parámetros a_{ij} del modificador semántico β_1 se inicializan a 1, mientras que los parámetros b_{ij} y c_{ij} correspondientes a los modificadores semánticos β_2 y β_3 se inicializan a 0.

Este criterio de inicialización de los parámetros de los modificadores semánticos es justificable, ya que deseamos que las etiquetas utilizadas para representar el conocimiento extraido sean lo más parecidas posibles a las que se proponen inicialmente.

Al igual que sucede con la generación de la población inicial, los operadores genéticos aplicados al nivel de variable y al subnivel de activación coinciden con los ya propuestos en el capítulo. Sin embargo, es necesario establecer los operadores genéticos para los otros tres subniveles, para los cuales únicamente se definen los dos operadores estandar, es decir, el operador de mutación y el operador de cruce.

El criterio utilizado para la construcción de los operadores de mutación se basa en provocar pequeñas modificaciones que consisten en incrementos o decrementos de los parámetros de los operadores semánticos, de manera que el algoritmo genético determine el sentido más apropiado en el que los parámetros deben evolucionar. El uso estricto del criterio anterior provoca que los parámetros de los modificadores semánticos tiendan a tomar valores que modifican las etiquetas originales. Por esta razón, en el operador de mutación aplicado a estos subniveles aumentamos la probabilidad de que cada parámetro tome el valor que no modifica la etiqueta original. El esquema 5.1 representa de forma general el criterio anterior del operador de mutación para el subnivel ACA y AD.

Esquema 5.1 *Esquema general del operador de mutación para el subnivel ACA y AD.*

1. Sea d_{ij} el parámetro de un determinado modificador semántico para la etiqueta j-ésima de la variable i-ésima.
2. Sea p un número generado de forma aleatoria en el rango [0,1]
3. Si $p \in [0, PN[$ entonces $d_{ij} = NE$, donde PN representa un valor entre [0,1[y NE es el valor que permite que el modificador no altere la semántica de la etiqueta inicial.
4. Si $p \in [PN, PN + \frac{PN}{3}[$ entonces $d_{ij} = d_{ij} + \delta$, donde δ representa la variación mínima que se desea provocar sobre dicho modificador.
5. Si $p \in [PN+\frac{PN}{3}, PN+\frac{2\times PN}{3}[$ entonces $d_{ij} = d_{ij}-\delta$
6. Si $p \in [PN + \frac{2\times PN}{3}$,1] entonces d_{ij} no modifica su valor.

El operador de mutación para el subnivel ACD mantiene una estructura semejante, aunque algo más compleja debido al comportamiento que presenta el operador ACD sobre su rango de definición. Así, si el valor del parámetro está en el rango [0, 1[parece lógico que la mínima modificación definida sobre él, sea más pequeña que si el valor del parámetro se encuentra en el rango]1,∞[. Por esta razón, se definen dos valoraciones diferentes de variación mínima sobre el operador, δ_1 asociada a valores en [0,1] y δ_2 para valores en]1,∞[. En este sentido, el esquema anterior es aplicable sin más que modificar el comportamiento de los pasos 4 y 5, de manera que, en función del intervalo en el cuál se encuentre el parámetro a mutar, se sustituirá δ por δ_1 o δ_2.

El operador de cruce definido para estos tres subniveles es el cruce por intercambio sobre dos puntos y coincide con el definido para el subnivel de activación. A pesar de que estos tres subniveles mantienen codificación real, no aplicamos cruces del tipo BLX_α como se hace en el nivel de variable, porque no deseamos que se produzcan fuertes modificaciones

	Codificación	Op. genéticos
Nivel de Variable	Real [0,1]	Cruce BLX_α Mutación no uniforme
Subnivel de Activación	Binaria	Cruce sobre 2 puntos Mutación estandar Operador de rotación Operador de generalización Operador OR Operador AND
Subnivel ACD	Real $[0,\infty[$	Cruce sobre 2 puntos Mutación tipo esquema 5.1
Subnivel ACA	Real [-1,1]	Cruce sobre 2 puntos Mutación tipo esquema 5.1
Subnivel AD	Real [-1,1]	Cruce sobre 2 puntos Mutación tipo esquema 5.1

Tabla 5.1: Características de los algoritmos genéticos implicados en la coevolución

en los párametros de los operadores semánticos, ya que esto aumentaría el tiempo de convergencia del proceso evolutivo, aún a pesar de que posiblemente las descripciones que se obtengan sean mejores.

La tabla 5.1 muestra de forma resumida las características de los algoritmos genéticos implicados en cada nivel del proceso de coevolución.

La característica fundamental del proceso de coevolución que se propone consiste en que no se establece una evaluación individualizada por niveles, sino que existe una única función de evaluación que establece la bondad para cada individuo dependiendo de la constitución global (es decir, de la agrupación de todos los niveles). Debido a esto, el proceso de selección no se realiza de forma individualizada, sino que en función de la valoración de cada individuo se establece un orden lineal sobre todos los elementos de la población. De esta forma, la nueva población se construye seleccionando individuos completos de la última población. Por consiguiente, mientras la aplicación de los operadores genéticos es particular para cada nivel, las operaciones de selección y evaluación son aplicadas a individuos completos.

La función de evaluación usada en SLAVE se definía en base a una EFL sobre varios criterios, que sigue siendo válida para esta nueva redefinición del módulo de selección de reglas, sin más que incorporar en el cálculo, tanto en el grado de satisfacción de la condición de consistencia débil, como en el grado de completitud, las modificaciones provocadas por los modificadores en las etiquetas originales. Además, deseamos incorporar nuevos criterios a la EFL para encontrar la mejor asignación a cada una de las variables relevantes de la descripción de la regla, de manera, que dicha asignación tenga el mayor parecido posible con las etiquetas que se proponían inicialmente.

Se puede establecer un criterio apropiado atendiendo a una medida de distancia, tomando como referencia el valor del operador semántico que no modifica la etiqueta original.

Definición 5.6 *Sean $(d_1, \ldots d_n)$ los parámetros de un determinado operador semántico M que afectan a las etiquetas iniciales que aparecen para cada variable en la descripción de un individuo I, y sea b el valor que hace que dicho modificador no afecte al cambio de semántica de la etiqueta. Definimos la diferencia entre las etiquetas resultantes con respecto a las inicialmente propuestas, que notaremos por $Difer_M(I)$, como la distancia entre los parámetros $(d_1, \ldots d_n)$ con respecto a $(b, \ldots, b)$, es decir,*

$$Difer_M(I) = \sqrt[2]{\sum_{i=1}^{n}(d_i - b)^2}$$

Para aclarar la anterior definición, proponemos el siguiente ejemplo:

Ejemplo 5.5 *Sea I un cromosoma compuesto por dos variables X_1 y X_2 donde cada variable tiene un dominio asociado con tres etiquetas $\{L_{11}, L_{12}, L_{13}\}$ y $\{L_{21}, L_{22}, L_{23}\}$ respectivamente. Supongamos que dicho cromosoma tiene la siguiente configuración:*

Nivel de Variable					
	τ_1	τ_2	τ_{umbral}		
	0.9	*0.1*	*0.6*		
Subnivel de Activación					
L_{11}	L_{12}	L_{13}	L_{21}	L_{22}	L_{23}
0	*1*	*1*	*1*	*0*	*0*
Subnivel ACD					
L_{11}	L_{12}	L_{13}	L_{21}	L_{22}	L_{23}
5.5	*0.5*	*2*	*1*	*0.5*	*3.5*
Subnivel ACA					
L_{11}	L_{12}	L_{13}	L_{21}	L_{22}	L_{23}
-1	*0*	*0.4*	*-0.4*	*1*	*0*
Subnivel AD					
L_{11}	L_{12}	L_{13}	L_{21}	L_{22}	L_{23}
0	*-0.2*	*0.6*	*-1*	*1*	*0.2*

El umbral de activación del nivel de variable τ_{umbral} toma el valor 0.6, por lo que la variable X_2 no aparece en la descripción de la regla y por lo tanto, ninguna de sus etiquetas será computada para determinar la medida de distancia. Por el contrario, la variable X_1 aparece como parte de la descripción, y sus etiquetas activas serán las que utilizaremos para establecer la diferencia. Si nos fijamos en el nivel de activación para *la primera variable, podemos observar que únicamente aparecen en la descripción de la regla las etiquetas L_{12} y L_{13}, y será sobre estas, sobre las que se establezca el cálculo de la distancia.*

Así sobre el subnivel ACA, tenemos que

$$Difer_{ACA}(I) = \sqrt[2]{(0-0)^2 + (0.4-0)^2} = 0.4$$

ya que el cero representa la no modificación de la etiqueta original para este operador. De igual forma podemos calcular:

$$Difer_{AD}(I) = \sqrt[2]{(-0.2-0)^2 + (0.6-0)^2} = 0.632$$

Para el subnivel ACD, es el valor uno el que representa la no modificación de la etiqueta original, por tanto

$$Difer_{ACD}(I) = \sqrt[2]{(0.5-1)^2 + (2-1)^2} = 1.118$$

La medida anteriormente definida establece un criterio claro para valorar la brusquedad de la alteración de las etiquetas con respecto a las que inicialmente se propusieron. En este sentido, incluimos $Difer_{ACD}$, $Difer_{ACA}$ y $Difer_{AD}$ como criterios que deben ser evaluados para establecer la bondad de una descripción, y por tanto, incluidos en la EFL. El problema se plantea al establecer el orden en que estos criterios se deben evaluar. Parece claro, que los criterios ya incluidos en la EFL son más relevantes que estos, y por este motivo mantenemos su orden original. La ordenación de estos tres nuevos criterios entre sí puede ser discutible, pero nosotros optamos por la siguiente: en primer lugar minimizaremos las alteraciones del operador AD, que desde nuestro punto de vista, es el que provoca modificaciones más bruscas sobre la semántica de las etiquetas originales. En segundo lugar, la minimización de las modificaciones del operador ACA y en último lugar las del operador ACD. Situamos en último lugar el operador ACD, ya que es el único modificador semántico que no altera el soporte de las etiquetas iniciales, y por tanto, no provoca cambios bruscos en el significado de las etiquetas.

