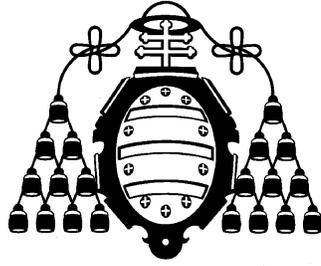


UNIVERSIDAD DE OVIEDO



Departamento de Informática

Un sistema de aprendizaje de reglas explícitas
mediante la generalización de instancias

Tesis Doctoral

Autor: Oscar Luaces Rodríguez
Director: Antonio Bahamonde Rionda

Septiembre, 1999

Quiero dedicar el esfuerzo realizado en la consecución de este trabajo a Rocío y Raquel, por los ratos que no hemos pasado juntos.

Agradecimientos

Quiero expresar mi agradecimiento a Pedro Domingos por su deferencia al facilitarme amablemente el código fuente de su sistema RISE, para ser incorporado a la librería MLC++ utilizada para realizar los experimentos. En este sentido, también es de agradecer que Ross Quinlan haya hecho de dominio público su sistema C4.5.

Índice General

1	Presentación	1
1.1	Introducción	1
1.2	¿Dónde se sitúa INNER?	2
1.3	Aportaciones principales	7
1.4	Estructura de la memoria	10
2	Una visión general del sistema	11
2.1	El algoritmo principal del sistema	11
2.2	La selección inicial de instancias	13
2.3	El cálculo de las distancias	15
3	Generalización de instancias	23
3.1	Introducción	23
3.2	\mathcal{K} -HEOM: Una métrica auto-organizada	24
3.3	El proceso de generalización	26
3.3.1	Modificación de antecedentes continuos	26
3.3.2	Modificación de antecedentes simbólicos	28
3.4	Regularización	32
3.4.1	La calidad de una regla	35
3.5	Cualificación	40
4	Post-proceso de reglas	47
4.1	Introducción	47
4.2	Extensión de reglas	48
4.2.1	\mathcal{R} -HEOM: La distancia entre reglas	49
4.2.2	Los antecedentes extensibles	53
4.3	Selección de reglas	58
4.4	Extensión con solapamientos	65
4.5	Inflado final de reglas	69

5	Resultados experimentales	73
5.1	Introducción	73
5.2	Problemas usados en las pruebas	73
5.3	Resultados	82
5.3.1	\mathcal{K} -INNER: Una variante menos explícita de INNER	86
5.3.2	La influencia del parámetro de cobertura	86
5.3.3	La influencia de la selección inicial de instancias	87
6	Conclusiones	91

Índice de Figuras

2.1	Proceso de generalización.	12
2.2	Número máximo de reglas iniciales por clase.	14
2.3	La cruz alargada	20
3.1	Generalización de instancias.	23
3.2	Sigmoide para atributos continuos.	27
3.3	Modificación de antecedentes continuos	29
3.4	Sigmoide para atributos simbólicos.	30
3.5	Función para acotar distancias en el terreno simbólico.	31
3.6	Crecimiento excesivo de una regla.	34
3.7	Grado de pertenencia de un símbolo a un antecedente.	37
3.8	Optimización en el cálculo del nivel de impureza.	39
3.9	Cualificación de una regla	40
3.10	Fases de la cualificación de una regla.	45
4.1	Función de pertenencia de un símbolo a un antecedente.	52
4.2	Extensiones aconsejables y no aconsejables.	55
4.3	Selección de antecedentes a extender.	56
4.4	Buscando el mejor corte por nivel de impureza.	62
4.5	Reglas eliminables por inclusión de regla por defecto.	64
4.6	Asignación explícita de prioridad	65
4.7	Situación de “ <i>caja de cartón</i> ”.	68
4.8	Extensiones con solapamiento.	70
4.9	Inflado final con prioridad explícita en los solapamientos.	71

Índice de Tablas

1.1	Esquema general del conjunto de entrenamiento.	2
5.1	Banco de pruebas.	81
5.2	Comparativa de precisión de INNER.	84
5.3	Tamaño de la solución obtenida.	85
5.4	Comparativa de precisión de \mathcal{K} -INNER.	88
5.5	Diferentes umbrales de cobertura.	89
5.6	La selección especializada de instancias iniciales por BETS.	90

Índice de Algoritmos

1.2.1 Estructura típica de los algoritmos de tipo separa-y-vencerás	8
2.1.1 El algoritmo de aprendizaje en INNER.	11
2.2.1 La obtención de un conjunto de reglas.	13
3.1.1 Generalización de instancias/reglas.	24
3.4.1 Regularización de reglas.	33
3.4.2 Regularización de antecedentes continuos.	33
3.4.3 Regularización de antecedentes simbólicos.	36
3.4.4 El nivel de impureza de una regla.	39
3.5.1 Cualificación de reglas.	41
3.5.2 Generación de reglas candidatas.	43
3.5.3 Eliminación de antecedentes redundantes.	44
3.5.4 Filtrado de reglas candidatas.	44
4.1.1 Post-proceso de reglas.	47
4.2.1 Extensión de reglas.	50
4.2.2 Construcción de la lista de reglas para extender.	51
4.2.3 Extensión de una regla sobre otra.	54
4.2.4 Antecedentes susceptibles de ser extendidos.	57
4.3.1 Selección de reglas.	58
4.3.2 Eliminación de reglas ruidosas.	60
4.3.3 Búsqueda del nivel de impureza con mejor rendimiento.	61
4.3.4 Eliminación de reglas redundantes.	62
4.3.5 Inclusión de regla por defecto.	64
4.4.1 Asignación de prioridades en las intersecciones.	67

Capítulo 1

Presentación

1.1 Introducción

El aprendizaje automático puede entenderse de forma general como el proceso mediante el cual se consigue

synetizar procedimientos capaces de realizar tareas de las que sólo tenemos algunas descripciones parciales sobre el modo en que nos gustaría que se llevasen a cabo. [Ranilla, 98]

En esta disertación se presenta el sistema INNER, que aprende a realizar una determinada tarea de clasificación, extrayendo para ello el conocimiento necesario mediante la observación de instancias o ejemplos representativos de las distintas clases que se pretenden distinguir. El conocimiento extraído se va a representar mediante un conjunto de reglas de clasificación muy compacto y legible para el ser humano.

Es decir, nuestro sistema producirá un resumen explícito de los ejemplos de entrenamiento; su lectura nos permitirá observar con todo detalle las razones que tendrán las reglas sintetizadas por INNER para predecir las clases o categorías de los casos no vistos.

Los ejemplos de entrenamiento van a venir descritos por un conjunto de pares atributo/valor, con un atributo especial que va a ser precisamente la clase a la que pertenece ese caso. En la Tabla 1.1 puede verse una representación esquemática de lo que podría ser un conjunto de entrenamiento.

Estos ejemplos pueden considerarse pares formados por preguntas (los valores de los atributos salvo la clase) y la respuesta (la clase) dada por un maestro. El algoritmo que se va a describir tendrá por misión clonar el comportamiento del maestro para construir un asistente inteligente y amigable en cuyas clasificaciones podamos confiar.

Ejemplos	Atributos			Clase
	Atributo ₁	...	Atributo _n	
1)	valor _{1,1}	...	valor _{1,n}	clase ₁
2)	valor _{2,1}	...	valor _{2,n}	clase ₂
⋮	⋮		⋮	⋮
m)	valor _{m,1}	...	valor _{m,n}	clase _m

Tabla 1.1: Esquema general del conjunto de entrenamiento.

Esta confianza se constata en:

- la gran precisión que los clasificadores sintetizados por INNER son capaces de alcanzar como se probó con los experimentos realizados sobre una colección de problemas ampliamente reconocidos como representativos por la comunidad científica.
- el carácter explícito y compacto del conocimiento sintetizado que, por una parte, permitiría su modificación previa a las labores de clasificación. Pero, además, brinda la ocasión de acompañar las clasificaciones con una explicación perfectamente comprensible para alguien relacionado con el problema manejado.

1.2 ¿Donde se sitúa INNER?

En esta sección se va a tratar de situar el sistema que se presenta en el lugar que le corresponde dentro de las diferentes categorías de sistemas de aprendizaje automático existentes. Estos sistemas se pueden clasificar atendiendo a muchos aspectos, algunos de los cuales son:

- en función del resultado que calculan
 - clases discretas: los sistemas aprenden a clasificar las entradas en un número fijo de clases.
 - valores continuos: se podría decir que el conocimiento que extraen estos sistemas durante el proceso de aprendizaje es la función que relaciona las entradas con las salidas en el dominio correspondiente.
- por la legibilidad del conocimiento aprendido
 - simbólicos: la representación del conocimiento es humanamente inteligible; esto puede, en cierta manera, *enseñar* a resolver el problema que el sistema ha aprendido.

- subsimbólicos: aunque computacionalmente efectivo, el conocimiento que extraen no puede ser ofrecido a un humano por su ininteligibilidad.
- por los principios que guían su diseño [Quinlan, 93b]:
 - estadísticos: suelen hacer conjeturas sobre las distribuciones de los valores de los atributos.
 - basados en redes de neuronas artificiales: estos sistemas tratan de simular, de forma simplificada, el funcionamiento de redes de neuronas biológicas. Lo que aprenden estos sistemas no es humanamente inteligible aunque sí es computacionalmente efectivo.
 - basados en el recuerdo de instancias: los sistemas basados en instancias o IBL¹ memorizan prototipos de casos que se le suministran como entrenamiento para equipararlos con nuevos casos y asignarles su clase en función del parecido entre ellos. Más adelante volveremos sobre estos sistemas debido a la relación que guardan con el sistema que aquí se presenta.
 - Generalizaciones simbólicas: representan el conocimiento que aprenden de forma inteligible para los humanos. Tratan de generalizar la información contenida en los casos que se utilizan durante el entrenamiento. Los sistemas de este tipo pueden a su vez clasificarse según la representación simbólica que utilizan:
 - * árboles de decisión: estos sistemas se basan en el principio denominado “*divide-y-vencerás*”, construyendo un árbol que en cada nodo establece unas condiciones sobre un atributo, dividiendo así el conjunto de casos en subconjuntos que cumplen cada condición. De nuevo los subconjuntos se vuelven a dividir añadiendo así nuevos niveles al árbol de decisión, hasta alcanzar un determinado criterio de parada. El ejemplo por excelencia de este tipo de sistemas es el C4.5 de Ross Quinlan [Quinlan, 93a].
 - * reglas de clasificación: las reglas suelen ser cláusulas de lógica proposicional donde las premisas son condiciones sobre los atributos de los ejemplos y el consecuente es una etiqueta con la clase que se asignará en caso de que se aplique la regla. Muchos de estos sistemas se basan en un principio denominado “*separa-y-vencerás*” que se detallará más adelante.
- por el mecanismo de aplicación del conocimiento sintetizado
 - ajuste exacto: para emitir una determinada salida han de cumplirse unas condiciones. En términos de cláusulas lógicas la entrada será clasificada por algún clasificador que ve satisfechas sus premisas.

¹*Instance-Based Learning* en la literatura anglosajona.

- por distancia mínima: en estos sistemas la salida la produce el clasificador cuyas premisas se asemejan más a los valores de entrada.
- por la geometría de las regiones de decisión: los clasificadores se utilizan cuando la entrada “cae” dentro de su área de influencia; esta puede venir definida por
 - lados paralelos a los ejes: las regiones de decisión son hiperrectángulos en el espacio de atributos.
 - lados oblicuos: las regiones son, generalmente, polígonos irregulares multi-dimensionales (hiperpolígonos irregulares)
 - áreas de geometría irregular: regiones en el espacio de atributos delimitadas por contornos irregulares. Los sistemas que aplican el conocimiento aprendido por distancia mínima, como los sistemas IBL o el sistema objeto de esta disertación, dan lugar a estas regiones.

Atendiendo a los principios que guían su diseño, el sistema que aquí se presenta ha de enmarcarse en algún punto intermedio entre los sistemas de aprendizaje basados en instancias o IBL y los sistemas de inducción de reglas del tipo separa-y-vencerás

Los sistemas IBL se basan en la hipótesis de que en los alrededores de un ejemplo de una determinada clase habrá, probablemente, más ejemplos de esa misma clase. Para ello, lo que hacen es utilizar los propios ejemplos o instancias como clasificadores. En el caso más sencillo, estos sistemas almacenan todos los ejemplos del conjunto de entrenamiento y, cuando se presenta un nuevo ejemplo para ser clasificado, se le atribuye la misma clase que la de la instancia almacenada más *cercana*. Este mecanismo de clasificación implica usar una determinada medida de similitud o distancia entre ejemplos, que juega un papel fundamental en la eficacia de clasificación de estos sistemas; para este fin se han propuesto diferentes métricas [Fürnkranz, 97], [Wilson, Martínez, 97].

Los sistemas basados en instancias pueden presentar algunos inconvenientes, como son la sensibilidad al ruido en el conjunto de entrenamiento y a la presencia de atributos no relevantes. Los conjuntos con ruido provocan que las instancias incorrectas clasifiquen de forma errónea todos aquellos casos que caigan dentro de su área de influencia; es decir, todos aquellos casos que estén más cerca del ejemplo erróneo que de otros ejemplos. Por otra parte, los atributos irrelevantes pueden influir negativamente en el cálculo de las distancias, eje central de los sistemas IBL. Existen diferentes métodos para paliar estos efectos negativos [Aha, 90], [Del Coz *et al.*, 99], [Wilson, Martínez, 99].