Por consiguiente, la nueva función $Fitness$ de evaluación de individuos queda definida como una EFL con seis criterios ordenados de la siguiente forma: Dada una regla $R_B(d)$, definimos la valoración de la regla $R_B(d)$ como una EFL con los siguientes criterios:

Orden	Criterio
1	$\max\{\Lambda(R_B(d)) \times \Gamma_{k_1k_2}(R_B(d))\}$
2	$\max\{S(R_B(d))\}$ o $\max\{1 - S(R_B(d))\}$ (discriminante / descriptiva)
3	$\max\{CP(R_B(d)\}$
4	$\min\{Difer_{AD}(d)\}$
5	$\min\{Difer_{ACA}(d)\}$
6	$\min\{Difer_{ACD}(d)\}$

Parámetro	Valor
Número máximo de iteraciones	1000
Número máximo de iteraciones sin mejora	200
Número de individuos en la población	20
Probabilidad de mutación (nivel de valor)	0.001
Probabilidad de cruce (nivel de valor)	0.1
Parámetro de consistencia k_1	0
Parámetro de consistencia k_2	1
Parámetro de recubrimiento λ	0.8
Probabilidad del operador de rotación	0.1
Probabilidad del operador de generalización	0.1
Parámetro de penalización de errores definitivos	1.0
Probabilidad de mutación (nivel de variable)	0.01
Parámetro α del operador BLX_α	0.05
Probabilidad de mutación (nivel ACD, ACA y AD)	0.05
Probabilidad de cruce (nivel ACD, ACA y AD)	0.1

Tabla 5.2: Parámetros fijados para la ejecución de SLAVE con modificadores semánticos

5.4 Pruebas experimentales

Una vez planteado el modelo de aprendizaje con la inclusión de los modificadores, y definidos tanto los operadores genéticos para cada subnivel, como los nuevos criterios en la evaluación de las reglas, deseamos comprobar experimentalmente el comportamiento de este algoritmo de aprendizaje.

Antes de definir el planteamiento general de la experimentación, es necesario realizar algunas consideraciones previas sobre el algoritmo. El hecho de incluir en el proceso las búsqueda de los valores más adecuados de los tres modificadores semánticos que se han definido, provoca un aumento considerable del espacio de búsqueda de las soluciones deseables. Por consiguiente, el número de iteraciones requeridas para que el algoritmo genético converja hacia buenas soluciones debe ser mayor que el utilizado para las anteriores pruebas experimentales. Por esta razón, se han modificado los parámetros propuestos para esta experimentación. Dichos parámetros se muestran en la tabla 5.2.

Los resultados obtenidos usando el algoritmo SLAVE dos niveles con umbral de activación y los anteriores operadores sobre las bases de datos con consecuente crisp se muestran en la tabla 5.3. No se incluyen las bases de datos MONK1, MONK2 y MONK3, ya que todas sus variables predictivas son nominales, por lo que los modificadores semánticos no afectan. Los resultados obtenidos sobre las bases de datos con consecuente difuso se muestran en la tabla 5.4.

Los resultados expuestos muestran claramente una tendencia del algoritmo a ajustarse mucho a los ejemplos del conjunto de entrenamiento. Como se comento en el algoritmo de

Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
IRIS	98.74	92.09	4.4	167
INFARTO	96.70	89.47	2.8	65
WINE	99.34	91.64	4.8	195
PIMA	93.06	68.75	40	9905
IONOSPHERE	99.86	89.38	6	485

Tabla 5.3: Resultados de SLAVE con modificadores semánticos para el caso crisp

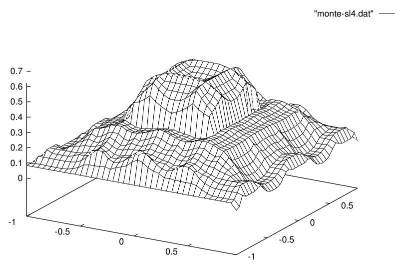
Bases de datos	Entrenamiento	Prueba	N.Reglas	Tiempo
SIN	0.0005	0.0005	7	310
PARABOLA	0.0005	0.0005	3	214
3DPARABOLA	0.0003	0.0009	7	8117
MONTE	0.0009	0.0011	13	18501
PENDULO	0.0978	0.1314	6	512
RICE	0.0038	0.0090	5.6	317

Tabla 5.4: Resultados de SLAVE con modificadores semánticos para el caso difuso

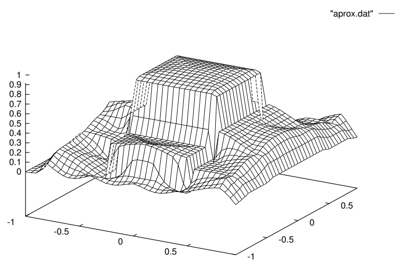
refinamiento, propuesto en el capítulo 2 de esta memoria, este hecho conlleva en la mayoría de los casos a un sobreaprendizaje, provocando una disminución del porcentaje de acierto en los conjuntos de prueba. Justamente esta situación es la que se produce sobre las bases de datos con variable consecuente crisp. Un ejemplo claro de este comportamiento, lo podemos observar sobre la base de datos PIMA, donde se produce un procentaje de acierto del 93.06 % sobre el conjunto de entrenamiento (el más alto de todos los obtenidos sobre este conjunto de ejemplos), mientras que sólo un 68.75 % sobre el conjunto de prueba (el peor de todos los resultados obtenidos sobre esta base de datos por SLAVE).

Con respecto al número de reglas, podemos observar que, exceptuando el elevado número de reglas obtenido para la base de datos PIMA, sus valores coinciden prácticamente con los obtenidos con la versión de SLAVE dos niveles con información, e incluso para algunas bases de datos, este número es inferior al obtenido al aplicar a SLAVE el algoritmo de refinamiento.

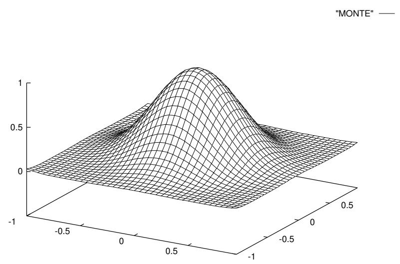
Sobre las bases de datos con variable consecuente difusa, se puede observar una mejora sustancial en la disminución del error sobre el conjunto de entrenamiento, pero al contrario de lo que sucede en el caso crisp, la mejora también se observa en la disminución del error sobre el conjunto de prueba. Esta aparente diferencia de comportamiento entre el caso crisp y el caso difuso del algoritmo tiene una explicación: en las bases de datos crisp, el parecido entre el conjunto de entrenamiento y el conjunto de prueba es menor que el parecido existente entre ambos conjuntos de las bases de datos difusas. Por tanto, si se produce un sobreaprendizaje del conjunto de entrenamiento, el conocimiento extraido representa mejor a aquellos conjuntos de prueba que se asemejan lo más posible al conjunto



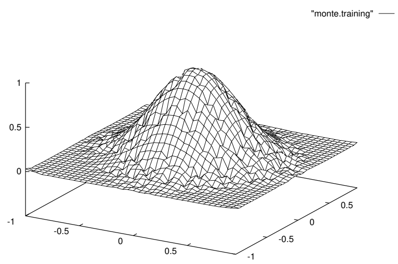
(a) Aproximación obtenida por SLAVE dos niveles con umbral de activación



(b) Aproximación obtenida por el algoritmo WM



(a) Función original



(b) Aproximación obtenida por SLAVE con modificadores semánticos

Figura 5.13: Distintas aproximación obtenidas para la base de datos MONTE

de ejemplos sobre el que se ha aprendido.

Obviamente, al aumentar el espacio de búsqueda, y el número de iteraciones en el algoritmo genético, los resultados del tiempo de respuesta del algoritmo de aprendizaje no son comparables con las versiones propuestas en el capítulo anterior. Sin embargo, si que podemos comparar este parámetro, con los obtenidos por el método de refinamiento propuesto en el capítulo 3, pudiendo observar que, en general, esta nueva aproximación necesita menos tiempo para encontrar el conjunto de reglas que describen el sistema.

De los resultados obtenidos en el caso difuso, debemos destacar la mejora sustancial que se produce con sobre la bases de datos MONTE. En la figura 5.13 se muestran distintas aproximación obtenidas sobre esta base de datos por distintos algoritmos y la función original.

Podemos observar, que el algoritmo SLAVE con los modificadores semánticos propuestos en este capítulo obtiene una aproximación bastante mejor que las que obtienen las versiones anteriores y la que obtiene el algoritmo WM. Además, la similitud entre esta propuesta y la función original, indica que esta aproximación representa correctamente la función de partida.

Las resultados experimentales mostrados en esta sección ponen de manifiesto que la inclusión de operadores semánticos en las descripciones de las reglas es una buena vía para obtener un conocimiento que se aproxime mejor al comportamiento real del sistema. Sin embargo, estos resultados muestran que la implementación aqui utilizada tiene una elevada tendencia al sobre-aprendizaje del conjunto de ejemplos. Por consiguiente, nos proponemos como una futura línea de trabajo la reducción de la tendencia al sobre-aprendizaje del algoritmo propuesto.

Conclusiones

En esta memoria hemos presentado el desarrollo en varias etapas de un algoritmo inductivo de aprendizaje basado en la metodología iterativa para la obtención de reglas difusas, llamado SLAVE. SLAVE usa algoritmos genéticos para determinar en cada iteración del proceso de aprendizaje la mejor regla que representa cada clase teniendo en cuenta la información existente en el conjunto de ejemplos. Una característica fundamental de SLAVE es que ha sido concebido para abordar problemas de clasificación tanto en sutuaciones en las que la variable de clasificación toma valores sobre un refencial crisp, como para aquellas en la que la variable de clasificación toma valores sobre un referencial continuo. Para este segundo tipo de problemas, la aplicación de SLAVE es directa, sin más que discretizar el referencial de la variable de clasificación usando conjuntos difusos.

Una de las características fundamentales de SLAVE es que obtiene la relevancia parcial de las variables de entrada, es decir, durante el proceso de aprendizaje el algoritmo selecciona las características relevantes para construir la descripción de la regla, no considerando para la descripción de la misma, las variables irrelevantes.

En la figura 5.14 se muestra la evolución del algoritmo SLAVE a lo largo de esta memoria. Así, en el capítulo 2 se expuso la metodología básica del algoritmo de aprendizaje. Los resultados obtenidos experimentalmente demuestran que SLAVE tiene un comportamiento para problemas con consecuente crisp, semejante a los resultados que proporcionan otros algoritmos de aprendizaje, aunque únicamente obtiene el mejor resultado para dos de las bases de datos propuestas. El comportamiento del algoritmo mejora con la inclusión de nuevos operadores genéticos definidos para el aprendizaje. Sin embargo, para problemas con consecuente difuso, los resultados de SLAVE no son tan buenos como cabría esperar. Esta deficiencia se debe fundamentalmente a que este segundo tipo de problemas, requieren la colaboración de las reglas extraídas para determinar la clasificación de los ejemplos. Este requisito no se contempla en esta primera versión de SLAVE, ya que las mejores descripciones para cada concepto se aprenden de forma aislada, sin tener en cuenta durante el proceso de aprendizaje, las reglas que se aprendieron para el resto de los conceptos.