Sin embargo, hay que destacar que las fronteras efectivas entre clases que definen estos sistemas son generalmente complejas [Domingos, 96] debido precisamente a que la clasificación se hace por distancia. Las regiones de influencia que se forman alrededor de cada instancia dependen de la presencia de otras instancias y el sistema puede clasificar correctamente ejemplos en problemas en los que las clases se distribuyen de forma poco regular en el espacio de atributos.

Por otra parte, los algoritmos de inducción de reglas que utilizan la técnica de separa-y-vencerás deben su nombre a que construyen una regla que cubre tantos ejemplos positivos como sea posible y pocos o ningún ejemplo negativo². A continuación se retiran los ejemplos cubiertos por la regla obtenida y se repite el proceso con objeto de obtener nuevas reglas que expliquen los ejemplos que quedan por cubrir (por “conquistar”). El Algoritmo 1.2.1 reproduce, en versión traducida de [Fürnkranz, 97], el esquema típico de un algoritmo de este tipo. Falta precisar únicamente que las reglas que manejan estos sistemas están compuestas por un *consecuente*, que será la clase que predice dicha regla, y unos *antecedentes*, que serán condiciones sobre los atributos del problema. Se dice que una regla *cubre* un ejemplo cuando los valores de los atributos de éste satisfacen todos y cada uno de los antecedentes de la regla

$$C \leftarrow Atr_1 \in [x_1, x_2] \wedge Atr_2 = Valor_2 \wedge \dots \wedge Atr_n = Valor_m \quad (1.1)$$

En (1.1) puede verse el aspecto típico de una regla de clasificación de estos sistemas. En este caso, el primer antecedente establece una condición sobre un atributo que toma valores continuos, concretamente la condición de pertenecer a un determinado intervalo. Sin embargo, cuando se trata de atributos simbólicos, la condición suele ser, simplemente, que el atributo tome un valor concreto.

Estos algoritmos parecen ser más adecuados para resolver problemas cuya solución puede ser expresada en un lenguaje lógico del que (1.1) es un caso particular. Sin embargo pueden tener dificultades para resolver problemas con atributos con valores numéricos en los que las clases vienen delimitadas por fronteras que no son paralelas a los ejes. Además, a medida que el proceso de inducción avanza, el conjunto de entrenamiento es cada vez más pequeño, y las últimas reglas se obtienen, en muchos casos, a partir de un número de ejemplos poco representativo: esto se conoce como el problema de la *fragmentación* [Domingos, 96]. El resultado es que se inducen reglas que cubren muy pocos ejemplos y que suelen ser más propensas a cometer errores; sin embargo, eliminar esas reglas generalmente conduce al sistema a tener menos precisión: es el problema de las *pequeñas disyunciones* [Holte, Acker, Porter, 89] (reglas pequeñas que cubren pocos ejemplos).

Se puede decir, que el sistema INNER pretende aprovechar las ventajas aportadas por ambos enfoques, tratando de conseguir no sólo precisión en la clasificación, sino también una solución lo más *explícita* posible, en forma de reglas. Para ello, en aplicación del principio de economía conocido como *la navaja de Ockham*, es deseable obtener pocas reglas, que éstas sean simples, con pocos antecedentes y que cubran el espacio de atributos de forma que se puedan distinguir con claridad las regiones que se caracterizan por pertenecer a una determinada clase.

²Para el caso de problemas con más de dos clases se puede extender esta idea fácilmente, considerando como ejemplos negativos para una regla de clase C todos los ejemplos que no sean de esta clase.

Las reglas que induce INNER tendrán una sintaxis como la que se muestra a continuación:

$$C \leftarrow Atr_1 \in [x_1, x_2] \wedge Atr_2 \in \{Valor_{2,1}, Valor_{2,2}, \dots\} \wedge \dots \quad (1.2)$$

donde puede verse que los antecedentes de tipo continuo, como Atr_1 , establecen una condición de pertenencia a un intervalo cerrado de valores, mientras que los antecedentes de tipo simbólico establecen una condición de pertenencia a un conjunto de valores posibles, unidos por disyunciones implícitas; así, la regla expresada en (1.2) indica que:

Un ejemplo es de clase C si

el valor x del atributo Atr_1 cumple que $x_1 \leq x \leq x_2$ y

el valor y del atributo Atr_2 cumple ($y = Valor_{2,1} \vee y = Valor_{2,2} \vee \dots$) y

... así sucesivamente con el resto de antecedentes.

Para la obtención de reglas INNER sigue un procedimiento similar al descrito en [Fürnkranz, 97] como prototípico de los algoritmos de tipo separa-y-vencerás y que se reproduce en el Algoritmo 1.2.1. Sin embargo, aunque el esquema general de INNER encaja bastante bien con ese algoritmo, hay diferencias destacables que impiden clasificarlo como algoritmo de tipo separa-y-vencerás. Estas diferencias son:

- En cada iteración no se obtiene una sola regla, sino que se van construyendo varias reglas a un tiempo.
- Entre los distintos ciclos de obtención de reglas no se separan los ejemplos cubiertos del resto, sino que el conjunto de ejemplos de entrenamiento permanece invariable, con lo que se evita el problema de la fragmentación que presentan los algoritmos de tipo separa-y-vencerás.
- Las fronteras que definen las regiones de decisión o área de influencia de las reglas no son paralelas a los ejes del problema, puesto que las reglas se aplican por distancia tal y como se hace en los sistemas IBL. Esto permite en muchos casos obtener mayor precisión en la clasificación.

En [Fürnkranz, 97] se menciona que cualquier algoritmo de aprendizaje puede caracterizarse por la predilección que tiene a la hora de seleccionar una generalización sobre otra, aparte de la consistencia con los ejemplos de entrenamiento; esa predilección se denomina *bias* en la literatura anglosajona y permite clasificar los algoritmos de aprendizaje en función de:

- *La predilección por la representación del conocimiento* (Language Bias), que determina en gran medida el espacio de búsqueda del problema, puesto que hay conceptos que pueden no ser (fácilmente) expresables utilizando determinadas representaciones. En INNER la representación del conocimiento extraído se hace mediante reglas como la expresada en (1.2).
- *La predilección en la búsqueda* (Search Bias), que hace referencia al algoritmo de búsqueda utilizado (escalada, primero el mejor, etc.), a la estrategia de búsqueda empleada (de más específico a más general o viceversa) y a los heurísticos incluidos en el algoritmo de búsqueda. INNER hace uso de un algoritmo de escalada informado por un heurístico denominado *nivel de impureza* [Ranilla, Mones, Bahamonde, 98], [Ranilla, 98] que se presentará en la sección 3.4.1.
- *La técnica para evitar el sobreajuste* (Overfitting Avoidance Bias), que está relacionado con el tratamiento de problemas con ruido, en los que lo mejor no es acertar en la clasificación de todos los casos presentes en el conjunto de entrenamiento³. El nivel de impureza antes mencionado está diseñado para abordar adecuadamente este problema.

Todo ello hace que, de hecho, INNER tenga un parentesco más fuerte con el sistema RISE [Domingos, 96], que también “*unifica*”, los procesos de inducción basados en reglas y los basados en instancias. Ambos sistemas, siguiendo una estrategia de aprendizaje de abajo hacia arriba, parten de reglas puntuales que cubren un sólo ejemplo y van generalizándolas de forma progresiva. No obstante las diferencias más destacables son:

- INNER incluye procesos específicos de poda de reglas, lo que le va a permitir obtener conjuntos más reducidos y reglas más sencillas, con menos antecedentes.
- El aprendizaje que efectúa INNER es *incremental* lo que indica, entre otras cosas, que el orden de presentación de ejemplos puede afectar a los resultados que se obtienen.
- La medida de las distancias entre reglas y ejemplos se efectúa de forma similar, pero INNER incorpora un mecanismo original para los antecedentes/atributos simbólicos, que *aprende* a medir distancias al tiempo que el sistema aprende a resolver el problema que se le plantea.

1.3 Aportaciones principales

La solución a un problema de aprendizaje consiste en el caso de INNER, en extraer de los ejemplos de entrenamiento un conjunto de reglas del tipo de (1.2) con la mejor

³De hecho, se debería fallar en los ejemplos que constituyen el ruido del problema.

Algoritmo 1.2.1 Estructura típica de los algoritmos de tipo separa-y-vencerás

Función SEPARA-Y-VENCERÁS(*Ejemplos*) : Cjto. Reglas
Teoría = \emptyset
mientras POSITIVOS(*Ejemplos*) $\neq \emptyset$ **hacer**
 Regla = ENCONTRARMEJORREGLA(*Ejemplos*)
 Cubiertos = COBERTURA(*Regla*, *Ejemplos*)
 si CRITERIOPARADAREGLA(*Teoría*, *Regla*, *Ejemplos*) **entonces**
 salir mientras
 fin si
 Ejemplos = *Ejemplos* \ *Cubiertos*
 Teoría = *Teoría* \cup *Regla*
fin mientras
Teoría = POSTPROCESAR(*Teoría*)
retornar(*Teoría*)

Función ENCONTRARMEJORREGLA(*Ejemplos*) : Cjto. Reglas
ReglaInicial = INICIALIZARREGLA(*Ejemplos*)
ValorInicial = EVALUARREGLA(*ReglaInicial*)
MejorRegla = \langle *ValorInicial*, *ReglaInicial* \rangle
Reglas = {*MejorRegla*}
mientras *Reglas* $\neq \emptyset$ **hacer**
 Candidatas = SELECCIONARCANDIDATAS(*Reglas*, *Ejemplos*)
 Reglas = *Reglas* \ *Candidatas*
 para *Candidata* \in *Candidatas* **hacer**
 Refinamientos = REFINARREGLA(*Candidata*, *Ejemplos*)
 para *Refinamiento* \in *Refinamientos* **hacer**
 Valor = EVALUARREGLA(*Refinamiento*, *Ejemplos*)
 si NOCRITERIOPARADA(*Refinamiento*, *Ejemplos*) **entonces**
 NuevaRegla = \langle *Valor*, *Refinamiento* \rangle
 Reglas = INSERTARORDENADA(*NuevaRegla*, *Reglas*)
 si *NuevaRegla* $>$ *MejorRegla* **entonces**
 MejorRegla = *NuevaRegla*
 fin si
 fin para
 fin para
 Reglas = FILTRARREGLAS(*Reglas*, *Ejemplos*)
fin mientras
retornar(*MejorRegla*)

calidad posible. La calidad de las soluciones obtenidas por un sistema de aprendizaje automático, en general, puede medirse de dos formas:

- *Cuantitativamente*: un sistema será mejor cuanto menor error cometa en su tarea de clasificación.
- *Cualitativamente*: aunque es difícil establecer cuándo un sistema da mejores soluciones que otro en el plano cualitativo, parece razonable evaluar la calidad en función de:
 - *Claridad conceptual*, que tiene una relación inversa con el número de elementos clasificadores (reglas, hojas en los árboles de decisión, etc.) ya que, a menor número de éstos, mayor claridad. En aras a esta claridad es deseable también que los clasificadores sean sencillos (reglas con pocos antecedentes, por ejemplo). Es obvio que en sistemas que representan los conceptos de forma ininteligible, la claridad conceptual es nula.
 - *Capacidad de explicación* que permita justificar de una forma similar a como lo haría un ser humano las decisiones tomadas.

Habitualmente será imposible saber con exactitud cual es el error cometido en todos los posibles ejemplos de un dominio, por lo que esta medida ha de estimarse mediante pruebas experimentales con un soporte estadístico adecuado. En este trabajo de investigación se ha utilizado la prueba de la *validación cruzada estratificada* sobre un conjunto de problemas estándar en el aprendizaje automático. Este método consiste en dividir el conjunto de ejemplos en k partes aproximadamente iguales y que mantengan la distribución de clases del conjunto original. A continuación se procede a evaluar sobre cada partición el conocimiento aprendido de las otras $k - 1$ particiones; este procedimiento se suele repetir n veces, obteniéndose así $k \cdot n$ resultados. El estimador usado será la media de todos esos resultados.

En lo que respecta a la calidad de las soluciones, la filosofía seguida para la obtención de las reglas hace que éstas *cubran explícitamente* regiones del espacio de atributos donde hay una acumulación importante de ejemplos de una determinada clase. En general y a primera vista, puede parecer que la capacidad de explicación de un sistema es mayor cuando las reglas se aplican porque cubren los ejemplos que cuando se aplican por distancia. La razón sería que, a la vista de un ejemplo y de la expresión sintáctica de la regla que lo clasifica, el usuario puede ver que se cumplen las premisas que hacen que esa regla sea aplicable, mientras que si la regla se aplica por distancia el usuario sólo sabría que el ejemplo se parece más a los ejemplos de esa clase que a los de otra.