Por esta razón, en el capítulo 3 expusimos dos posibles vías para mejorar el comportamiento del algoritmo mediante la inclusión de mecanismos que permitan la colaboración en unos casos, y la competición en otros, de las reglas aprendidas. La primera de las vías sugeridas consiste en la inclusión de un algoritmo de refinamiento de teorías como fase

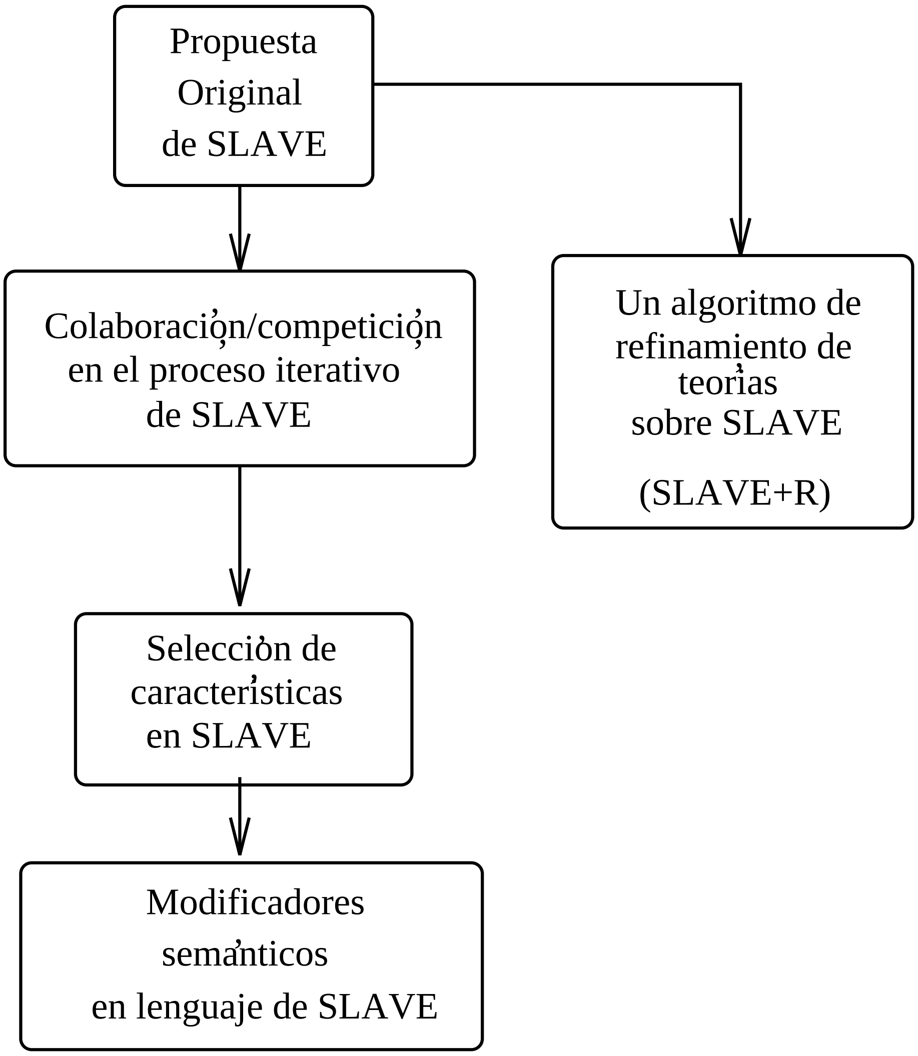


Figura 5.14: Evolución del algoritmo de aprendizaje SLAVE

final del algoritmo de aprendizaje. Este algoritmo de refinamiento será el encargado de establecer de forma conveniente la colaboración o competición entre las reglas, teniendo como objetivo no sólo mejorar la capacidad de predicción de las mismas, sino también mejorar en la medida de lo posible la comprensibilidad del conocimiento extraído.

Como segunda vía, se contempla la posibilidad de que sea el propio algoritmo el que incorpore la colaboración/competición entre las reglas durante el proceso de aprendizaje. Con tal fin, se propuso un nuevo mecanismo de evaluación de reglas, que contempla el conocimiento previamente extraído para los conceptos ya aprendidos, para determinar en que medida las nuevas reglas que se proponen afectarán a la clasificación de los conceptos.

Los resultados experimentales obtenidos en este capítulo ponen de manifiesto que ambas vías son válidas para resolver el problema anteriormente mencionado. Aunque estas dos vías presentan una capacidad de clasificación y un número de reglas parecidos, la segunda mejora sustancialmente el tiempo de respuesta del algoritmo de aprendizaje. Por este motivo, consideramos que la evolución de SLAVE debe seguir esta segunda vía, si se desea obtener un algoritmo que ofrezca un mejor tiempo de respuesta. En cualquier caso, el algoritmo de refinamiento de teorías, aquí propuesto sigue siendo una alternativa válida, no sólo para mejorar las bases de reglas obtenidas por SLAVE, sino para cualquier base de reglas (con un modelo de regla semenjante al aquí propuesto) que proporcione otro algoritmo de aprendizaje o un experto.

Además, de los resultados que se muestran en este capítulo, podemos deducir las siguientes conclusiones:

- La nueva versión de SLAVE mejora sustancialmente su comportamiento para los problemas con consecuente difuso, de tal manera, que mejora para algunas bases de datos los resultados obtenidos por el algoritmo de aprendizaje WM, un algoritmo que muestra tener un buen comportamiento para este tipo de problemas.

- Por el contrario, el algoritmo tiene problemas para seleccionar de forma correcta el conjunto de variables relevantes, cuando el conjunto de ejemplos tiene un número elevado de ellas. Por esta razón, la capacidad de predicción es menor sobre el conjunto de prueba y es elevado el número de reglas que presenta el conocimiento obtenido para la base de datos IONOSPHERE.

Esta última conclusión, nos llevó a proponer en el capítulo 4, una nueva modificación de SLAVE para aumentar su capacidad de detección de variables irrelevantes. Con este propósito, en el módulo de selección de reglas se establecen dos procesos de búsqueda diferentes, uno que trata de encontrar el subconjunto de variables relevantes que formarán parte de la descripción de la regla, y otro que intenta encontrar la mejor asignación de valores a estas variables. Estas dos búsquedas se llevan a cabo de forma simultánea mediante la ejecución de forma paralela de dos algoritmos genéticos. Estos algoritmos genéticos, mediante coevolución, tratan de encontrar la mejor descripción para un concepto. Así, se consigue no sólo agilizar la selección, sino que además, se aumenta el nivel de exploración del espacio de los subconjuntos de variables relevantes.

Se propusieron en este capítulo tres formas posibles para llevar a cabo esta modificación. En la primera de ellas se propone un algoritmo genético para la selección de características que representa para cada variable de entrada, su presencia o ausencia en la descripción de la regla para cada individuo de la población, dejando que la evolución del sistema determine la configuración más adecuada para el concepto. Las otras dos propuestas, para cada variable en el algoritmo genético de selección de características, establecen una medida de información que es interpretada como la probabilidad de que dicha variable pertenezca a la descripción de la regla. Esta medida de probabilidad se obtiene a partir de la información que aporta la distribución de los ejemplos sobre cada uno de los conceptos que se desean aprender. Por consiguiente, el uso de esta información en el algoritmo genético, permite acercar las descripciones que se generan inicialmente a las descripciones que se desean obtener. La diferencia entre estas dos últimas propuestas está en la interpretación que se hace de las medidas de probabilidad: en la primera se utiliza directamente la probabilidad asociada para determinar su pertenencia o no a la descripción de la regla que define cada individuo (modelo por copia). En la segunda, se establece un valor de probabilidad diferente para cada individuo que se interpreta como un umbral de activación, de manera que toda variable que tenga un valor de probabilidad superior al umbral se considera en la descripción de la regla (modelo por umbral de activación). En ambos casos, la población inicial contiene las medidas de información obtenidas del conjunto de ejemplos, y será el proceso evolutivo el que, alterando mediante operadores genéticos dichas medidas, obtenga la configuración más adecuada para la descripción de la regla.

Los resultados experimentales reflejaron, que si bien todas ellas conseguían mantener o mejorar los resultados de las versiones anteriores, el modelo por umbral de activación presentaba los mejores resultados, disminuyendo significativamente el tiempo de respuesta del algoritmo. Por consiguiente, el algoritmo SLAVE dos niveles siguiendo el modelo por umbral de activación demuestra ser el algoritmo más apropiado, tanto para problemas con consecuente crisp como con consecuente difuso.

Aunque SLAVE dos niveles siguiendo el modelo por umbral de activación es un algoritmo de aprendizaje que produce muy buenos resultados, en el capítulo 5 nos planteamos la posibilidad de mejorar aún más los resultados obtenidos previamente. Para mejorar estos resultados era necesario dotar al algoritmo de la capacidad de alterar la semántica de sus etiquetas iniciales, de manera que éstas respondieran lo mejor posible a la información que contienen los ejemplos. Con esta idea, se propuso enriquecer el lenguaje de representación de SLAVE mediante la inclusión de modificadores semánticos. Las pruebas realizadas sobre este nuevo algoritmo ponen de manifiesto que si el conjunto de entrenamiento contiene un número suficiente de ejemplos que representen ampliamente el comportamiento del sistema, los resultados son muy buenos. En este sentido, cabe destacar que para el caso difuso, se mejora para todas las bases de datos los resultados obtenidos por el algoritmo WM. Por el contrario, se observa un sobreaprendizaje si la base de datos contiene pocos ejemplos.

Como conclusión final, resaltamos que se han conseguido los objetivos propuestos, proporcionando un algoritmo de aprendizaje que presenta un buen comportamiento en los dos tipos de problemas de clasificación planteados, dotando de un alto grado de comprensibilidad desde el punto de vista humano al conocimiento extraído.

Las líneas futuras de trabajo de esta memoria se pueden resumir en:

1. Profundizar en el problema planteado en el capítulo5 para tratar de reducir la tendencia al sobre-aprendizaje que provoca la inclusión de los modificadores semánticos en la descripción de las reglas.

2. La obtención de un algoritmo más eficiente sobre el modelo presentado en la memoria considerando fundamentalmente dos alternativas: la primera consiste en dotar

al algoritmo de la capacidad de autoadaptación de los parámetros de entrada del mismo. En las últimas propuestas de SLAVE se requieren muchos parámetros de entrada de los cuales depende la bondad de los resultados obtenidos. La idea consiste en introducir algoritmos genéticos autoadaptativos que permitan, durante el proceso de búsqueda, encontrar estimaciones correctas a estos parámetros que mejoren el tiempo de respuesta de SLAVE. La segunda de las alternativas, es sustituir los algoritmos genéticos del algoritmo de aprendizaje, por otros algoritmos de búsqueda que permitan obtener más rápidamente las soluciones deseables.

3. Dotar a SLAVE de la capacidad de readaptar su conocimiento ante la entrada de nuevos ejemplos, es decir, convertir a SLAVE en un algoritmo de aprendizaje incremental.