Sin embargo, la obsesión de algunos sistemas por cubrir todo el espacio de atributos puede hacer que algunas reglas resulten un tanto antinaturales. Parece, pues, deseable un buen equilibrio entre sistemas con reglas que cubran todo el espacio de atributos o sólo una pequeña porción. Así, determinadas regiones con una alta densidad de

ejemplos de entrenamiento deben aparecer como reglas de clasificación, mientras que puede permitirse que algunos ejemplos aislados sean clasificados por reglas próximas que proporcionarán unas explicaciones más válidas.

1.4 Estructura de la memoria

En el resto de esta memoria se van a describir con detalle todas y cada una de las partes de que consta INNER. En primer lugar, en el Capítulo 2 se dará una visión general del sistema, descomponiéndolo en las partes más relevantes que serán detalladas en los capítulos posteriores.

En el Capítulo 3 se presenta con detalle el proceso central que caracteriza al algoritmo INNER: la *generalización a partir de instancias* que se toman inicialmente como reglas puntuales y se convierten en clasificadores más generales. En este capítulo se va a describir también el heurístico utilizado para evaluar la calidad de una regla, el *nivel de impureza* antes citado. Finalmente se presenta en este capítulo un mecanismo de poda de antecedentes que ha sido tomado de otro sistema, ABANICO [Ranilla, 98], y que se denomina *cualificación*.

El Capítulo 4 muestra como, tras haber obtenido un conjunto de reglas fruto de la generalización de un número más o menos grande de instancias iniciales, se efectúa un post-proceso que tiene por objeto tratar de reducir el número de reglas sin perder precisión. Una parte de este post-proceso, la *selección* es también un proceso heredado de ABANICO e incluido en INNER con unas ligeras modificaciones.

La comparativa del sistema INNER con sistemas de renombre en el ámbito del aprendizaje automático se muestra en el Capítulo 5; aquí se pueden ver los resultados obtenidos frente a los de otros sistemas de probada calidad contra una batería de problemas que goza ya de una cierta fama en el ámbito del aprendizaje automático.

El Capítulo 6 recoge las conclusiones más destacables que se pueden extraer del trabajo de investigación que ha dado lugar al sistema INNER y a esta memoria.

Capítulo 2

Una visión general del sistema

2.1 El algoritmo principal del sistema

En el Algoritmo 2.1.1 se presenta el mecanismo general de aprendizaje de reglas incorporado en el sistema INNER. En este algoritmo se parte de un conjunto vacío de reglas denominado *Teoría* al que se van añadiendo de forma iterativa los conjuntos de reglas que se obtienen en cada ciclo mediante un proceso de generalización aplicado a algunas de las instancias del conjunto de entrenamiento. A la unión de todos esos conjuntos, la *Teoría*, se le aplicará finalmente un proceso de agregación y poda, con objeto de prescindir de reglas innecesarias desde un punto de vista global.

Algoritmo 2.1.1 El algoritmo de aprendizaje en INNER.

```
Función INNER(Ejemplos) : Cjto. Reglas
  Teoría =  $\emptyset$ 
  NoCubiertos = Ejemplos
  mientras NO(CRITERIOPARADA(NoCubiertos)) hacer
    Reglas = ENCONTRARMEJORESREGLAS(Ejemplos, NoCubiertos)
    Teoría = Teoría  $\cup$  Reglas
    Cubiertos = COBERTURA(Teoría, Ejemplos)
    NoCubiertos = Ejemplos \ Cubiertos
  fin mientras
  Teoría = POSTPROCESAR(Teoría, Ejemplos)
  retornar(Teoría)
```

En cada ciclo del algoritmo principal se seleccionan unas instancias que pasan a considerarse como reglas de clasificación *elementales* o *puntuales*, como sucede en los algoritmos de aprendizaje basados en instancias que se mencionaron en el Capítulo 1.

A continuación esas reglas de aprendizaje sufren un proceso de generalización similar a un *inflado*, ya que crecen en la medida de lo posible, cubriendo el vecindario de la instancia de partida, como muestra la Figura 2.1. De hecho, INNER es un acrónimo de **INflando Ejemplos para obtener Reglas**.

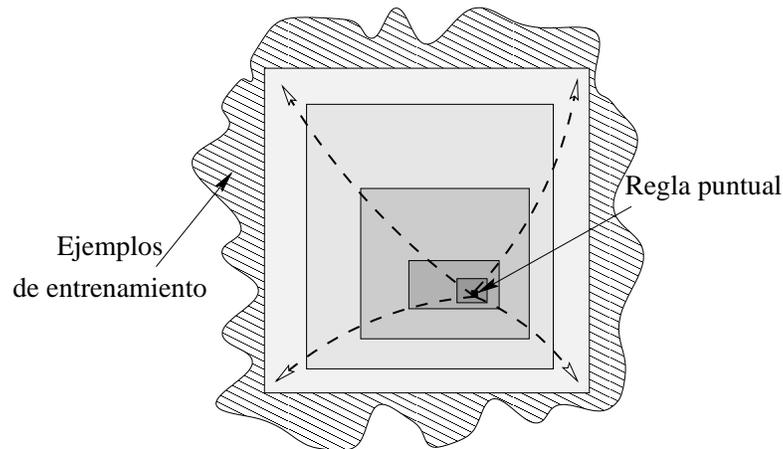


Figura 2.1: El proceso de generalización consiste en *inflar* las reglas todo lo que se pueda, manteniendo o incrementando la calidad de las clasificaciones obtenidas.

El criterio de parada en la obtención de los distintos subconjuntos de reglas, implementado en CRITERIOPARADA, tiene que ver con el porcentaje de ejemplos cubiertos por las reglas obtenidas hasta el momento, ya que este sistema pretende sintetizar reglas que cubran explícitamente las regiones que van a clasificar. Por defecto el sistema repite los ciclos de obtención de reglas hasta que el 95% de ejemplos de cada clase está cubierto o hasta que se alcanza un número máximo de cinco ciclos.

Nótese que la cobertura se calcula teniendo en cuenta todas las reglas obtenidas hasta el momento y no sólo las correspondientes a la iteración actual. Sin embargo las reglas de ciclos anteriores no interfieren en la obtención de las del ciclo actual; las reglas obtenidas en cada iteración se añaden al conjunto *Teoría* y se retiran de forma que en el ciclo siguiente se vuelve a empezar desde el principio. La filosofía es similar a la de los algoritmos separa-y-vencerás pero en cada ciclo no se retiran los ejemplos cubiertos, sólo las reglas que los cubren. La idea subyacente es que, si en un ciclo las reglas puntuales no han crecido lo suficiente para cubrir el porcentaje deseado de ejemplos seguramente se debe a que las instancias seleccionadas como reglas de partida se tomaron de regiones poco adecuadas, así que vamos a seleccionar las reglas puntuales iniciales de otras zonas pero retirando del espacio del problema a las anteriores para que no estorben en el crecimiento en este nuevo ciclo.

Algoritmo 2.2.1 La obtención de un conjunto de reglas.

Función ENCONTRARMEJORESREGLAS(*Ejemplos*, *NoCubiertos*) : Cjto. Reglas
 $C_R = \emptyset$
para cada clase C del problema **hacer**
 /* Seleccionar de entre los ej. no cubiertos de C , salvo si no quedan... */
 $NoCubiertos_C = \text{EJEMPLOSDECLASE}(C, NoCubiertos)$
si $NoCubiertos_C \neq \emptyset$ **entonces**
 $C_R = C_R \cup \text{INICIALIZARREGLAS}(NoCubiertos_C)$
si no
 $C_R = C_R \cup \text{INICIALIZARREGLAS}(EjemplosDeClase(C, Ejemplos))$
fin si
fin para
 GENERALIZACIÓN(C_R , *Ejemplos*)
retornar(C_R)

Por último, se aplica un post-proceso a la teoría inducida. En esta etapa se va a tratar de reducir el número total de reglas obtenidas utilizando dos procedimientos que se detallan en el Capítulo 4 y que son: un proceso de extensión y pegado (agregación) de reglas que puede dar lugar a la eliminación por contención de algunas de ellas, y un proceso de selección de reglas, tomado del sistema ABANICO [Ranilla, 98], que va a quedarse con un subconjunto de la *Teoría* sin perder precisión en la clasificación.

2.2 La selección inicial de instancias

La obtención de reglas parte, como se puede ver en el Algoritmo 2.2.1, de la selección de un conjunto de instancias cuyas descripciones van a ser extendidas todo cuanto se pueda, siguiendo un proceso descrito en detalle en el Capítulo 3. La selección de las muestras iniciales se hace de forma aleatoria sobre los ejemplos *aún no cubiertos* de cada clase. Por defecto el número de instancias seleccionadas y, por tanto, de reglas iniciales en cada ciclo es de 10, si bien está limitado por la ecuación (2.1) representada en la Figura 2.2.

$$M_{m\acute{a}x} = \min \left(10, \max \left(1, \left\lfloor e^{\frac{m}{50 \cdot 10^6}} \right\rfloor \right) \right) \quad (2.1)$$

El límite lo impone m , el número de ejemplos de la clase que es minoritaria en el conjunto de entrenamiento.

La razón para limitar el número de ejemplos o reglas de partida es que, en el post-proceso final del conjunto de reglas hay un mecanismo que elimina reglas que cubren pocos ejemplos si cree estar ante un problema con ruido. Este mecanismo utiliza el número de ejemplos de la clase menos abundante para filtrar y eliminar las reglas que considera ruidosas.

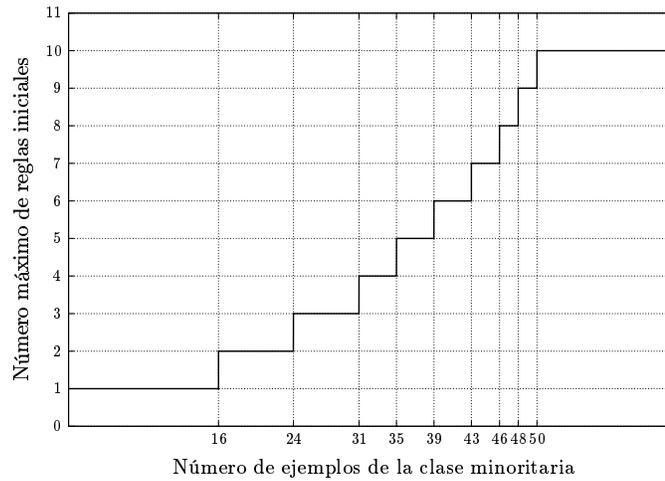


Figura 2.2: Número máximo de reglas iniciales por clase. El uso de esta función garantiza que, si repartimos uniformemente el número de ejemplos de la clase minoritaria entre las reglas iniciales, a cada una le corresponderá una cantidad suficiente como para sobrepasar el umbral que la consideraría ruidosa.

Pues bien, si la distribución de los ejemplos en el espacio de atributos es más o menos uniforme y para cada clase C se seleccionan M instancias como reglas de partida, entonces cabe esperar que, tras la generalización efectuada por INNER, cada una de estas reglas cubra aproximadamente N_C/M , donde N_C es el número de ejemplos de la clase C . Es evidente que, a mayor valor de M , menos ejemplos se espera que sean cubiertos por cada regla, y puede ocurrir que el proceso final antes mencionado elimine gran parte de esas reglas por considerarlas ruidosas, reduciendo al mismo tiempo la precisión en la clasificación. Estaríamos ante el problema ya observado por Holte en [Holte, Acker, Porter, 89] al hablar de las *pequeñas disyunciones*, reglas de poca cobertura.

El problema radica en que un número elevado de reglas haría que se estorbasen entre ellas, impidiendo un crecimiento adecuado ya que, como se verá en el Capítulo 3, las reglas crecen en función de los ejemplos no cubiertos próximos a ellas. Esta es también la razón por la que cuando alguna clase tiene ya todos sus ejemplos cubiertos, las reglas iniciales se construyen igualmente a partir de ejemplos cubiertos, como muestra el Algoritmo 2.2.1. Se podría renunciar a tomar reglas iniciales de esa clase, puesto que si ya están todos los ejemplos cubiertos, parece que el sistema ha aprendido a caracterizar esa clase. Sin embargo, puesto que en cada ciclo no se retiran los ejemplos cubiertos, si alguna clase no tiene reglas sus ejemplos van a afectar de forma negativa al crecimiento de las reglas de otras clases.

Una mejora en la selección de reglas iniciales viene de la mano de BETS, un sistema presentado en [Del Coz *et al.*, 99] que extrae un pequeño grupo de instancias *paradigmáticas* del conjunto de entrenamiento. Además, esas instancias sólo contienen los atributos relevantes en la región del espacio en la que se sitúan, por lo que pueden considerarse como unas reglas de partida ya ligeramente generalizadas y, sobre todo, bien situadas para ser *infladas* por INNER.

2.3 El cálculo de las distancias

El cálculo de distancias juega un papel fundamental en el sistema INNER, puesto que el mecanismo de clasificación se basa en la misma idea que se aplica en los sistemas basados en instancias: *un ejemplo será clasificado por la regla más cercana a él.*

Pero no sólo en la clasificación es importante el cálculo de distancias; INNER también necesita calcular distancias durante el proceso de obtención de reglas, tal como se pondrá de manifiesto al describir en la sección 3.1 el mecanismo concreto de generalización. En el resto de esta sección se muestran algunas de las posibilidades existentes para efectuar el cálculo de distancias y se presenta y justifica el método que usa INNER.

Atributos continuos

Cuando el problema a resolver sólo tiene atributos continuos suelen usarse con frecuencia funciones como la distancia Euclídea (2.2), la distancia de Chebychev (2.3), la distancia Manhattan (2.4), u otras descritas en [Wilson, Martinez, 97].

$$E(x, y) = \sqrt{\sum_{a=1}^m (x_a - y_a)^2} \quad (2.2)$$

$$Ch(x, y) = \max_{a=1}^m |x_a - y_a| \quad (2.3)$$

$$M(x, y) = \sum_{a=1}^m |x_a - y_a| \quad (2.4)$$

donde

- x e y son dos vectores de atributos que representan a dos ejemplos.
- m es el número de atributos usados para describir cada ejemplo.
- x_a e y_a son los valores del atributo a -ésimo en los ejemplos x e y respectivamente.

Estas funciones permiten medir la distancia entre dos puntos en el espacio de atributos. Teniendo en cuenta que los antecedentes de tipo continuo de una regla

vienen descritos por un intervalo cerrado de valores, la distancia del antecedente a -ésimo de una regla al atributo correspondiente en un ejemplo se calculará como la distancia entre el extremo más próximo y el valor del atributo en el ejemplo. Si el valor del atributo está entre el extremo inferior y el superior, la distancia para ese atributo es, obviamente, cero.

Atributos simbólicos

Sin embargo, en problemas en los que los atributos toman valores simbólicos o nominales, entre los que no hay un orden lineal definido, ninguna de las funciones antes citadas es útil. Este es el caso de atributos como *color* o *sabor*, donde no hay ninguna razón para pensar que la distancia entre *rojo* y *verde* tenga que ser mayor (o menor) que la distancia entre *rojo* y *azul*.

La forma más simple de medir la diferencia entre dos atributos simbólicos es considerar que la distancia es 0 si toman el mismo valor y 1 si toman distintos valores. Así, la distancia entre los valores x_a e y_a del atributo a viene definida por (2.5) y se denomina función de solapamiento¹.

$$\text{solapamiento}(x_a, y_a) = \begin{cases} 0, & \text{si } x_a = y_a \\ 1, & \text{si } x_a \neq y_a \end{cases} \quad (2.5)$$

La distancia entre dos puntos en el espacio de atributos simbólicos puede definirse, siguiendo una aproximación similar a la de la distancia Euclídea, como:

$$D(x, y) = \sqrt{\sum_{a=1}^m \text{solapamiento}^2(x_a, y_a)} \quad (2.6)$$

Esta función puede resultar excesivamente simple, ya que no hace uso de información adicional presente en el problema, que puede ser de ayuda en el proceso de generalización. Por ejemplo, podría ocurrir que los colores *rojo* y *verde* se dan en proporciones similares en todas las clases mientras la distribución de *azul* es muy distinta. En ese caso cabría pensar que el *rojo* y el *verde* son valores similares a los efectos de clasificación y muy diferentes a los casos de color *azul*. Para precisar esta idea se han diseñado otras funciones para medir distancias entre atributos simbólicos. Una de las más destacables es la VDM² de Stanfill y Waltz, cuya versión simplificada (SVDM usada en el sistema RISE [Domingos, 96]) se muestra en (2.7) y que ha servido de base para construir otras variantes como la HVDM, IVDM o la WVDM presentadas en [Wilson, Martinez, 97].

$$VDM(x_a, y_a) = \sum_{c=1}^C \left| \frac{N_{a,x_a,c}}{N_{a,x_a}} - \frac{N_{a,y_a,c}}{N_{a,y_a}} \right|^q = \sum_{c=1}^C |P_{a,x_a,c} - P_{a,y_a,c}|^q \quad (2.7)$$

¹ *Overlap* en la literatura anglosajona.

² *Value Difference Metric*.

donde

- $N_{a,x_a,c}$ es el número de instancias en el conjunto de entrenamiento cuyo valor en el atributo a es x_a y cuya clase es c .
- N_{a,x_a} es el número de instancias en el conjunto de entrenamiento cuyo valor en el atributo a es x_a . Obviamente, $N_{a,x_a} = \sum_{c=1}^C N_{a,x_a,c}$
- C es el número de clases presentes en el problema.
- q es una constante de tipo entero que debe ser determinada *ad hoc*.
- $P_{a,x_a,c}$ es la probabilidad de que una instancia sea de clase c condicionada a que el valor del atributo a es x_a .

La métrica VDM considera que dos valores están más próximos si las clasificaciones resultantes de esos dos valores son similares. Sin embargo, esta métrica no es directamente aplicable a problemas con atributos continuos, sino que es necesario efectuar una *discretización*, dividiendo los atributos en un determinado número de intervalos y tratándolos luego como si fuesen simbólicos. La desventaja que esto presenta es que se pierde información topológica inherente a los valores continuos ya que dos valores distintos que estén contenidos en el mismo intervalo serán considerados como si fuesen exactamente el mismo valor, aunque cada uno esté en un extremo del intervalo.

Problemas mixtos

Cuando los problemas presentan ejemplos con atributos de tipo simbólico y de tipo continuo se han de combinar distintas funciones para poder calcular distancias. Uno de los métodos más simples es el que utiliza la métrica HEOM³, combinando la distancia Euclídea normalizada y la función de solapamiento para los atributos continuos y simbólicos, respectivamente. Esta métrica calcula la distancia entre dos valores de un atributo a mediante:

$$d_h(x_a, y_a) = \begin{cases} 1, & \text{si } x_a \text{ o } y_a \text{ son desconocidos} \\ \text{solapamiento}(x_a, y_a), & \text{si } a \text{ es simbólico} \\ EN(x_a, y_a), & \text{si } a \text{ es contínuo} \end{cases} \quad (2.8)$$

donde EN es la distancia Euclídea normalizada y viene dada por:

$$EN(x_a, y_a) = \frac{|x_a - y_a|}{\max_a - \min_a} \quad (2.9)$$

siendo \max_a y \min_a el mayor y el menor valor, respectivamente, observados en el conjunto de entrenamiento para el atributo a .

³*Heterogeneous Euclidean-Overlap Metric.*

La distancia entre dos puntos en el espacio de atributos se calcula como:

$$HEOM(x, y) = \sqrt{\sum_{a=1}^m d_h(x_a, y_a)^2} \quad (2.10)$$

Esta es, con algunos matices que se comentan a continuación, la métrica utilizada por INNER. En primer lugar hay que puntualizar que la métrica HEOM se utiliza para clasificar ejemplos con un conjunto de reglas *una vez ha finalizado el proceso de aprendizaje*, pero el mecanismo es diferente durante el proceso de generalización y obtención de reglas. Además, puesto que siempre hay que calcular distancias entre reglas y ejemplos, es conveniente redefinir las fórmulas para que reflejen claramente cómo se hace este cálculo. El aspecto de las fórmulas para el cálculo de la distancia entre una regla y un ejemplo, usando HEOM, queda como sigue:

$$HEOM_{Inner}(Regla, Ejemplo) = \sqrt{\sum_{a=1}^m d_{Inner}(Ant_a, V_a)^2} \quad (2.11)$$

donde

- Ant_a es el valor del antecedente a de la regla. Será un intervalo $[x_1, x_2]$ si el antecedente es continuo o un conjunto discreto de valores $\{S_1, \dots, S_k\}$ si el antecedente es simbólico y puede tomar k valores diferentes.
- V_a es el valor del atributo a en el ejemplo.
- m es el número de atributos que describen cada ejemplo.

Teniendo esto en cuenta, la función d_{Inner} se define como:

$$d_{Inner}(Ant_a, V_a) = \begin{cases} 1, & \text{si } V_a \text{ es desconocido} \\ O_{Inner}(Ant_a, V_a) & \text{si } Ant_a \text{ es simbólico.} \\ E_{Inner}(Ant_a, V_a), & \text{si } Ant_a \text{ es continuo.} \end{cases} \quad (2.12)$$

donde

$$O_{Inner}(\{S_1, \dots, S_k\}, V_a) = \begin{cases} 0, & \text{si } V_a \in \{S_1, \dots, S_k\} \\ 1, & \text{si } V_a \notin \{S_1, \dots, S_k\} \end{cases} \quad (2.13)$$

$$E_{Inner}([x_1, x_2], V_a) = \begin{cases} 0, & \text{si } V_a \in [x_1, x_2] \\ \frac{\min\{|x_1 - V_a|, |x_2 - V_a|\}}{\max_a - \min_a}, & \text{si } V_a \notin [x_1, x_2] \end{cases} \quad (2.14)$$

Como se puede observar en las ecuaciones que describen la HEOM y, por supuesto, la $HEOM_{Inner}$, se introduce la capacidad de manipular ejemplos en los que no todos los atributos están presentes; en ese caso, cuando se desconoce el valor que toma

un atributo, lo que se hace es actuar de modo pesimista, suponiendo que para ese atributo desconocido la distancia es 1, es decir, es la máxima posible.

Se observa también que se normalizan las distancias en los antecedentes continuos. De lo contrario, si uno de los atributos toma valores en un rango mucho más amplio que los demás, tendrá mayor influencia que el resto de atributos en el cálculo de la distancia. Esto puede ser contraproducente puesto que, en cierta forma, equivale a dar a priori mayor importancia a un atributo que a los demás, ya que una “pequeña” variación dentro del rango de valores posibles para ese atributo puede suponer una gran diferencia en la distancia resultante.

Para ilustrar este indeseable efecto consideremos el siguiente problema artificial: se trata de un conjunto de 1500 ejemplos descritos por dos atributos continuos, X e Y , y dos clases, *DENTRO* y *FUERA*, de forma que aquellos puntos para los que $0.3 \leq X \leq 0.7$ o que $-200000 \leq Y \leq 200000$ son de la clase *DENTRO* y el resto son de la clase *FUERA*, tal como se muestra en la Figura 2.3. En este ejemplo, los valores en el eje X oscilan entre 0 y 1 y los valores en el eje Y oscilan entre -500000 y 500000 . Con este problema se hizo una prueba de validación cruzada estratificada, dividiendo el conjunto de entrenamiento en 10 porciones y repitiendo la prueba 5 veces, utilizando inicialmente una versión de INNER que usaba valores no normalizados y repitiendo luego el experimento *en las mismas condiciones*⁴ pero con una versión que normalizaba los valores entre 0 y 1.

Los resultados que se muestran corresponden al error medio obtenido \pm error estándar. Se observa una clara mejoría en los resultados donde se utilizan valores normalizados para el cálculo de las distancias:

	Cruz Alargada	
	normalizando	sin normalizar
Error \pm Err. Estándar	1.69% \pm 0.48%	10.23% \pm 0.88%

En [Wilson, Martinez, 97] se critica el uso de la métrica HEOM porque utiliza la función de solapamiento para antecedentes simbólicos y, como se ha comentado previamente, es demasiado simple y no aprovecha información del conjunto de entrenamiento. Esto es justificable en el caso de sistemas basados en instancias, puesto que al tomar una instancia como clasificador podría ocurrir que en un antecedente simbólico tuviese un valor muy poco frecuente para la clase que pretende representar. Eso llevaría a dicha instancia a estar más lejos de otros ejemplos de su clase de lo que sería deseable y quizá más cerca de los de otras clases, convirtiéndose así en un mal clasificador.

⁴En ambos casos la semilla para el generador de números aleatorios usado por INNER fue 34259944 y para construir los 50 pares de conjuntos de entrenamiento y prueba se usó la librería MLC++ [Kohavi *et al.*, 94] con semilla 2032. De esta forma se garantiza que en ambos experimentos se usaron los mismos conjuntos en las mismas condiciones de aleatoriedad interna del propio INNER.

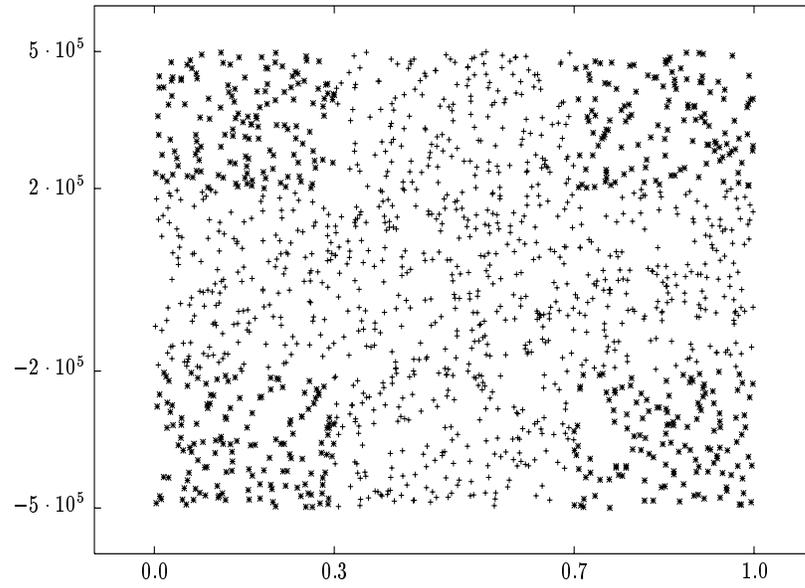


Figura 2.3: El problema de la cruz alargada, diseñado para mostrar la necesidad de utilizar distancias normalizadas.