4. Aumentar la capacidad de descripción de SLAVE, incluyendo la posibilidad de determinar relaciones entre las variables de entrada para generar nuevos atributos. En este sentido, SLAVE se convertiría en un algoritmo que aprende relaciones entre los atributos iniciales.

Apéndice A

Bases de datos con variable de clasificación crisp

Para la confección de la parte experimental de esta memoria, se han seleccionado algunas bases de datos. Estas bases de datos reflejan los diversos problemas a los que se puede enfrentar un algoritmo de aprendizaje.

De cada base de datos discribiremos cada uno de los atributos. Para cada atributo aportaremos la siguiente información:

- **Variable**: Identificador utilizada para identificar a dicho atributo en la descripción de las reglas.
- **Referencia**: Tipo del universo de discurso de la variable que puede tomar únicamente tres valores: Nominal, Discreto o Continuo.
- **Rango**: Describe el universo de discurso de las variables con referencial discreto o continuo. En el caso de que el atributo sea nominal, se describe el número de valores que dicho atributo puede tomar.
- **Val. perdidos**: Indica si este atributo contiene valores perdidos.
- **N. etiquetas**: Indica el número de etiquetas difusas definidas para discretizar los universos de discurso de las variables continuas o discretas. En el caso de variables nominales, este valor coincide con mismo asignado al rango. Para el caso discreto y continuo, asumiremos que el número de etiquetas se distribuye de forma uniforme sobre el rango de definición cortandose a grado 0.5.

La bases de datos que más adelante pasaremos a describir, se han obtenido del UCI Respository of Machine Learning Databases and Domain Theories, cuya dirección ftp es *ftp.ics.uci.edu*, excepto la base de datos INFARTO.

A continuación describimos las diferentes bases de datos utilizadas.

- **Base de datos IRIS**

  Descripción: Esta es una de las bases de datos más conocidas y utilizadas para mostrar el comportamiento de los algoritmos de clasificación. Fué propuesta en [38] y propone la clasificación de tres clases de plantas usando cuatro variables predictivas, usando 150 ejemplos.

  Para la parte experimental, se han construido 5 particiones en ficheros de entrenamiento y prueba. Dichas particiones se han realizado de forma completamenta aleatoria, de manera, que aproximadamente el 70 % de los ejemplos están en el conjunto de entrenamiento, y el 30 % en el conjunto de prueba.

  La variable de clasificación es nominal y puede tomar los tres siguientes valores: *Setosa, Virginica, Versicolor.*

Variable	Referencial	Rango	Val. perdidos	N. etiquetas
Sepal_l	Continuo	[4,8]	No	5
Sepal_w	Continuo	[2,4.5]	No	5
Petal_l	Continuo	[1,7]	No	5
Petal_w	Continuo	[0,3]	No	5

Tabla A.1: Base de datos IRIS

- **Base de datos INFARTO**

  Descripción: Esta base de datos fué creada por la profesora Dña Aurora Valenzuela, miembro del Departamento de Medicina Legal y Forense de la Universidad de Granada. A su condición de profesora, une la de ser medico-forense del Hospital Clínico de la ciudad de Granada. Los datos representan análisis reales realizados por la propia doctora sobre pacientes. La base de datos contiene 14 variables para determinar una variable nominal de diagnóstico. Dos de las variables utilizadas son nominales, el resto tienen un referencial continuo. Las variables continuas se pueden dividir en dos diferentes de pruebas: métodos morfológicos y análisis bioquímicos. Estas pruebas se aceptan como los mejores marcadores para el diagnóstico post-morten del infarto de miocardio. En algunos trabajos, tales como [52, 78], se ha utilizado esta base de datos. La base de datos INFARTO tiene 78 ejemplos y presenta valores perdidos.

  Para la parte experimental, se han construido 5 particiones en ficheros de entrenamiento y prueba. Dichas particiones se han realizado de forma completamenta aleatoria, de manera, que aproximadamente el 70 % de los ejemplos están en el conjunto de entrenamiento, y el 30 % en el conjunto de prueba.

  La variable de clasificación puede tomar dos valores: *INFARTO y NO-INFARTO.*

- **Bases de datos MONK**

Variable	Referencial	Rango	Val. perdidos	N. etiquetas
ApoA	Continuo	[0,140]	No	5
ApoB	Continuo	[0,140]	No	5
Lp(a)	Continuo	[0,2]	No	5
HDL	Continuo	[0,100]	No	5
LDL	Continuo	[0,150]	No	5
PARAOXONASA	Continuo	[0,200]	Si	5
SEXO	Nominal	2	No	2
EDAD	Discreto	[0,120]	No	5
MIOGLEB	Discreto	4	No	4
NTZ	Discreto	2	No	2
K/Na(A)	Continuo	[0,2]	No	5
K/Na(B)	Continuo	[0,2]	No	5
K/Na(C)	Continuo	[0,2]	No	5
K/Na(D)	Continuo	[0,2]	No	5

Tabla A.2: Base de datos INFARTO

Descripción: Los problemas del MONK fueron la base de la primera comparación internacional de algoritmos de aprendizaje. Los resultados de esta comparación se presentaron en [148]. Hay tres problemas del MONK. Los dominios para cada uno de ellos son los mismos, y además, se construyó una partición en ejemplos de prueba y en ejemplos de entrenamiento para cada uno de ellos.

Cada problema se compone de 432 ejemplos, seis variables discretas predicitivas, y una variable de clasificación, también discreta que toma dos valores: *0, 1*.

Variable	Referencial	Rango	Val. perdidos	N. etiquetas
a1	Discreto	3	No	3
a2	Discreto	3	No	3
a3	Discreto	2	No	2
a4	Discreto	3	No	3
a5	Discreto	4	No	4
a6	Discreto	2	No	2

Tabla A.3: Base de datos MONK

Cada problema representa un comportamiento diferente sobre los mismos atributos. El comportamiento que describen los ejemplos en cada uno de ellos es el siguiente:

- **MONK1**: El valor de la variable consecuente es 1, si a1=a2 o a5=1. En otro caso la variable consecuente toma valor 0.
- **MONK2**: La variable consecuente toma valor 1 si se satisfacen estrictamente dos de las siguientes condiciones: a1=1, a2=1, a3=1, a4=1, a5=1, a6=1. En

otro caso el valor de la variable consecuente es 0.

- **MONK3**: El valor de la variable consecuente es 1, si (a5=3 y a4=1) o (a5$\neq$4 y a2 $\neq$ 3). En otro caso la variable consecuente toma el valor 0. En esta base de datos, hay que destacar que se incluye un 5 % de ruido sobre el conjunto de entrenamiento.

- **Base de datos WINE**

  Descripción: Esta base de datos es el resultado de un análisis químico de vinos de un misma región con diferentes tipos de uva, usando 13 variables continuos y 178 ejemplos. Hemos usado 7 etiquetas lingüísticas uniformemente distribuidas para discretizar el rango de cada variable.

  La variable de clasificación es nominal, y puede tomar los siguientes tres valores: *Tipo1, Tipo2, Tipo3.*

  Para la parte experimental, se han construido 5 particiones en ficheros de entrenamiento y prueba. Dichas particiones se han realizado de forma completamenta aleatoria, de manera, que aproximadamente el 70 % de los ejemplos están en el conjunto de entrenamiento, y el 30 % en el conjunto de prueba.

Variable	Referencial	Rango	Val. perdidos	N. etiquetas
Att1	Continuo	[11,15]	No	7
Att2	Continuo	[0.74,5.8]	No	7
Att3	Continuo	[1.36,,3.23]	No	7
Att4	Continuo	[10.6,30]	No	7
Att5	Continuo	[70,162]	No	7
Att6	Continuo	[0.9,3.88]	No	7
Att7	Continuo	[0.34,5.08]	No	7
Att8	Continuo	[0.13,0.66]	No	7
Att9	Continuo	[0.41,3.58]	No	7
Att10	Continuo	[1.2,13]	No	7
Att11	Continuo	[0.48,1.71]	No	7
Att12	Continuo	[1.27,4]	No	7
Att13	Continuo	[278,1680]	No	7

Tabla A.4: Base de datos WINE

- **Base de datos PIMA**

  Descripción: Esta base de datos es una muestra extraída sobre una población cercana a Fenix, Arizona, EEUU sobre la diagnosis de la diabetes tomando como marco de referencia los signos propuestos por la Organización Mundial de la Salud. La base de datos consta de 768 ejemplos, dividos en 576 de entrenamiento y 192 de prueba. Esta base de datos se compone de 8 variables predictivas continuas más

Variable	Referencial	Rango	Val. perdidos	N. etiquetas
Pregnant	Continuo	[0,17]	No	5
Glucosa	Continuo	[0,199]	No	5
Presión	Continuo	[0,122]	No	5
Triceps	Continuo	[0,99]	No	5
Insulina	Continuo	[0,846]	No	5
Body_mass	Continuo	[0,67.1]	No	5
Pedigree	Continuo	[0.08,2.42]	No	5
Age	Discreto	[21,81]	No	5

Tabla A.5: Base de datos PIMA

una variable de clasificación binaria, donde el valor 1 se intrepreta como que el test de diabetes es positivo.

- **Base de datos IONOSPHERE**

  Descripción: Estos datos de radar fuerón recogidos por un sistema en Goose Bay, Labrador [141]. Este sistema consiste en un vector que contempla valores de 16 antenas de alta frecuencia con una potencia total transmitida del orden de 2.4 Kilowatios. El objetivo consistía en detectar electrones libres en la ionosfera. La base de datos está compuesta de 34 atributos continuos además de la variable de clasificación con 351 ejemplos.

  Para la parte experimental, se han construido 5 particiones en ficheros de entrenamiento y prueba. Dichas particiones se han realizado de forma completamenta aleatoria, de manera, que aproximadamente el 70 % de los ejemplos están en el conjunto de entrenamiento, y el 30 % en el conjunto de prueba.

  La variable de clasificación es nominal y puede tomar los siguientes dos valores: *tipo-g, tipo-b*. Todas las variables tienen un referencial común, para las cuales se construye un mismo dominio constituido por siete etiquetas.

- **Base de datos SOYBEAN**

  Descripción: Esta base de datos se tomó como punto de partida para el desarrollo de un sistema experto para la diagnosis de enfermedades de las habas [111]. El conjunto de ejemplos contiene 35 variables predictivas nominales o discretas y una variable de clasificación con 19 clases. Hay 316 ejemplos que contienen valores perdidos para muchas de las variables predictivas que se dividen en una única partición en fichero de entrenamiento y prueba.

Apéndice B

Bases de datos con variable consecuente difuso

En este apéndice se describen las bases de datos con consecuente difuso utilizadas en la parte experimiental de esta memoria. Debemos destacar que todas las bases de datos se han generado por el autor de este trabajo, excepto la base RICE que se ha obtenido del trabajo [119]. Sobre todas, se han construido cinco particiones de entrenamiento y prueba para evitar el sesgo de los resultados obtenidos. Todas las bases de datos, se han discretizado usando siete etiquetas difusas uniformemente repartidas sobre el referencial de cada variables, excepto en el caso de RICE que se han considerado únicamente cinco.

A continuación pasamos a describir cada una de las bases de datos.

- **Base de datos SIN**

  Descripción: Esta base de datos contiene 1000 ejemplos obtenidos por la aplicación de la función $\sin(x)$ donde $x \in [0, 1]$.

- **Base de datos PARABOLA Descripción:** Esta base de datos contiene 1000 ejemplos obtenidos por la aplicación de la función x^2 donde $x \in [0, 1]$.

- **Base de datos 3DPARABOLA**

  Descripción: Esta base de datos contiene 1000 ejemplos obtenidos por la aplicación de la función $f(x, y) = x^2 + y^2$ donde $x, y \in [-5, 5]$.

- **Base de datos MONTE**

  Descripción: Esta base de datos contiene 1000 ejemplos obtenidos por la aplicación de la función $f(x, y) = (cos^2(x) + cos^2(y))^4$ y que se muestra la figura B.4. La base de datos contiene dos variables predictivas $x, y \in [-1, 1]$.

- **Base de datos PENDULO**

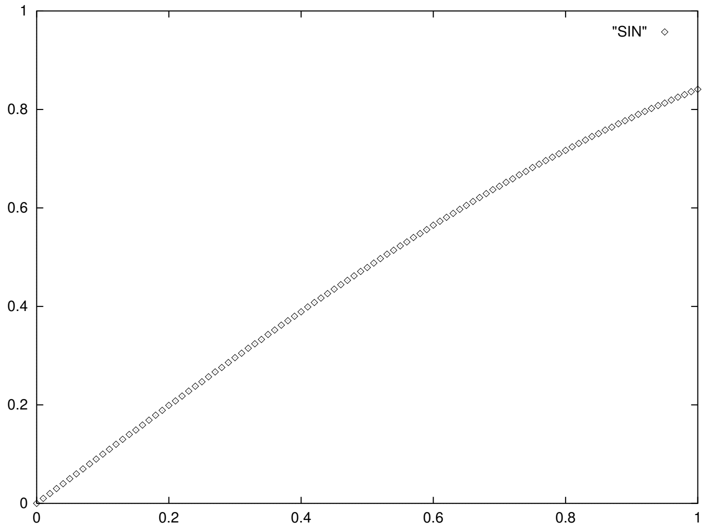


Figura B.1: Representación de la base de datos SIN

Descripción: El problema del péndulo es un problema tradicional de control y consiste en que, dado un conjunto de ejemplos que describe el comportamiento del sistema, deseamos aprender el conjunto de reglas que permiten controlarlo. Si asumimos que el valor de $|\theta| << 1$ (radianes) la equación diferencial no lineal que describe el comportamiento del péndulo es

$$m\frac{l^2}{3}\frac{d^2\theta}{dt^2} = \frac{l}{2}(-f + m\ g\ sin\theta - k\frac{d\theta}{dt})$$

donde $k\frac{d\theta}{dt}$ es una aproximación de la fuerza de fricción.

Por tanto, el sistema se puede describir a través de dos variables de estado θ (ángulo) y ω (velocidad angular) y la variable de control f (fuerza). Para la simulación real se ha considerado un péndulo de 5 kg de peso y 5 m de largo, al que se han aplicado fuerzas sobre el centro de gravedad cada 10 ms. Con estos parámetros, el universo de discurso de las variables son los siguientes:

$$\theta \in [-0.277, 0.277] \quad \omega \in [-0.458, 0.458] \quad f \in [-1593, 1593]$$

De forma experimental, hemos obtenido dos conjuntos de ejemplos bajo las anteriores condiciones, el primero de ellos tiene 213 ejemplos y se ha usado para aprender (conjunto de entrenamiento) y el segundo contiene 125 ejemplos y se ha utilizado para comprobar el comportamiento de las reglas aprendidas (conjunto de test). Estos conjuntos de ejemplos se han obtenido de las ecuaciones anteriores teniendo en cuenta dos condiciones iniciales diferentes:

a) θ=-0.277 y ω=0.

b) θ=0.277 y ω=0.

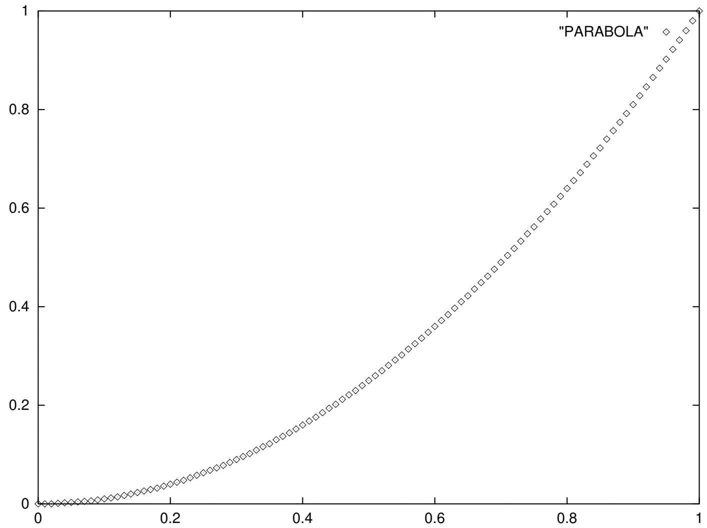


Figura B.2: Representación de la base de datos PARABOLA

El primer paso para aprender consiste en discretizar el rango de las variables mediante etiquetas difusas. La figura B.5 muestras las etiquetas construidas para cada variable, mientras que la figura B.6 muestra la función que describe el comportamiento del péndulo.

- **Base de datos RICE**

  Descripción: Esta base de datos se ha obtenido a través del trabajo [119]. Los datos se han extraído de un problema real. La base de datos RICE tiene cinco variables de entrada y una salida cuyos valores estan asociados con las siguientes medidas subjetivas: sabor, apariencia, gusto, viscosidad y dureza. La variable de salida establece una evaluación general del arroz. La base de datos contiene 105 ejemplos que muestran 105 tipos diferentes de arroz.

  Para este problema, se han considerado cinco particiones en cada variable para construir el dominio difuso asociado tanto a las variables de entrada, como a la variable de salida.

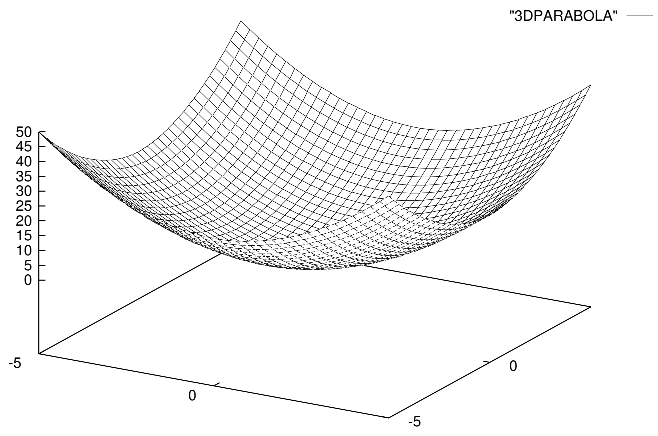


Figura B.3: Representación de la base de datos 3DPARABOLA

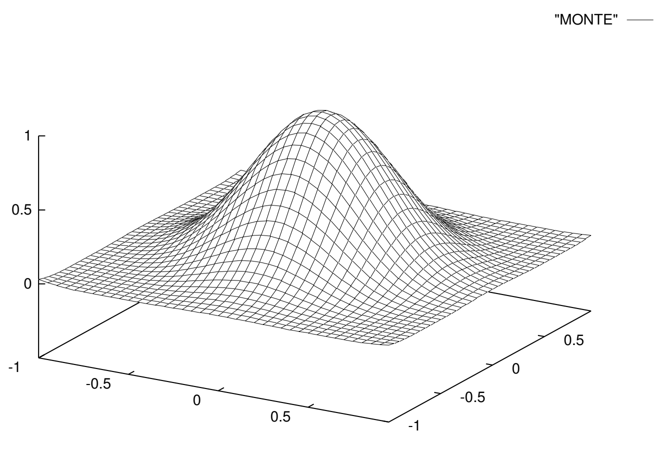


Figura B.4: Representación de la base de datos MONTE

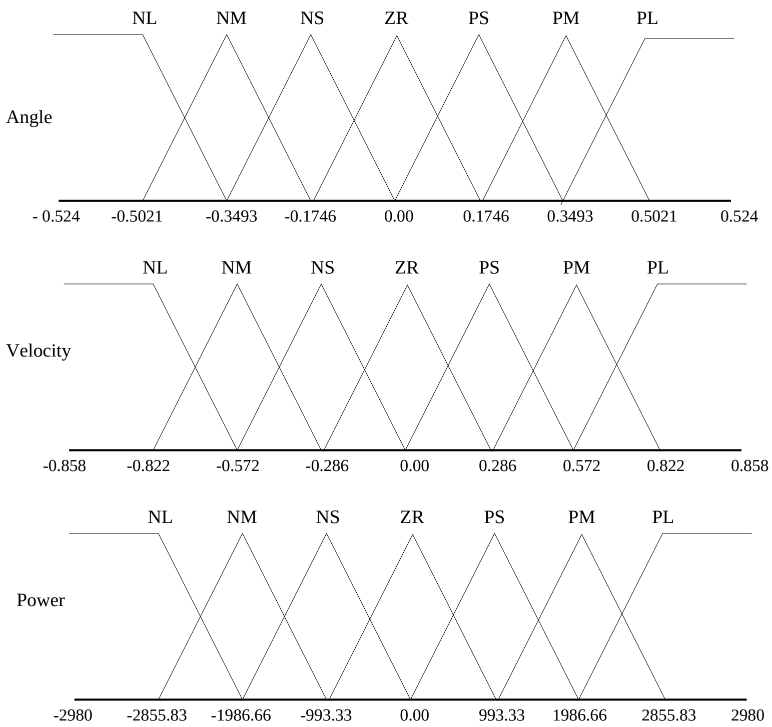


Figura B.5: El problema del péndulo

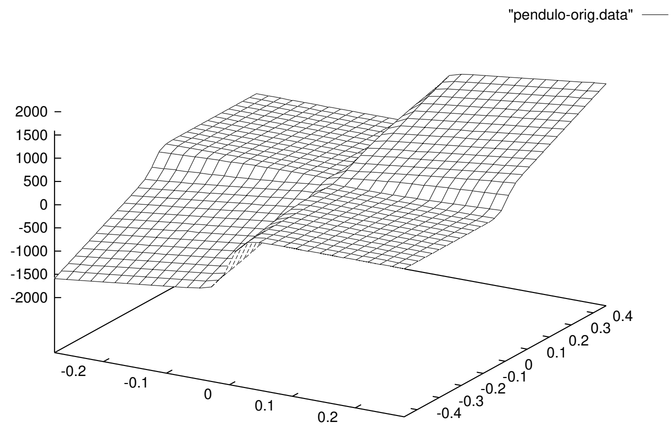


Figura B.6: Superficie original del problema del péndulo

Apéndice C

Algoritmos de aprendizaje utilizados para la confección de la parte experimental

Para determinar la bondad del algoritmo de aprendizaje que se propone en esta memoria, se han estudiado sus resultados con los que obtienen otras algoritmos muy conocidos y ampliamente referenciados en la bibliografía del aprendizaje automático. Para llevar a cabo esta comparación hemos utilizado implementaciones de tales algoritmo que se han obtenido de diferentes formas.