Sin embargo, en el caso de INNER, los clasificadores no son ejemplos sino reglas, en cuyos antecedentes simbólicos puede aparecer más de un valor, extendiendo así el área de influencia de la regla o, dicho de otra forma, haciendo que esa regla se acerque a más ejemplos. Lo realmente interesante es que el conjunto de valores que definen un antecedente simbólico en una regla aprendida por INNER se obtiene mediante un proceso en el que *sí que se tiene en cuenta* la información presente en el conjunto de entrenamiento. Podemos decir que la métrica para los valores simbólicos también se aprende en INNER en un proceso similar al que se sigue para construir los mapas auto-organizados (SOM) de Kohonen [Kohonen, 95]; en la sección 3.3.2 se describe con detalle este mecanismo.

Además, este sistema pretende alcanzar una solución lo más explícita posible por lo que es interesante incorporar un criterio de aplicación de reglas sencillo que redunde en una mayor simplicidad y claridad en la expresión de las reglas obtenidas, por lo que $HEOM_{Inner}$ parece conveniente.

Capítulo 3

Generalización de instancias

3.1 Introducción

El proceso de inducción que sigue INNER consiste, como se ha descrito en la sección 2.1, en seleccionar algunas instancias del conjunto de entrenamiento que son generalizadas, para obtener así un conjunto de reglas. Esta generalización de instancias es un proceso en dos pasos, mostrados en la Figura 3.1; el primero puede considerarse trivial y consiste en reescribir la instancia como regla puntual; en la sección 3.2 se muestra un ejemplo de construcción de una regla a partir de una instancia. El segundo paso es un proceso de generalización de reglas aplicable, en particular, a reglas puntuales. A partir de este momento se hará referencia a la generalización de instancias y de reglas indistintamente.



Figura 3.1: Generalización de instancias.

El mecanismo de generalización de reglas, que se corresponde con el Algoritmo 3.1.1, consiste en presentar al sistema uno a uno los ejemplos de entrenamiento, modificando en cada paso la regla *más cercana* al ejemplo presentado. La modificación de una regla dará lugar a una tentativa de regla que más tarde se convertirá en definitiva mediante un proceso denominado *regularización* y sustituirá a la regla original, si su *calidad* es superior. La modificación que se efectúa debe *acercar* la regla al

ejemplo presentado si ambos son de la misma clase y debe *alejarse* si son de clases diferentes. Por último, un proceso denominado *calificación* efectúa una generalización por poda de antecedentes en las reglas obtenidas en los pasos anteriores.

Es evidente que, en el proceso de generalización, el cálculo de la distancia entre reglas y ejemplos es de suma importancia, puesto que es el criterio utilizado para decidir qué regla se va a modificar en cada presentación; una vez que se ha decidido cual va a ser esa regla, la modificación que se aplicará será inversamente proporcional a la distancia con el ejemplo presentado. No de menos importancia es el criterio que se usa para evaluar la calidad de una regla. En el resto de este capítulo se desarrolla en profundidad el proceso de generalización, detallando todos estos puntos.

Algoritmo 3.1.1 Generalización de instancias/reglas. Las reglas puntuales sufren un proceso de inflado y otro de simplificación del número de antecedentes. En el inflado, el conjunto *Ejemplos* de ejemplos de entrenamiento se presenta *M* veces al conjunto de reglas C_R que van a generalizarse. Los ejemplos se ordenan de forma aleatoria y luego se presentan de uno en uno, modificando en cada iteración la regla más cercana.

Procedimiento GENERALIZACIÓN($C_R, Ejemplos$)

INFLADO($C_R, Ejemplos$)

CUALIFICACIÓN($C_R, Ejemplos$)

Procedimiento INFLADO($C_R, Ejemplos$)

$N = \text{TAMAÑO}(Ejemplos) / * \text{Número de ejemplos de entrenamiento} * /$

para i desde 1 hasta M **hacer**

BARAJAEJEMPLOS($Ejemplos$)

para j desde 1 hasta N **hacer**

$Regla = \text{REGLAMÁSCERCANA}(C_R, E_j)$

$Regla = \text{GENERALIZAREGLA}(Regla, E_j)$

si CRITERIOREGULARIZACIÓN(j) **entonces**

REGULARIZACIÓN($C_R, Ejemplos$)

fin si

fin para

fin para

3.2 \mathcal{K} -HEOM: Una métrica auto-organizada

Como se puede ver en la descripción del Algoritmo 3.1.1, durante el proceso de generalización de reglas se usa la función `REGLAMÁSCERCANA`, que retorna la regla más próxima¹ al ejemplo E_j presentado. Esta función utiliza un procedimiento para

¹Si hubiese más de una posibilidad entonces se busca la regla más específica, esto es, la que tenga mayor número de antecedentes. Si persiste el empate, se retorna una al azar.

calcular distancias que denominamos \mathcal{K} -HEOM, y que difiere un poco del usado en la clasificación de ejemplos, una vez que ha finalizado el aprendizaje. La diferencia estriba en la manera de medir las distancias entre antecedentes y valores de tipo simbólico, que permite que la generalización sea capaz de decidir qué valores simbólicos deben incluirse en los antecedentes de las reglas.

La distancia entre una regla y un ejemplo vendrá dada por la expresión:

$$\mathcal{K}\text{-HEOM}(\text{Regla}, \text{Ejemplo}) = \sqrt{\sum_{a=1}^m d_{\mathcal{K}}(\text{Ant}_a, V_a)^2} \quad (3.1)$$

donde

$$d_{\mathcal{K}}(\text{Ant}_a, V_a) = \begin{cases} 1, & \text{si } V_a \text{ es desconocido} \\ D_{Inner}(\text{Ant}_a, V_a), & \text{si } \text{Ant}_a \text{ es simbólico} \\ E_{Inner}(\text{Ant}_a, V_a), & \text{si } \text{Ant}_a \text{ es continuo} \end{cases} \quad (3.2)$$

Para valores continuos se usa E_{Inner} , que es la distancia Euclídea normalizada, usada también en la clasificación, y que viene expresada por la ecuación (2.14), expuesta anteriormente. Para valores simbólicos se utiliza por cada antecedente Ant_a una *tabla de diferencias* T_a con tantas entradas como valores pueda tomar el atributo. En esa tabla figura, inicialmente, un 0 en el valor presente en la regla inicial (que, recordemos, es un ejemplo de conjunto de entrenamiento) y un 1 en los demás valores. Para medir la distancia de un antecedente al valor de un atributo en un ejemplo simplemente se consulta esa tabla y se retorna el contenido de la entrada² correspondiente al valor simbólico del atributo en el ejemplo, que es lo que hace la función D_{Inner} :

$$D_{Inner}(\text{Ant}_a, V_a) = \begin{cases} 0, & \text{si } T_a[V_a] \leq 0 \\ 1, & \text{si } T_a[V_a] \geq 1 \\ T_a[V_a], & \text{en otro caso} \end{cases} \quad (3.3)$$

Las reglas han de tener, pues, una representación interna que permita utilizar el mecanismo descrito para medir distancias. Esa representación interna puede verse en el ejemplo siguiente, que ilustra el proceso de conversión de una instancia en regla puntual: supongamos un problema en el que los ejemplos están descritos por un atributo continuo como *diámetro* y dos atributos simbólicos como *color* y *sabor*, que pueden tomar, respectivamente, los valores {rojo, verde, azul} y {amargo, dulce}. Si el sistema selecciona un ejemplo con *diámetro*=2.41, *color*=*azul* y *sabor*=*dulce* como regla inicial, ésta se construye con un intervalo de valores y dos tablas de diferencias, tal como se muestra a continuación:

DIÁMETRO		COLOR			SABOR	
Ext. Inf.	Ext. Sup.	rojo	verde	azul	amargo	dulce
2.41	2.41	1	1	0	1	0

²Esto no es exactamente así. En la sección 3.3.2 se explica por qué los valores en las tablas de diferencias pueden caer fuera del $[0, 1]$ y por qué D_{Inner} debe retornar un valor en $[0, 1]$

Esta tabla es la representación interna de la regla:

$$C \leftarrow DIÁMETRO \in [2.41, 2.41] \wedge COLOR \in \{azul\} \wedge SABOR \in \{dulce\} \quad (3.4)$$

A primera vista, utilizar $d_{\mathcal{K}}$ equivale a utilizar la función de solapamiento de la ecuación (2.5) para los antecedentes simbólicos. Sin embargo, el uso de estas tablas de diferencias permite *acercar* o *alejar* un antecedente a un valor ya que no hay más que modificar el contenido de la entrada adecuada. Y eso es exactamente lo que hace el proceso de generalización con los antecedentes simbólicos, tal como se explica en la sección 3.3.2.

Puede suceder que se seleccione una instancia con valores desconocidos como regla de partida para una clase C . En ese caso se procede de la siguiente manera:

- Si el valor desconocido es el de un atributo A de tipo continuo, la regla se construye como si ese atributo tuviese el valor medio de todos los valores que aparecen en A para el resto de ejemplos de su misma clase.
- Si el valor desconocido es el de un atributo simbólico, cada entrada en su tabla de diferencias toma el valor $1 - P(V_a|C)$, donde $P(V_a|C)$ es la probabilidad de que un ejemplo de clase C tome el valor V_a en el atributo A .

3.3 El proceso de generalización

Tal y como se comentaba en la sección 3.1, el algoritmo de generalización presenta de forma iterativa los ejemplos de entrenamiento, modificando en cada paso la regla más cercana a través de la función GENERALIZARREGLA.

La modificación dependerá de la *distancia* al ejemplo presentado y de su *clase* de forma que, si ambos son de la misma clase, la regla debe *crecer*, acercándose al ejemplo en proporción inversa a la distancia que los separa, con la intención de llegar a cubrirlo, mientras que si son de distintas clases la regla se ve rechazada y debe ser alejada de ese punto, también en función de la distancia. Así, una regla puntual puede ir creciendo y cubriendo cada vez más ejemplos a su alrededor, haciéndose más general y más explícita.

3.3.1 Modificación de antecedentes continuos

La forma de acercar o alejar las reglas a los ejemplos es de fácil implementación para antecedentes continuos, puesto que basta con modificar los extremos de los intervalos que los definen. La modificación se hace utilizando una expresión similar a la descrita en [Kohonen, 95] para construir mapas auto-organizados:

$$Ext = Ext \pm h_C(D, t) \cdot d_a \quad (3.5)$$

$$h_C(D, t) = \alpha_C(t) \cdot S_C(D) \cdot W_{Ext} \quad (3.6)$$

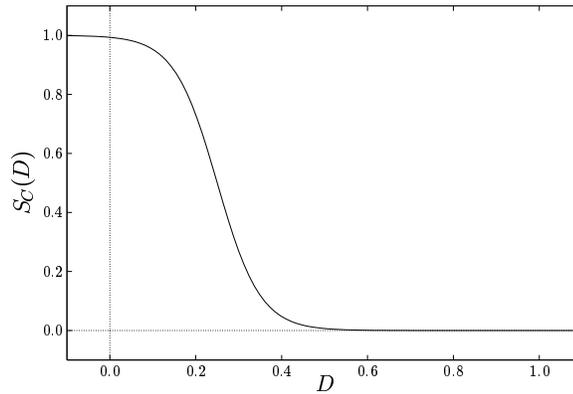


Figura 3.2: Sigmoide para atributos continuos.

donde

- Ext el extremo del intervalo más próximo al valor del atributo en el ejemplo, siempre y cuando el valor no caiga dentro del intervalo.
- d_a la distancia entre Ext y el valor del atributo en el ejemplo.
- D es la distancia de la regla al ejemplo.
- t es un contador que se incrementa con la presentación de cada ejemplo; puede considerarse como una metáfora del tiempo.
- $\alpha_C(t)$ es la *ganancia de aprendizaje* para antecedentes continuos y se define como $\alpha_C(t) = 0.75 \cdot (1 - t/M)$, donde M el número total de ejemplos que se van a presentar durante el entrenamiento.
- $S_C(D)$ es el *factor de vecindad* y viene definido por la sigmoide cuya expresión es $S_C(D) = 1 / (1 + e^{20x-5})$ que puede verse en la Figura 3.2.
- W_{Ext} es un factor que sirve para ponderar la influencia de la distancia al ejemplo en función de la diferencia en cada antecedente. Con este factor se pretende modificar cada extremo del intervalo dependiendo de cuanto influye la distancia en cada antecedente para la obtención de la distancia final entre el ejemplo y la regla. En la Figura 3.3 se explica gráficamente el efecto de este factor, que se calcula como $d_a / \sum_i d_i^2$, donde i itera sobre los antecedentes continuos de la regla afectada.