Antes de describir el software utilizado, debemos destacar, que todas las pruebas se han ejecutado sobre diferentes estaciones de trabajo (workstations) de similares características con la versión SOLARIS del sistema operativo UNIX.

A continuación describimos brevemente las características fundamentales de cada una de las implementaciones seleccionadas:

- **SLAVE:** El desarrollo de este software es el núcleo central de esta memoria. Las diferentes versiones utilizadas en este trabajo han sido desarrolladas por este autor, utilizando para ello el lenguaje de programación C. Además, el software utiliza librerias *xview* para desarrollar un interfaz gráfico que facilita el uso de este software. Este entorno gráfico, permite cambiar al usuario de forma muy simple los diferentes parámetros que son necesarios en la ejecución de SLAVE.

- **Algoritmo C4.5:** Esta es una implementación del conocido algoritmo de clasificación del mismo nombre basado en árboles de decisión. Este software se incluye en [128]. Los parámetros utilizados para la confección de la parte experimental, son los que propone el propio software por defecto.

- **Algoritmo LVQ:** Este software contiene todos los programas necesarios para la aplicación del algoritmo LVQ (Learning Vector Quantization) a la tarea de la clasificación estadística o del reconocimiento de patrones. LVQ es un método de aprendizaje adaptativo basado en los mapas auto-organizativos de Kohonen. Una descripción más detallada del mismo se puede consultar en [90].

La implementación utilizada en este trabajo corresponde a la versión 3.1. de LVQ PAK, y que se puede obtener en *ftp://cochlea.hut.fi/pub/lvq_pak*.

Hemos cogido los parámetros entre los rangos recomendados por los autores. El número de prototipos es aproximadamente un tercio del tamaño del conjunto de entrenamiento, el número de iteraciones es cien veces el número de prototipos y el parámetro α se ha tomado a 0.03.

- **Algoritmo CN2:** Este algoritmo aprende inductivamente un conjunto de reglas proposicionales a partir de un conjunto de ejemplos. El algoritmo fué diseñado por Niblett T. y Clark P. [19] y puede considerarse como un algoritmo que sigue la filosofía de los algoritmos AQ y que trata de mejorar el comportamiento de estos ante la presencia de ruido en el conjunto de entrenamiento.

  La implementación del software utilizada en esta memoria, fué realizada por Boswell R. en 1990 y puede obtenerse en *http://www.cs.utexas.edu/users/pclark/software.html*.

  Los parámetros utilizados para la obtención de los resultados presentados en este trabajo con este software, son los que dicha implementación asume por defecto.

- **Algoritmo WM:** Este software se ha utilizado para la clasificación de problemas con consecuente difuso. La implementación utilizada ha sido desarrollada por Cordón O. a partir de la descripción de la metodología de aprendizaje que se propone en [159].

Bibliografía

[1] Almuallim H., Dietterich T.G., *Learning with many irrelevant features.* Proceedings of Ninth National Conference on Artificial Intelligence. (1991), pp. 547-552.

[2] Almuallim H., Dietterich T.G., *Efficient algoritms for Identifying Relevant Features* (1993)

[3] Antonisse J., *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint.* Proceeding of the Third Inter. Conf. on Genetic Algorithms, J. David Schaffer (Ed.) (Morgan Kaufmann Publishers, San Mateo, 1989), pp. 86-91.

[4] Bäck T., Hoffmeister F., *Genetic Algorithms and Evolution Strategies: Similarities and Differences.* Technical Report No. SYS-1/92. Dept. of Computer Science. Universidad de Dortmund, Alemania, Febrero, (1992).

[5] Bäck T., Schwefel H-P., *An Overview of Evolutionary Algorithms for Parameter Optimization.* Evolutionary Computation 1(1), (1993), pp. 1-23.

[6] Barone J.M., *Pruning Fuzzy Decision Trees.* Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks, (1992), pp. 321-324.

[7] Bethke A.D., *Genetic Algorithms as Function Optimizers.* Ph. D. Dissertation, University microfilms No. 8106101, University of Michigan, Ann Arbor, MI (1981).

[8] Blumer A., Ehrenfeucht A., Haussler D., Warmuth M., *Learnability and the Vapnik-Chervonenkis Dimension.* Journal of ACM, 36 (4), (1990) pp. 929-965.

[9] Bongard N., *Pattern Recognition.* Spartan Books, New York, (1970).

[10] Bonelli P., Parodi A., Sen S., Wilson S. *NEWBOOLE: A Fast GBML System.* Proceedings of the Seventh International Conference on Machine Learning, (1990), pp. 153-159.

[11] Booker, L.B., *Intelligent Behavior as an Adaptation to the Task Enviroment.* Doctoral Dissertation. Department of Computer and Communication Science, University of Michigan, Ann Arbor, (1982).

[12] Booker, L.B., Goldber, D.E., Holland, J.H., *Classifier Systems and Genetic Algorithms* Artificial Intelligence 40 (1989) pp. 235-282.

[13] Breiman L., Friedman J.H., Olshen R.A., Stone C.J., *Classification and Regression Trees.* Wadsworth International Group, Belmont, California (1984).

[14] Buchanan B.G., Feigenbaum E.A., Lederberg J., *A Heuristic Programming Study of Theory Formation in Sciences.* Proceedings of the Second International Joint Conference on Artificial Intelligence, London (1971), pp. 40-48.

[15] Buchanan B.G., Michell T.M., *Model-Directed Learning of Production Rules.* Pattern-Directed Inference Systems, Academic Press, New York (1978)

[16] Carse, B., Fogarty, T.C., *A Fuzzy Classifier System Using the Pittsburgh Approach.* Parallel Problem Solving from Nature - PPSN III, Springer-Verlag, (1994), pp. 260-269.

[17] Cestnik B., Kononenko I., Bratko I., *ASSISTANT86: A Knowledge-Elicitation Tool for Sophisticated Users*. Progress in Machine Learning, (1987), pp. 31-45.

[18] Cheng J., Fayyad U.M., Irani K.B., Qian Z., *Improved decision trees: A generalized version of ID3*. Proceedings of the Fifth International Conference on Machine Learning, (1988), pp. 100-107.

[19] Clark P., Niblett T., *Learning If Then Rules in Noisy Domains*. TIRM 86-019, The Turing Institute, Glasgow (1986).

[20] Cohen B.L., *A Powerful and Efficient Structural Pattern Recognition System*. Artificial Intelligence, Vol. 9, No 3, (1977).

[21] Cordón O., del Jesús M.J., Herrera F., López E., *Selecting Fuzzy Rule-Based Classification Systems with Specific Reasoning Methods Using Genetic Algorithms*. Proceeding of IFSA'97, (1997).

[22] Cordón O., Herrera F., *A Three-Stage Process for Learning Descriptive and Approximative Fuzzy Logic Controller Knowledge Bases from Examples*. Por aparecer en Intl. Journal of Approximate Reasoning. (1997).

[23] Cordón O., Herrera F., *A Two-Stage evolucionary process for designing TSK fuzzy rule based system*. Technical Report #DECSAI-97115 (1997).

[24] Darwin C., *On the Origin of Species by Means of Natural Selection*, John Murray, 1859.

[25] Davies W.D.T., *System Identification for Self-Adaptative Control*. Wiley-Interscience, Wiley and Sons Ltd. (1970).

[26] Davis R., Lenat D.B., *Knowledge Based Systems in Artificial Intelligence*. McGraw-Hill, (1981).

[27] Davis L.D., *Applying Adaptive Algorithms to Epistatic Domains*, Proceedings IJCAI-85, Los Angeles, CA (1985), pp. 162-164.

[28] Davis L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, (1991).

[29] De Jong K.A., *An Analysis of the Behavior of a Class of Genetic Adaptative Systems*. Ph. D. Dissertation, University microfilms No. 76-9381, University of Michigan, Ann Arbor, MI (1975).

[30] De Jong K., *Learning with Genetic Algorithms: An overview*. Machine Learning 3, pp. 121-138 (1988).

[31] De Jong K.A., Spears W.M., Gordon D.F., *Using Genetic Algorithms for Concept Learning*. Machine Learning, 13, (1993) pp. 161-188.

[32] Dietterich T.G., *The Metodology of Knowledge Layers for Inducing Descriptions of Sequentially Ordered Events*. Master's thesis, University of Illinois, Urbana, (1979).

[33] Dorigo M., *New Perspectives about Default Hier Formation in Learning Classifier Systems*. Technical Report No. 91-002. Dipartamento di Elettronica, Politecnico di Milano Italy (1991).

[34] Driankov D., Hellendoorn H., Reinfrank M., *An Introduction to Fuzzy Control*. Springer-Verlag (1993).

[35] Eshelman L.J., Schaffer J.D., *Real-Coded Genetic Algorithms and Interval-Schemata*. Foundations of Genetic Algorithms 2, L.D. Whitley (Ed.) (Morgan Kaufmann Publishers, San Mateo, (1993), pp. 187-202.

[36] Feigenbaum E.A., *The Simulation of Verbal Learning Behavior*. Computers and Thought, McGraw-Hill, New York (1963), pp. 297-309.

[37] Fensel D., Wiese M., *Refinement of Rule Sets with JoJo.* Lectures Notes in Artificial Intelligence 667, (1993), pp. 378-383 .

[38] Fisher, R.A., *The use of multiple measurements in taxonomic problems.* Annual Eugenics. **7** (1936) Part II pp. 179–188.

[39] Firlej M., Hellens D., *Knowledge Elicitation : A Practical Handbook.* Prentice Hall, (1991).

[40] Fogel D.B., *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*, Gin Press, Needham Heights, 1991.

[41] Fogel L.J., Owens A.J., Walsh M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, New York, 1966.

[42] Friedberg R.M., *A Learning Machine: Part 1.* IBM Journal, Vol. 2 (1958). pp. 2-13.

[43] Friedberg R.M., Dunham B., North T., *A learning Machine: Part 2.* IBM Journal of Research and Develpment, Vol 3, No. 3 (1959), pp. 282-287.

[44] Fu K.S., *Pattern Recognition and Machine Learning.* Plenum Press, New York, (1971).

[45] Fu K.S., Tou J.T., *Learning Systems and Intelligent Robots.* Plenum Press, New York, (1974).

[46] Goldberg D.E., *Computer-aided gas pipeline operation using genetic algorithms and rule learning.* Doctoral Dissertation, Universidad de Michigan, Ann Arbor, MI (1983).

[47] Goldberg D.E., Richardson J. *Genetic Algorithms with sharing for Multimodal Function Optimization.* Proceedings of the Second International Conference on Genetic Algorithms. (1987).

[48] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, New York, (1989).

[49] De Jong, K., Spears, W.M., Gordon, D.F., Using genetic algorithms for concept learning, Machine learning, 13, pp. 161-188, (1993).

[50] Goldberg D.E, *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking.* Complex Systems 5, 1991, pp. 139-167.

[51] González A., Pérez R., Verdegay J.L., *Learning the Structure of a Fuzzy Rule: an Genetic Approach.* Proceedings del First European Congress on Fuzzy and Intelligent Technologies, Vol 2, (1993), pp. 814-819, también publicado en Fuzzy Systems and Artificial Intelligence, Vol 3, 1, (1994), pp. 57-70.

[52] González, A., Pérez, R., Valenzuela A., *Diagnosis of Myocardial Infarction through Fuzzy Learning Techniques.* Proceedings de IFSA'95 Vol 1, (1995), pp. 273-276.

[53] González, A., Pérez, R.: *Completeness and Consistency conditions for learning fuzzy rules.* Technical Report #DECSAI-95103 1995, también por aparecen en Fuzzy Sets and Systems (1998).

[54] González, A., *A learning methodology in uncertain and imprecise environments*, International Journal of Intelligent Systems, Vol 19, (1995), pp. 357-371.

[55] González, A., Pérez, R., *A Learning System of Fuzzy Control Rules.* In: F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, (1996), pp. 202-225.

[56] González, A. Herrera, F., *Multi-stage Genetic Fuzzy Systems Based on the Iterative Rule Learning Approach*, Mathware and Soft Computing, Vol 4, n. 3, (1997), pp. 233-249.

[57] González, A., Pérez, R., *A Refinement Algorithm of Fuzzy Rules for Classification Problems.* Proceedings de la Sixth Int. Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems, Granada, (1996), 533-538.

[58] González, A., Pérez, R., *Aplicación de un Sistema de Refinamiento de Reglas a Problemas de Clasificación*, Memoria del V Congreso Iberoamericano de Inteligencia Artificial, Editorial Limusa (1996), pp. 20-29.

[59] González, A. Pérez, R., *A two level genetic fuzzy learning algorithm for solving complex problems.* Proceedings de IFSA'97, Vol 2, (1997), pp. 192-197.

[60] González, A. Pérez, R., *Using information measures for determining the relevance of the predictive variables in learnings problemas*, proceedings del FUZZ-IEEE'97, Vol 3, (1997), pp. 1423-1428.

[61] González, A. Pérez, R., *A fuzzy theory refinement algorithm*, por aparecer en International Journal of Aproximate Reasoning, (1998).

[62] González, A. Pérez, R., *SLAVE: A genetic learning system based on an iterative approach*, Sometido al IEEE Transaction on Fuzzy System.

[63] Grefenstette J.J. *Credit Assignment in Rule Discovery Systems based on Genetic Algorithms.* Machine Learning 3 (1988), pp. 225-245.

[64] Grefenstette J.J., *Lamarckian learning in multi-agent enviroments.* Proceedings of the Fourth International Conference on Genetic Algoritms, (1991), pp. 303-310.

[65] Hàjek P., Havránek T., *Mechanizing Hypothesis Formation: Mathematical Foundations for a General Theory.* Springer-Verlag. (1978).

[66] Hayes-Roth F., *Schematic Classification Problem and their Solution.* Pattern Recognition, Vol 6, (1974), pp. 105-113.

[67] Hayes-Roth F., McDermott J., *An Interference Matching Technique for Inducing Abstractions.* Communications of the ACM, Vol. 21, No 6, (1978), pp. 401-410.

[68] Hellendoorn, H., Driankow, D., Reinfrank, M., *An introduction to Fuzzy Control.* Springer-Verlag (1993).

[69] Herrera F., Lozano M., Verdegay J.L., *Tuning Fuzzy Logic Controllers by Genetic Algorithms.* International Journal of Approximate Reasoning 12, (1995), pp. 299-315.

[70] Highleyman W.H., *Linear Decision Functions, with Applications to Pattern Recognition.* Proceedings of IRE, No. 50 (1967), pp. 1501-1504.

[71] Hilgard E.R., Bower G.H., *Theories of Learning* Third Edition, Appleton-Century-Grofts, New York, (1966).

[72] Holland J.H., *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, (1975).

[73] Holland J.H., *Adaptation.* Progress in Theorical Biology, 4. New York: Plenum, (1976).

[74] Holland J.H., Reitman, J.S., *Cognitive Systems Based on Adaptative Algorithms.* Pattern-Directed Inference Systems. New York, Academic Press, (1978).

[75] Holland J.H., *Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases.* Policy Analysis and Information Systems, Vol. 4, No. 3, (1980).

[76] Holland J.H., *Properties of the Bucket Brigade Algorithm.* Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Pittsburgh, PA, (1985), pp. 1-7.

[77] Holland J.H., *Escaping Brittleness.* Machine Learning: An artificial intelligence approach (Vol 2), San Mateo, CA: Morgan Kaufmann, (1986).

[78] Hougen H.P., Villanueva E., Valenzuela A., *Sudden Cardiac Death: a Comparative Study of Morphological, Histochemical and Biochemical Methods.* Forensic Science International 52, (1992), pp. 161-169

[79] Huang D., *The context-array Bucket Brigade Algorithm: An Enhanced Approach to Credit-Apportionment in Classifier Systems.* Proceedings of the Third International Conference on Genetic Algorithms. (1989), pp. 311-316.

[80] Hunt E.B., and Hovland C.I., *Programming a Model of Human Concept Formation*, Computers and Thought, McGraw-Hill, New York, (1963), pp. 310-325.

[81] Hunt E.B., Marin J., Stone P.T., *Experiments in Induction*, Academic Press,, New York, (1966).

[82] Ichikawa Y., Ishii Y., *Retainig Diversity of Genetic Algorithms for Multivariable Optimization and Neural Network Learning.* Proc. IEEE Int. Conf. on Neural Networks, San Francisco, California, (1993), pp. 1110-1114.

[83] Ishibuchi H., Morisawa T., Nakashima T., *Voting Schemes for Fuzzy-Rule-Based Classification Systems.* Proc. Fifth IEEE Intl. Conference on Fuzzy Systems,, pp. 614-620. (1996).

[84] Janikow C.Z., *Inductive Learning of decision rules from attribute-based examples.* A knowledge-intensive genetic algorithm approach. Technical Report No.91-030. The University of North Carolina at Chapel Hill. Departament of Computer Science, (1991).

[85] Janikow C.Z., *A Knowledge-intensive Genetic Algorithm for Supervised Learning.* Machine Learning, 13, (1993) pp. 189-228.

[86] Janikow C.Z., *Fuzzy Decision Tree with Missing Values: FIDMV 1.3.* User's manual. Universidad de Missouri-St.Louis.

[87] John G.H., Kohavi R., Pfleger K., *Irrelevant features and the selection problem.* Machine Learning: Proceedings of the 11th International Conference. Morgan Kaufmann. (1994), pp. 121-129

[88] Kaspinski J., Michalski R.S., *A System that Learns to Recognize Hand-written Alphanumeric Characters.* Technical Report 35, Proce Institute Automatyki, Polish Academy of Sciences, (1966)

[89] Koford T.S., Groner G.F., *The Use of an Adaptive Threshold Element to Design a Linear Optimal Pattern Classifier.* IEEE transactions-Information Theory, Vol. 1T-12, (1966) pp. 42-50.

[90] Kohonen *Self-Organizing Maps.* Springer Series in Information Sciences, Vol. 30, (1995).

[91] Koza J.R., *Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm.* Proc. of the Fourth Int. Conf. on Genetic Algorithms, R. Belew, L.B. Booker (Eds.) (Morgan Kaufmmann, San Mateo), (1991) pp. 37-44.

[92] Koza J.R., *Genetic Programing*, The MIT press, Cambridge, 1992.

[93] Kullback S., *Information Theory and Statistics*, Gloucester, Mass, (1978).

[94] Langley P.W., Neches R., Neves D., Anzai Y., *A domain-independent framework for procedure learning.* Journal of Policy Analysis and Information Systems, Vol 3. No. 2, (1980), pp. 163-197.

[95] Langley P.W., *Selection of relevant features in Machine Learning*. Proceedings of the AAAI Fall Symposium on Relevance. (1994).

[96] Langley P.W., Sage S., *Oblivious decision trees and abstract cases*. Proceedings of the Tenth Conference on Uncertainty in AI. (1994), pp. 399-406.

[97] Larson J., *Inductive inference in the variable-valued predicate logic system VL21: metodology and computer implementation*. Ph.D. dissertation, University of Illinois, (1977).

[98] Lenat D.B., *AM: An artificial intelligence approach to discovery in mathematics as heuristic search*. Ph.D.dissertation, Stardford University, (1976).

[99] Liepins G.E., Vose M.D., *Characterizing Crossover in Genetic Algorithms*. Annals of Mathematics and Artificial Intelligence 5(1), (1992), pp. 27-34.

[100] Lozano M., *Aplicación de técnicas basadas en la lógica difusa para la mejora del comportamiento de los algoritmos genéticos con codificación real*. Tesis Doctoral. Universidad de Granada (1996).

[101] Lucasius C.B., Kateman G., *Applications of Genetic Algorithms in Chemometrics*. Proc. of the Third Int. Conf. on Genetic Algorithms, J.D. Schaffer (Ed.) (Morgan Kaufmann Publishers, San Mateo, 1989), pp. 170-176.

[102] Magdalena L., Velasco J.R., *Fuzzy Rule-Based Controllers that Learn by Evolving their Knowledge Base*. In: F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, 172-201, (1996).

[103] McCallum R.A., Spackman K.A., *Using Genetic Algorithms to Learn Disjuntive Rules from Examples*. Proceedings of the Seventh International Conference on Machine Learning, (1990), pp. 149-152.

[104] McCulloch W.S., Pitts W., *A Logical Calculus of Ideas Imminent in Nervous Activity*. Bull. Math. Biophysics, Vol. 5 (1943) pp. 115-133.

[105] Mendel T., Fu K.S., *Adaptive Learning and Pattern Recognition: Theory and Applications*. Spartan Books, New York (1970).

[106] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York, 1992.