En la ecuación (3.5) se utilizará + si el extremo es el superior y el ejemplo es de la misma clase que la regla, o si el extremo es el inferior y ejemplo y regla son de clases diferentes. Se utilizará – en otro caso, es decir, si el extremo es el inferior y ejemplo y regla son de la misma clase, o si el extremo es el superior y el ejemplo es de clase diferente a la de la regla. La Figura 3.3 muestra como se efectúa la modificación de una regla tras haber sido atraída por un ejemplo de su misma clase.

3.3.2 Modificación de antecedentes simbólicos

Con los antecedentes simbólicos se pretende mantener la misma filosofía pero, ¿como acercar progresivamente un antecedente a un valor simbólico?. Puesto que los antecedentes de tipo simbólico cuentan con una tabla en la que se registra la diferencia con cada posible valor, basta con modificar adecuadamente la entrada correspondiente a ese valor. Este mecanismo es una versión ligeramente modificada del presentado en [Bahamonde *et al.*, 97]; las modificaciones de las entradas de la tabla se efectúan de la siguiente manera:

$$T_a[V_a] = T_a[V_a] \pm h_S(D, t) \cdot (T_a[V_a] + 1) \quad (3.7)$$

$$h_S(D, t) = \alpha_S(t) \cdot S_S(D) \quad (3.8)$$

donde

- V_a es el valor simbólico del atributo a en el ejemplo presentado.
- $T_a[V_a]$ es, como ya se ha indicado, el contenido de la entrada asociada al valor V_a en la tabla de diferencias del antecedente a .
- D es la distancia de la regla al ejemplo.
- t es el mismo contador utilizado en (3.5), que se incrementa con la presentación de cada ejemplo.
- $\alpha_S(t)$ es la *ganancia de aprendizaje* para antecedentes simbólicos y se define como $\alpha_S(t) = 0.675 \cdot (1 - t/M)$, donde M el número total de ejemplos que se van a presentar durante el entrenamiento.
- $S_S(D)$ es el *factor de vecindad* para atributos simbólicos, y viene dado por la sigmoide $S_S(D) = 1 / (1 + e^{10x-5})$ que puede verse en la Figura 3.4.

En la ecuación (3.7) se usará + cuando la regla y el ejemplo presentado sean de clases diferentes, para hacer aumentar la distancia entre ellos, y se usará – cuando la regla y el ejemplo sean de la misma clase.

El mecanismo de modificación descrito puede provocar que los valores en las tablas de diferencias desciendan por debajo del 0 o asciendan por encima del 1. Es especialmente delicada la situación en la que los valores se hacen negativos, ya que

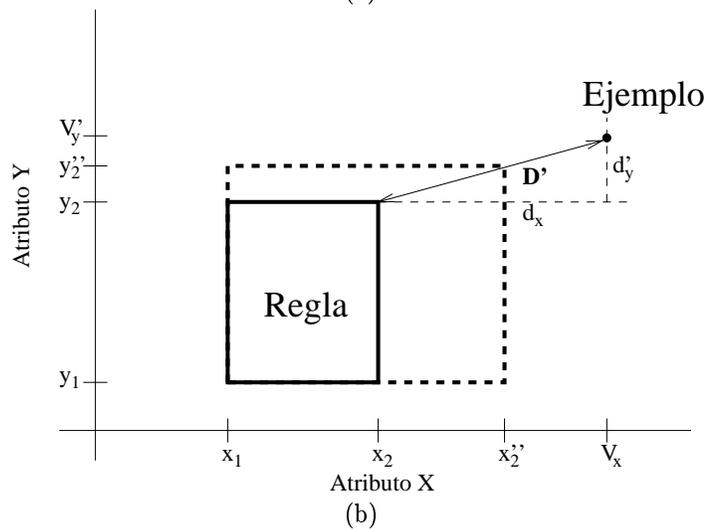
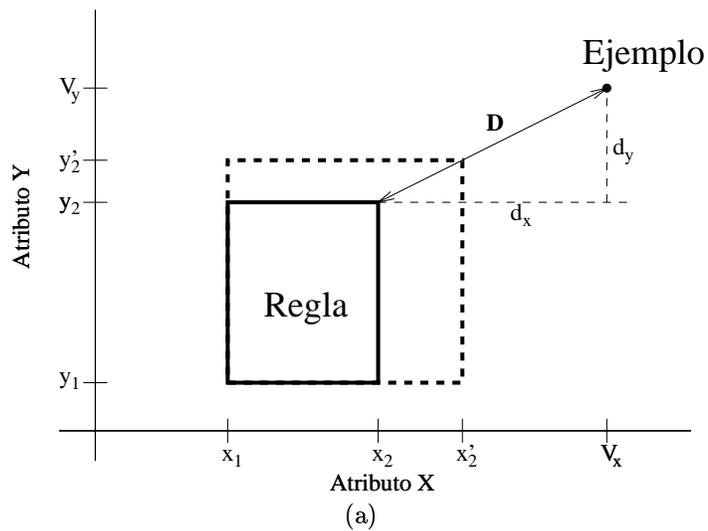


Figura 3.3: Modificación de los extremos de antecedentes continuos. Los antecedentes originales $X \in [x_1, x_2]$ e $Y \in [y_1, y_2]$ pasan a ser $X \in [x_1, x'_2]$ e $Y \in [y_1, y'_2]$. En la figura (b) puede apreciarse el efecto del factor de ponderación W_{Ext} , donde se puede ver que, aunque la distancia del antecedente X de la regla original al ejemplo es exactamente igual que en (a), d_x , en el segundo caso la modificación afecta mucho más al antecedente X que al Y .

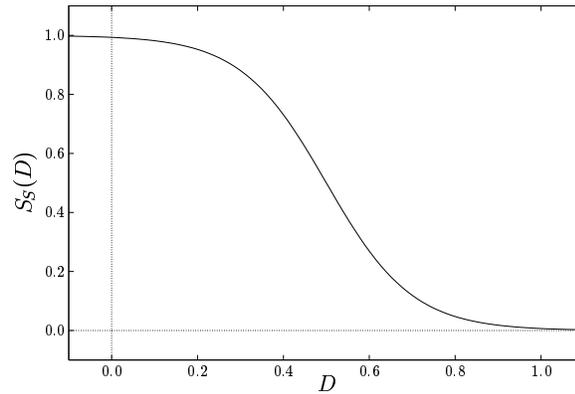


Figura 3.4: Sigmoide para atributos simbólicos.

no se pueden usar directamente en las fórmulas para calcular la distancia porque, en primer lugar, una distancia negativa es conceptualmente difícil de interpretar: ¿cómo puede un valor estar a menos distancia de la menor distancia posible?. En segundo lugar un valor negativo en la ecuación (3.1) sería elevado al cuadrado, convirtiéndose en un valor positivo que ayudaría a incrementar el valor final de la distancia. No parece razonable permitir que un antecedente cuya distancia a un valor es menor que cero (tremendamente próximo), colabore para decir que la regla está mas lejos de ese ejemplo que si fuese cero. Por eso el valor retornado por la función D_{Inner} de la ecuación (3.3) no se toma directamente de la tabla de diferencias sino que se acota en el intervalo $[0, 1]$. La representación gráfica de esta función puede verse en la Figura 3.5.

La variación de valores fuera del intervalo $[0, 1]$ se debe a la aparición del término $(T_a[V_a] + 1)$ en la ecuación (3.7). Según indica la expresión original utilizada por Kohonen, este término debería ser $T_a[V_a]$, pero eso nunca permitiría que las entradas en la tabla con diferencia inicial 0 viesen modificado su valor. Sin embargo, al usar la ecuación (3.7) cualquier valor inicial (0 ó 1) puede ser modificado. Si, desafortunadamente, se elige como regla inicial una instancia de clase C con ruido y en la que figura un valor en algún atributo que caracteriza, en presencia del resto de valores del ejemplo, a otras clases pero no a la clase C , entonces el mecanismo de generalización descrito es capaz de “expulsar” ese valor del antecedente.

Para mostrar un ejemplo de esta capacidad de auto-organización se presenta a continuación el comportamiento de INNER en un problema artificial ya presentado en la página 25, en el que los ejemplos pueden pertenecer a dos clases, A y B , y vienen descritos por un atributo continuo, *diámetro*, y dos atributos simbólicos, *color* y *sabor*, que pueden tomar los valores de los conjuntos {rojo, verde, azul} y {amargo,

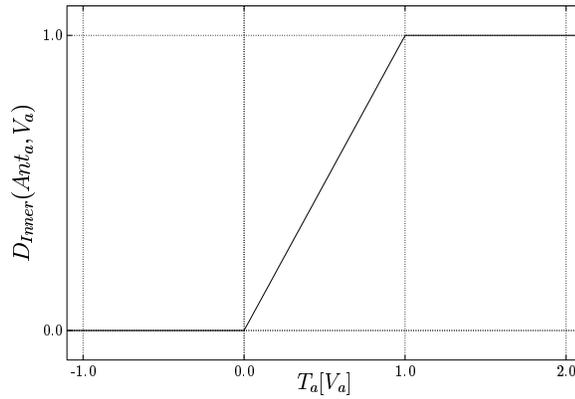


Figura 3.5: Función para acotar en $[0,1]$ la distancia entre un atributo y un valor simbólico.

dulce}, respectivamente. La relación existente entre los valores de los atributos y la clase de cada ejemplo es la siguiente:

Para cualquier valor del diámetro, si el color es rojo o verde y el sabor es dulce entonces el ejemplo es de clase A; en otro caso, es de clase B.

Para este problema se ha construido un conjunto de prueba con 200 ejemplos y un conjunto de entrenamiento con 1000 ejemplos, en el que algo más del 5% es ruido, es decir, son ejemplos etiquetados con la clase contraria a la deberían tener. En el experimento se fuerza a INNER a que inicialmente tome como regla puntual un ejemplo ruidoso, concretamente uno que presenta los valores *diámetro*=2.41, *color*=*azul*, *sabor*=*dulce* y está etiquetado con la clase A, así que se construye una regla con la representación interna que se mostraba en la página 25 y se etiqueta como regla de clase A. Esta es, evidentemente, una mala clasificadora para la clase A pero INNER, utilizando el procedimiento de auto-organización de antecedentes modifica esta regla de forma que, tras haber presentado parte del conjunto de entrenamiento, muestra los siguientes valores en las tablas de diferencias:

DIÁMETRO		COLOR			SABOR	
Ext. Inf.	Ext. Sup.	rojo	verde	azul	amargo	dulce
2.41	2.41	0.86	0.68	1.03	1.01	-0.08

Se puede observar³ que el valor azul ha sido rechazado y su entrada en la tabla de diferencias toma ahora un valor superior a 1. Además, los valores de las entradas correspondientes a los colores rojo y verde han comenzado a descender y ya son menores que 1: la regla ha comenzado a desplazarse hacia el rojo y verde y a alejarse del azul. Al final del proceso de generalización los cambios son mucho más evidentes:

DIÁMETRO		COLOR			SABOR	
Ext. Inf.	Ext. Sup.	rojo	verde	azul	amargo	dulce
2.41	2.41	-1	-1	2	2	-1

Con los valores presentes en las tablas de diferencias y teniendo en cuenta la definición de la función D_{Inner} expresada en la ecuación (3.3), el antecedente *color* está ahora a distancia 0 de ejemplos de color rojo y de color verde, y a distancia 1 de ejemplos de color azul, y el antecedente *sabor* está igual que antes de la generalización, a distancia 0 de ejemplos con sabor dulce, y a distancia 1 de ejemplos con sabor amargo. Nótese que, si a esta regla se le quita el antecedente que hace referencia al diámetro, es precisamente la regla que caracteriza perfectamente a la clase A y se ha obtenido partiendo de un ejemplo ruidoso. De la eliminación de atributos irrelevantes se encarga otro proceso que se describe en la sección 3.5.

3.4 Regularización

Tras la generalización de las reglas, cada antecedente tiene una propuesta de modificación; en un antecedente continuo las modificaciones pueden afectar a los dos extremos del intervalo que lo definen, mientras que en un antecedente simbólico las modificaciones pueden referirse a cada posible valor del atributo. Como se puede ver en el Algoritmo 3.1.1 que describe el proceso de generalización, cada vez que se cumple una determinada condición, denominada CRITERIOREGULARIZACIÓN, se procede a *regularizar* las reglas que han sido obtenidas hasta el momento. Pues bien, la regularización de una regla sirve para *consolidar la mejora que el proceso de generalización haya podido aportar a la regla inicial*, decidiendo cuales de las modificaciones propuestas son adecuadas y cuales no. Este proceso parte, pues, de unas reglas que pueden considerarse como “tentativas” y las convierte en definitivas utilizando el mecanismo descrito por los Algoritmos 3.4.1, 3.4.2 y 3.4.3 y que se detalla en esta sección.

Esta conversión dará lugar a un resultado que se situará generalmente entre dos posturas extremas: aceptar la regla generalizada tal como está o rechazar todas las modificaciones hechas a sus antecedentes, retomando la regla original tal como había quedado tras el proceso de regularización anterior. En la primera regularización que

³Aparentemente no ha habido modificación en el antecedente *diámetro*. Sin embargo sí la hubo, pero otro proceso, que aquí se obvia por mantener la claridad del ejemplo y que se detalla en la sección 3.4, se ha encargado de retomar los valores iniciales.