[107] Michalski R.S., *A Variable-Valued Logic System as Applied to Picture Description and Recognition*. Graphic Languages, North-Holland (1972).

[108] Michalski R.S., *AQVAL/1- Computer implementation of a variable valued logic system VL1 and examples of its application to patter recognition*. Proceedings of the First International Joint Conference on Pattern Recognition, (1973), pp. 3-17.

[109] Michalski R.S., Larson J.B., *Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: The Underlying Methodology and Despcription of Programs ESEL and AQ11*. Report 867, University of Illinois, (1978).

[110] Michalski R.S., *Pattern Recognition as rule-guided Inductive Inference*. IEEE Transactions on Pattern Analysis and Machine Learning, Vol. PAMI-2, 4, (1980), pp. 249-361.

[111] Michalski R.S., *Learning by being told and learning from examples: an experimental comparison of the two methodes of knowledge acquisition in the context of developing an expert system for soybean desease diagnoiss*, International Journal of Policy Analysis and Information Systems, (1980), 4(2), pp. 125-161.

[112] Michalski R.S., *Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjuntiv Concepts.* Policy Analysis and Information Systems, Vol 4, 3, (1980), pp. 219-244.

[113] Michalski R.S., Stepp R., Diday E., *A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjuntive concepts.* Progress in Pattern Recognition, North-Holland, (1981), pp. 33-56.

[114] Michalski R.S., *Theory and Methodology of inductive learning.* Machine Learning: An artificial intelligence approach (Vol I). San Mateo, CA: Morgan Kaufmann (1983).

[115] Michalski R.S., *Understanding the nature of learning*, Machine Learning: An artificial intelligence approach (Vol II). San Mateo, CA: Morgan Kaufmann (1986).

[116] Mitchell T.M., *Version Spaces: A Candidate Elimination Approach to Rule Learning.* Proceedings of the Fifth International Joint Conference on Artificial Intelligence. (1977).

[117] Mitchell T.M., *Version Spaces: An approach to concept learning.* Ph.D. Dissertation, Stanford University (1978).

[118] Nilsson N.J., *Learning Machines.* McGraw-Hill, New York (1965).

[119] Nozaki K., Ishibuchi H., Tanaka H., *A simple but powerful heuristic method for generating fuzzy rules from numerical data.* Fuzzy Sets and Systems, 86 (1997) pp.251-270.

[120] Ourston D., Mooney R.J., *Theory refinement combining analytical and empirical methods.* Artificial Intelligence 66 pp. 273-309 (1994).

[121] Parodi, A., Bonelli, P., *A New Approach to Fuzzy Classifier System.* Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, (1993).

[122] Potter, M., De Jong K., *A cooperative coevolutionary approach to function optimization.* Proceedings of the Third Conference on Parallel Problem Solving from Nature. pp. 249-257. Springer-Verlag, (1994).

[123] Potter, M., De Jong K., Grefenstette *A coevolutionary approach to learning sequential decision rules.* Proceedings of the Sixth International (ICGA95). Morgan Kaufmann, (1995).

[124] Quinlan J.R., *Discovering rules from large collections of examples: a case study.* Expert Systems in the Micro Electronic Age. Edinburgh University Press, (1979).

[125] Quinlan J.R., *Induction of Decision Trees.* Machine Learning 1, (1986), pp. 81-106.

[126] Quinlan J.R., *Learning Decision Trees*, Technical Report 87.5, School of Computing Sciences, NSWIT, Sidney, Australia. (1987).

[127] Quinlan J.R., *Generating production rules from decision trees.* Proceedings of the Tenth International Joint Conference on Artificial Intelligence, (1987), pp. 304-307.

[128] Quinlan J.R., *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Francisco (1993).

[129] Radcliffe N.J., *Equivalence Class Analysis of Genetic Algorithms.* Complex Systems 5(2), (1991), pp. 183-205.

[130] Radcliffe N.J., *Non-Linear Genetic Representations.* Parallel Problem Solving from Nature 2, R. Männer, B. Manderick (Eds.) (Elsevier Science Publishers, Amsterdam, 1992), pp. 259-268.

[131] Rashevsky N., *Mathematical Biophysics.* University of Chicago Press, Chicago, IL, (1948).

[132] Rechemberg I., *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

[133] Rissanen J., *Stochastic complexity and modeling.* Annals of Statistical, 14 (1986), pp. 1080-1100.

[134] Rosenblatt F., *El Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review, Vol. 65, (1958) pp. 386-407.

[135] Samuel A.L., *Some Studies in Machine Learning Using the Game of Checkers.* IBM Journal of Research and Development, No. 3, (1959), pp. 211-229.

[136] Samuel A.L., *Some Studies in Machine Learning Using the Game of Checkers.* Computer and Thought, McGraw-Hill, New York, (1963), pp. 71-105.

[137] Schwefel H-P., *Numerical Optimization of Computer Models*, Wiley, Chichester, (1981).

[138] Schwefel H-P., *Evolution and Optimum Seeking. Sixth-Generation Computer Technology Series*, Wiley, New York, (1995).

[139] Selfridge O.G., *Pandemonium: A paradigm for Learning.* Proceedings of the Symposium on Mechanization of Thought Processes, Blake, D. and Uttley, A. (Eds), HMSO, London, (1959), pp. 511-529.

[140] Sestito S., Dillon T.S., *Automated Knowledge Adqusition.* Prentice Hall. (1994).

[141] Sigillito V.G., Wing S.P., Hutton L.V., Baker K.B., *Classification of Radar Returns from the Ionosphere using Neural Networks.* Johns Hopkins APL Technical Digest 10, (1989), pp. 262-266.

[142] Simon h.A., Kotovsky K., *Human acqusition of concepts for sequential patterns.* Psychological Review, Vol. 70, (1963), pp. 534-546.

[143] Simon H.A., Lea G., *Problem Solving and Rule Induction: A Unified View*, Knowledge and Cognition. Lawrence Erlbaum Associates, Potomac, Maryland (1974), pp. 105-127.

[144] Simon H.A., *Why Should Machine Learn?.* Machine Learning: An Artificial Intelligence Approach. Springer-Verlag (1983) pp. 25-38.

[145] Smith S.F., *A Learning System based on Genetic Algorithms.* Doctoral Dissertation, University of Pittsburgh, Departamente of Computer Science, (1980).

[146] Soloway E.M., Riseman E.M., *Levels of pattern description in learning.* Fifth International Joint Conference on Artificial Intelligence. (1977), pp. 801-811.

[147] Stepp R., *The investigation of the UNICLASS inductive program AQ7UNI and User's Guide.* Technical Report 949, Departament of Computer Science, University of Illinois (1978).

[148] Thrun S.B., Bala J., Bloedorn E., Bratko I., Cestnik B., Cheng J., De Jong K., Dzeroski S., Fahlman S.E., Fisher D., Hamann R., Kaufman K., Keller Sl, Reich Y., Vafaie H., Van de Welde W., Wenzel W., Wnek J., Zhang J., *The MONK's Problems: A Performance Comparison of Different Learning Algorithms.* CMU-CS-91-197, Carnegie Mellon University. (1991).

[149] Truxal T.G., *Automatic Feedback Control System Synthesis.* McGraw-Hill, New York (1955).

[150] Typskin Y.Z., *Self Learning - What is it?.* IEEE Transactions on Automatic Control, Vol. AC-18, No. 2, (1968), pp. 109-117.

[151] Typskin Y.Z., *Adaptation and Learning in Automatic Systems.* Academic Press, New York, (1971)

[152] Typskin Y.Z., *Foundations of the Theory of Learning Systems.* Academic Press, New York, (1973)

[153] Uhr L., *Pattern Recognition.* John Wiley and Sons, New York, (1966).

[154] Uthurusamy R., Fayyad U.M., Spangler S., *Learning useful rules from inconclusive data.* Knowledge Discovery in Databases, (1990).

[155] Valenzuela-Redón, M., *The Fuzzy Classifier System: Motivations and First Results.* Parallel Problem Solving from Nature II, Springer Verlag (1991), pp. 330-334.

[156] Velasco J.R., Magdalena L., *Genetic algorithms in Fuzzy Control Systems*, en: J. Periaux y G. Winter, Genetic Algorithms in Engineering and Computer Science, (1995).

[157] Venturini G., *SIA: a Supervised Inductive Algorithm with Genetic Search for Learning Attributes based Concepts.* Machine Learning: ECML-93, (1993) pp. 280-296.

[158] Vere S.A., *Induction of Concepts in the Predicate Calculus.* Proceedings of the Fourth International Joint Conference on Artificial Intelligence, IJCAI, (1975) pp. 281-287.

[159] Wang, L., Mendel, M., *Generating fuzzy rules by learning from examples.* IEEE Transactions on systems, man and cybernetics, vol.22 n.6, pp.1414-1427 (1992)

[160] Weber R., *Fuzzy-ID3: A Class of Methods for Automatic Knowledge Acquisition* Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Network, (1992), pp. 265-268.

[161] Wehenkel, L., *On Uncertainty measures Used for Decision Tree Induction*, Proc. of IPMU'96, (1996), pp. 413-418.

[162] WeiB G., *Learning the Goal Relevance of Actions in Classifier Systems.* Proceedings of the Tenth European Conference on Artificial Intelligence, (1992), pp. 430-434.

[163] Widrow B., *Generalization and Information Storage in Networks of Adelaine (Neurons)*, Spartan Books, Washington D.C., (1962) pp. 435-461.

[164] Wilson, S.W., *Classifier Systems and the Animat Problem*, Machine Learning 2, (1987), pp. 199-228.

[165] Wilson, S.W., Goldberg D.E., *A Critical Review of Classifier Systems.* Proceedings of the Third International Conference on Genetic Algorithms, (1989), pp. 244-255.

[166] Wilson, S.W., *ZCS: A Zeroth Level Classifier System.* Evolutionary Computation Vol. 2, No. 1 (1994) pp. 1-18.

[167] Winston P.H., *Learning Structural Descriptions from Examples.* Ph. D. dissertation, MIT, (970).

[168] Winston P.H., *Learning structural descriptions from examples.* The Psychology of Computer Vision. McGraw-Hill, New York, ch. 5 (1975).

[169] Winston P.H., *Artificial Intelligence*, Addison-Wesley, (1977).

[170] Wright A., *Genetic Algorithms for Real Parameter Optimization.* Foundations of Genetic Algorithms 1, G.J.E. Rawlin(Ed.) (Morgan Kaufmann, San Mateo, 1991), pp. 205-218.

[171] Zadeh L.A., *Fuzzy Set.* Information and Control 8, pp. 338-353 (1965).

[172] Zadeh L.A., *The concept of a linguistic variable and its applications to approximate reasoning.* Part I Information Sciences vol.8, pp. 199-249, Part II Information Sciences vol. 8, pp. 301-357, Part III Information Sciences vol. 9, pp. 43-80. (1975).