Algoritmo 3.4.1 Regularización de reglas.

Procedimiento REGULARIZACIÓN($C_R, Ejemplos$)
para cada regla $R \in C_R$ **hacer**
 REGULARIZAANTCONTÍNUOS($R, Ejemplos$)
 REGULARIZAANTSIMBÓLICOS($R, Ejemplos$)
fin para

efectúa el sistema INNER una regla podría, en el peor de los casos, volver a convertirse en la regla puntual de partida.

Al regularizar una regla hay que distinguir claramente entre el proceso seguido para los antecedentes continuos y para los simbólicos, no sólo porque la representación interna de los dos tipos de antecedentes es distinta, sino porque el mecanismo de generalización que sufren los antecedentes simbólicos hace innecesaria su regularización hasta el final del proceso de inducción.

Algoritmo 3.4.2 Regularización de antecedentes continuos. Para cada antecedente continuo se aplica la modificación de uno de sus extremos sugerida por el proceso de generalización, entonces se comprueba si el resultado es de mayor calidad que la regla original. Si lo es, la regla adopta la modificación y si no, el extremo vuelve a tomar el valor anterior. El proceso se repite con el otro extremo. El orden en el que se comprueban los extremos es: primero el que haga crecer más (o haga decrecer menos) el intervalo.

Procedimiento REGULARIZAANTCONTÍNUOS($R, Ejemplos$)
 $R_o = \text{REGLAORIGINAL}(R)$
 $C_o = \text{CALIDAD}(R_o, Ejemplos)$
para cada antecedente continuo A de R **hacer**
 para cada extremo Ext de A (ordenados por crecimiento) **hacer**
 $R' = \text{APLICARMODIFICACIÓN}(R_o, Ext)$;
 $C_n = \text{CALIDAD}(R', Ejemplos)$
 si C_n es superior a C_o **entonces**
 $C_o = C_n$
 $R_o = R' /*$ Se admite la modificación del extremo $*/$
 fin si
 fin para
fin para

Antecedentes continuos

Para ilustrar la necesidad de regularizar los antecedentes continuos con frecuencia supongamos un problema cuyos ejemplos pueden ser de dos clases, *positiva* o *negativa*,

y vienen descritos por dos atributos continuos. Supongamos también que, en alguna región del espacio de atributos los ejemplos están distribuidos tal como muestra la Figura 3.6, con dos regiones (A) y (B) que contienen ejemplos de la clase positiva y una región (C), con ejemplos de clase negativa, que se interpone entre ambas. Si tenemos una regla de clase positiva que cubre parte de la región (A), al presentar los ejemplos descubiertos e ir acumulando el efecto (de atracción o repulsión) que cada uno produce sobre la regla original, es muy posible que el resultado final sugiera modificar la regla hasta cubrir la región (B). Bastaría con que el azar llevase al sistema a presentar antes muchos más ejemplos de la región (B) que de la (C); para cuando se presentasen los ejemplos de la región (C), éstos ya estarían cubiertos y no provocarían el deseable efecto de rechazo sobre los límites de la regla; hay que tener en cuenta que en cuanto una regla invade una región, los ejemplos que quedan cubiertos ya no van a producir ningún efecto en los antecedentes continuos, aunque sean de clase diferente a la de la regla y estén, probablemente, cubiertos por error.

Si se efectúa la regularización para consolidar esa modificación, la nueva regla resulta ser de peor calidad que la original puesto que clasifica incorrectamente muchos casos de la región (C) que son de clase negativa, por lo que se rechazará la modificación y se retornará a la regla original. Hubiera sido más productivo regularizar con mayor frecuencia, tratando de consolidar pequeños cambios que darían lugar a una regla que cubriese de forma más adecuada la región (A); en sucesivos ciclos del algoritmo general de INNER se podrían seleccionar otras instancias de las regiones (B) y (C) que diesen lugar a reglas para esas zonas.

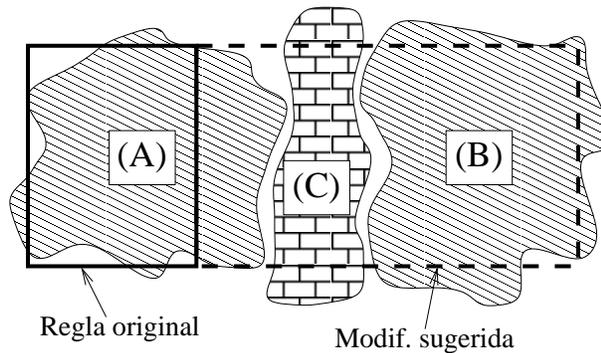


Figura 3.6: Crecimiento excesivo de una regla. Si en determinadas situaciones se acumula el efecto *generalizador* de muchos ejemplos es posible que la modificación final sugerida sea demasiado brusca y el proceso de regularización no la acepte.

El criterio para decidir cuando regularizar depende del número de ejemplos presentados; por defecto se utiliza un valor umbral que se corresponde con el 20% del

tamaño del conjunto de entrenamiento, salvo si el conjunto de entrenamiento tiene 100 ejemplos o menos; en ese caso el umbral por defecto es el 50% de su tamaño.

Antecedentes simbólicos

En el caso de los antecedentes simbólicos el procedimiento es diferente, ya que la presentación de un ejemplo afecta *siempre* a los antecedentes simbólicos de una regla y los valores contenidos en las tablas de diferencias pueden oscilar durante la generalización, aumentando o disminuyendo con la presentación de cada ejemplo. Por tanto se puede acumular el efecto producido por los ejemplos a lo largo de todo el proceso de generalización sin sufrir las consecuencias descritas para los antecedentes continuos. De hecho, es posible que un ejemplo convertido en regla inicial y, por tanto, cubierto por ésta, acabe rechazando valores contenidos inicialmente en sus antecedentes simbólicos como se mostró en la sección 3.3.2. El proceso de generalización de antecedentes simbólicos va a permitir que una regla de baja calidad evolucione hasta convertirse en una regla mejor, sin necesidad de hacer comprobaciones frecuentes con el proceso de regularización.

No obstante, los antecedentes simbólicos también necesitan ser regularizados. La regularización nos permite llevarlos al terreno de lo explícito ya que, durante la generalización de reglas, los valores que puede tomar un antecedente simbólico para satisfacer una regla tienen una representación equivalente a un conjunto difuso y como resultado del aprendizaje pretendemos obtener reglas explícitas. La función \mathcal{G} que se utiliza para decidir cuando se debe incorporar un valor simbólico al conjunto de valores de un antecedente se denomina *grado de pertenencia* y se representa en la Figura 3.7, de forma que los valores sólo se añaden si esta función retorna 1 o, dicho de otra forma, si la entrada en la tabla de diferencias de ese valor está por debajo del valor umbral -0.7 . En el Algoritmo 3.4.3 esta función se implementa en GRADOPERTENENCIA.

Retomando el ejemplo de la página 32, en el que se mostraba como se había modificado la representación interna de una regla durante la generalización, podemos comprobar que la regla regularizada será:

$$A \leftarrow DIÁMETRO \in [2.41, 2.41] \wedge COLOR \in \{rojo, verde\} \wedge SABOR \in \{dulce\} \quad (3.9)$$

ya que $\mathcal{G}(COLOR[rojo])$, $\mathcal{G}(COLOR[verde])$ y $\mathcal{G}(SABOR[dulce])$ son menores que el umbral -0.7 . El proceso de poda de antecedentes que se explica en la sección 3.5 se encargará más tarde de eliminar el *DIÁMETRO*, con lo que se obtendrá la regla que describe perfectamente a la clase *A*.

3.4.1 La calidad de una regla

La medida de calidad de las reglas es el factor decisivo para consolidar o no las modificaciones sugeridas durante la generalización. El sistema INNER utiliza para

Algoritmo 3.4.3 Regularización de antecedentes simbólicos. En cada antecedente simbólico se comprueba cada posible valor V_a que pueda tomar. La función GRADOPERTENENCIA indica si V_a debería ser incluido en el antecedente o no en función de la entrada correspondiente en la tabla de diferencias. Se construye una regla R' en la que se sustituye el antecedente que se está regularizando por otro en el que se incluye (o excluye) el valor V_a . Si eso supone una modificación en el antecedente entonces se comprueba la calidad de la nueva regla R' y si es mejor, se adopta la modificación.

Procedimiento REGULARIZAANTSIMBÓLICOS(R , $Ejemplos$)

$R_o = \text{REGLAORIGINAL}(R)$;

$C_o = \text{CALIDAD}(R_o, Ejemplos)$;

para cada antecedente simbólico A de R **hacer**

para cada valor V_a que pueda tomar A **hacer**

si $\text{GRADOPERTENENCIA}(V_a, A) = 1$ **entonces**

$R' = R_o$ con $A' = A \cup \{V_a\}$

si no

$R' = R_o$ con $A' = A \setminus \{V_a\}$

fin si

si $(V_a \in A \wedge V_a \notin A') \vee (V_a \notin A \wedge V_a \in A')$ **entonces**

 /* Si cambia el conjunto de valores que puede tomar el antecedente... */

$C_n = \text{CALIDAD}(R', Ejemplos)$

si C_n es superior a C_o **entonces**

$C_o = C_n$

$R_o = R'$ /* Se admite la incorporación o eliminación del valor V_a */

fin si

fin para

fin para

fin para

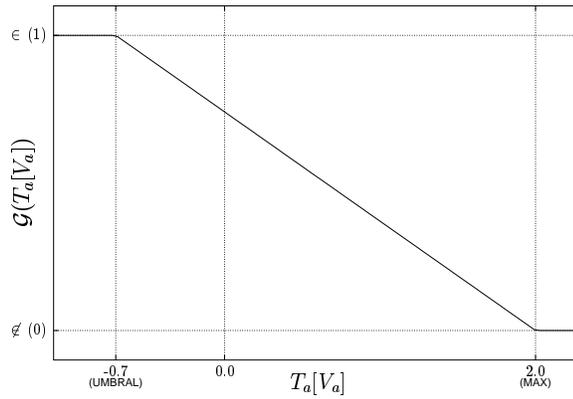


Figura 3.7: Grado de pertenencia de un símbolo a un antecedente. La función \mathcal{G} indica, tras un proceso de generalización, el grado de pertenencia de un valor a un conjunto de valores de un antecedente simbólico. Un valor cuya función de pertenencia sea, digamos, 0.9 “casi” pertenecerá al conjunto.

este fin la misma medida que el sistema ABANICO, el *nivel de impureza*, presentado en [Ranilla, Mones, Bahamonde, 98].

La regularización es un proceso en el que continuamente se debe elegir entre una regla y una versión modificada de ella. Cuando se ha de elegir entre distintas reglas es deseable utilizar un criterio de evaluación que nos lleve a seleccionar aquellas reglas que, previsiblemente, aporten un mayor número de aciertos en la clasificación de ejemplos no usados (no vistos) durante el período de aprendizaje.

La medida más elemental que se podría usar es el número de aciertos y fallos de una regla sobre el conjunto de entrenamiento. Sin embargo, esto no suele dar buenos resultados como se muestra en [Ranilla, Mones, Bahamonde, 98]. No parece muy equilibrado usar la proporción de aciertos sobre los ejemplos que la regla cubre explícitamente. Por ejemplo, parece razonable que una regla que sólo cubre un ejemplo y lo clasifique bien se considere peor que una regla que cubra y clasifique correctamente 10 ejemplos. Además, también hay que tener en cuenta que el conjunto de entrenamiento puede presentar ruido y en ese caso, la calidad de una regla que cometa algunos errores puede verse compensada si clasifica un gran número de ejemplos. Por ello, también parece razonable considerar que una regla que tenga 10 fallos entre 100 es mejor que una que tenga un fallo de 10 ejemplos que clasifica.

Lo que va a tratar de hacer, en definitiva, el nivel de impureza, es combinar la probabilidad p de acierto de una regla, definida como $p = \text{aciertos}/n$, con el número de ejemplos n que la regla clasifica de forma *nítida*; esto es, los ejemplos que hacen ciertos los antecedentes de dicha regla o, en términos geométricos, los ejemplos cubiertos por

dicha regla, sin aplicar la regla por distancia.

Se va a utilizar el *intervalo de confianza* de p como la probabilidad de éxito de cada regla (Aha utiliza esto para elegir los casos representativos en su sistema IB3 [Aha, 90], también basado en el principio del vecino más próximo). Dado un nivel de confianza α se va a calcular el intervalo de confianza como [Spiegel, 70, p. 162]:

$$\frac{p + \frac{z^2}{2n} \pm z \sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \quad (3.10)$$

Usando la ecuación 3.10 obtenemos el extremo inferior y el superior del intervalo de confianza, que vamos a representar como $[\text{Inf}(R), \text{Sup}(R)]$. El valor de z depende del nivel de confianza α y se obtiene de una tabla de distribución normal.

Pero además, si queremos comparar reglas de clases diferentes habrá que considerar la dificultad de acertar en la clasificación de ejemplos de cada clase. Para ello se va a calcular, para cada clase C , el intervalo de confianza de la *regla canónica del azar*. Esta regla canónica es una regla sin antecedentes que concluye la clase C :

$$C \leftarrow$$

Si representamos el intervalo de confianza de la regla canónica del azar como $[\text{Inf}(C), \text{Sup}(C)]$, entonces el nivel de impureza de la regla R viene definido por la ecuación (3.11).

$$\text{Nivel de Impureza}(R) = 100 * \frac{\text{Sup}(C) - \text{Inf}(R)}{\text{Sup}(R) - \text{Inf}(R)} \quad (3.11)$$

Es evidente que, a la hora de seleccionar entre varias reglas, preferiremos aquellas cuyo nivel de impureza sea *menor*. El Algoritmo 3.4.4 muestra como se implementa el cálculo de esta medida de calidad en INNER.

Durante la generalización se hace un uso intensivo de este algoritmo, ya que el nivel de impureza sirve para decidir si cada una de las modificaciones propuestas en los antecedentes de las reglas ha de ser aceptada o rechazada. Eso quiere decir que, para una regla con, digamos, 10 antecedentes continuos hay que calcular el nivel de impureza 21 veces como máximo; la primera vez se calcula el nivel de impureza de la regla original y luego puede haber hasta 20 sugerencias de modificación de los antecedentes (dos por cada antecedente) que hay que evaluar una a una. El cálculo del nivel de impureza requiere conocer cuantos de los *ejemplos cubiertos* por una regla son aciertos y cuantos son fallos lo que supone, a primera vista, recorrer 21 veces el conjunto de entrenamiento para ver qué ejemplos quedan cubiertos por cada posible tentativa de regla.

Sin embargo, para acelerar el proceso de cálculo, el algoritmo tendrá en cuenta únicamente un subconjunto de los ejemplos de entrenamiento con aquellos que, como mucho, quedarán cubiertos al final del proceso de regularización. La Figura 3.8 muestra en qué consiste esta optimización de forma gráfica.

Algoritmo 3.4.4 El nivel de impureza de una regla. Esta medida de calidad de las reglas se utiliza en muchas ocasiones en el sistema INNER. Aunque por claridad en los algoritmos siempre se utiliza la función CALIDAD, en realidad se está utilizando este algoritmo.

Función NIVELIMPUREZA(*Regla, Ejemplos*) : Nivel de Impureza
 $ICA = \text{INTERVALOCONFIANZA}(\text{AZAR}(\text{CLASE}(\text{Regla})))$
 $Cubiertos = \text{COBERTURA}(\text{Regla}, \text{Ejemplos})$
 $N = \text{TAMAÑO}(Cubiertos)$
si $N \neq 0$ **entonces**
 $Aciertos = \text{ACIERTOS}(\text{Regla}, Cubiertos)$
 $Fallos = \text{FALLOS}(\text{Regla}, Cubiertos)$
 $p = \text{Aciertos}/N$
 $IC = \text{INTERVALOCONFIANZA}(p, N, \alpha) /* \alpha = 0.95 */$
 $\text{NivelImpureza} = 100 \cdot (\text{SUP}(ICA) - \text{INF}(IC)) / (\text{SUP}(IC) - \text{INF}(IC))$
si no
 $\text{NivelImpureza} = 100 \cdot \text{SUP}(ICA)$
fin si
retornar (NivelImpureza)

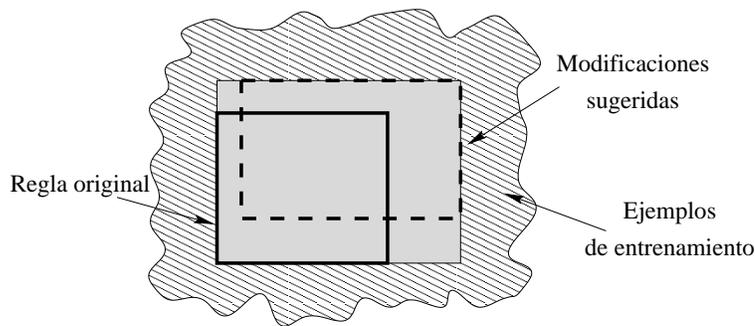


Figura 3.8: Optimización en el cálculo del nivel de impureza. Si la generalización de la regla dibujada con trazo continuo sugiere las modificaciones representadas con el trazo discontinuo, se puede llegar a tener una nueva regla que, como máximo, cubrirá los ejemplos de la región sombreada. Esto ocurrirá si se aceptan las modificaciones de los extremos superiores de los antecedentes y se rechazan las de los extremos inferiores. En cualquier otro caso, la regla resultante cubrirá un subconjunto de esos ejemplos así que, para calcular el nivel de impureza de cada una de las posibilidades no es necesario recorrer todos los ejemplos de entrenamiento de la zona rayada sino sólo los comprendidos en la región rectangular sombreada.

3.5 Cualificación

En el Algoritmo 3.1.1 puede verse que, tras una serie de iteraciones en las que se van generalizando reglas se aplica a éstas un procedimiento denominado *cualificación*. Este proceso, heredado del sistema ABANICO, trata de *simplificar* las reglas obtenidas hasta el momento mediante la reducción del número de antecedentes. La cualificación puede considerarse también como una generalización de las reglas a las que se aplica, ya que eliminar un antecedente implica aumentar la cobertura de la regla, si bien el crecimiento así conseguido es más brusco que el obtenido con la presentación de ejemplos. La Figura 3.9 muestra un ejemplo de cualificación.

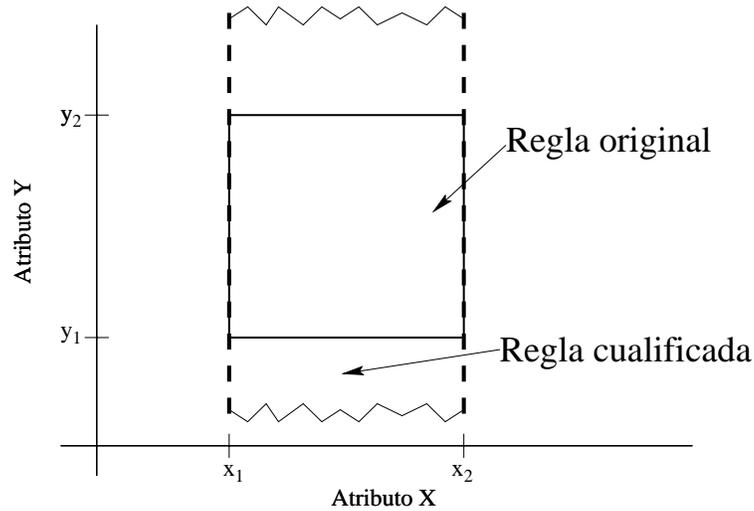


Figura 3.9: Cualificación de una regla. Tras la eliminación del antecedente Y , la regla $C \leftarrow X \in [x_1, x_2] \wedge Y \in [y_1, y_2]$, dibujada con trazo continuo, pasa a ser $C \leftarrow X \in [x_1, x_2]$ dibujada con trazo discontinuo.

El mecanismo de poda implementado por la cualificación evita el sobre-ajuste de las reglas a los ejemplos de entrenamiento. Para este fin utiliza el ya descrito nivel de impureza como evaluador de calidad de las reglas que resultan de la poda de antecedentes.

En [Fürnkranz, 97] se describen de forma general distintos procesos de poda⁴, clasificándolos en:

Poda previa: Tratan el ruido durante la generación de conceptos, esto es, durante la inducción de reglas. La idea es autorizar al meca-

⁴Se habla de *pre-pruning*, *post-pruning* y de la integración de ambos.

Algoritmo 3.5.1 Cualificación de reglas. El proceso de cualificación de un conjunto de reglas consiste en sustituir cada regla por un conjunto de descripciones parciales obtenidas a partir de ella. Esas descripciones parciales se generan en GENERARCANDIDATAS y ELIMINARANTREDUNDANTES y sufren finalmente un filtrado en FILTRADOCANDIDATAS.

Procedimiento CUALIFICACIÓN(C_R , *Ejemplos*)

ReglasCualificar = C_R

para cada regla $R \in$ *ReglasCualificar* **hacer**

Candidatas = GENERARCANDIDATAS(R , *Ejemplos*)

R' = MEJORREGLA(*Candidatas*) /* La de menor nivel de impureza */

Candidatas = *Candidatas* \cup ELIMINARANTREDUNDANTES(R' , *Ejemplos*)

Candidatas = FILTRADOCANDIDATAS(*Candidatas*, *Ejemplos*)

$C_R = (C_R \setminus R) \cup$ *Candidatas*

fin para

nismo de inducción para que construya reglas que, individualmente consideradas, puedan cometer algunos fallos de clasificación.

Poda final: Intentan mejorar la teoría aprendida en una fase de post-proceso final. Su objetivo es borrar condiciones redundantes del cuerpo de las reglas, así como borrar reglas innecesarias del conjunto final. Este método también evita el sobre-ajuste, es decir, que también sirve para tener en cuenta la posible presencia de ruido si se utiliza sólo parte del conjunto de entrenamiento para obtener las versiones podadas de las reglas y usando el resto del conjunto de entrenamiento para evaluar su calidad/precisión.

Poda intermedia: Es un proceso intermedio que se aplica a reglas cuyo proceso de obtención puede considerarse finalizado pero que no constituyen el conjunto de reglas final; todavía es posible obtener más reglas que, a su vez, podrán ser podadas. Un ejemplo de este tipo de poda es el que lleva a cabo el algoritmo I-REP, descrito también en [Fürnkranz, 97]. En este sentido, la cualificación ha de considerarse como un proceso de poda intermedia, similar al algoritmo I-REP.

En el Algoritmo 3.5.1 pueden apreciarse las tres fases en las que se divide el proceso de cualificación de un conjunto de reglas que se ilustran en la Figura 3.10 a través de un ejemplo. A continuación y de forma resumida, se detalla en qué consiste cada una de ellas.

Generación de reglas candidatas

La cualificación va a generar, por cada regla procesada, un conjunto de descripciones parciales denominadas *reglas candidatas*. Estas reglas se obtienen con el Algoritmo 3.5.2 que describe la función GENERARCANDIDATAS; esta función actúa de la siguiente manera:

- *Ordenación de antecedentes:* Primero se ordenan los antecedentes de la regla a cualificar por orden de calidad, de mejor a peor. La medida de calidad viene dada por el nivel de impureza de forma que, para cada antecedente se calcula el nivel de impureza que tendría una hipotética regla que concluyese la misma clase que la regla a cualificar pero que solo tuviese ese antecedente.
- *Generación de descripciones:* Con la lista ordenada de antecedentes de la regla original se van construyendo, una a una, descripciones parciales de tal forma que la i -ésima descripción tendrá los i primeros antecedentes de la lista. Cada descripción ha de pasar un filtro que requiere que acierte al menos el 80% de los ejemplos que cubre. La generación de las descripciones parciales terminará cuando se hayan utilizado todos los antecedentes o cuando se encuentre alguna descripción que clasifique correctamente el 100% de los ejemplos cubiertos.

Eliminación de atributos redundantes

Esta fase del algoritmo de cualificación se usa para explorar más combinaciones de antecedentes, buscando incrementar la calidad del conjunto de descripciones sin aumentar la complejidad de las reglas. El proceso, que puede verse en el Algoritmo 3.5.3, parte de la descripción de mayor calidad⁵, es decir, de la regla de menor nivel de impureza obtenida en la fase anterior y hace un recorrido de sus antecedentes de derecha a izquierda (en sentido contrario a la fase anterior) comenzando por el penúltimo y evaluando la posibilidad de prescindir de cada uno de los antecedentes.

Filtrado de reglas candidatas

Una última fase que se corresponde con el Algoritmo 3.5.4 se encarga de hacer una selección entre todas las descripciones generadas a partir de la regla original. Este filtrado sólo va a permitir que se añadan al conjunto de reglas aquellas descripciones con una calidad, medida como siempre en términos de niveles de impureza, próxima a la de la mejor descripción. Las descripciones que consigan pasar este filtro sustituirán a la regla de la que fueron obtenidas en el conjunto de reglas que está siendo cualificado. Nótese que un caso particular de descripción de una regla podría ser la propia regla original por lo que no se puede asegurar que, al cualificar una regla, ésta vaya a desaparecer del conjunto.

⁵En caso de empate, la más específica.

Algoritmo 3.5.2 Generación de reglas candidatas. Partiendo de una regla sin antecedentes se van añadiendo uno a uno los antecedentes de la regla original, dando lugar así a un conjunto de candidatas. El conjunto de candidatas depende del orden en que se añaden los antecedentes, que viene dado por el nivel de impureza.

Función GENERARCANDIDATAS(*Regla*, *Ejemplos*) : Cjto. Reglas
 $C = \text{CLASE}(\text{Regla})$
 $L = \emptyset$
para cada antecedente A de R **hacer**
 $R_1 = \text{CONSTRUIRREGLA}("C \leftarrow A")$
 $\text{Calidad} = \text{CALIDAD}(R_1, \text{Ejemplos})$ /* Nivel de Impureza */
 $L = \text{INSERTARENORDEN}(A, \text{Calidad}, L)$
fin para
 $\text{Descripciones} = \emptyset$
 $\text{Descripción} = \text{CONSTRUIRREGLA}("C \leftarrow ")$
para cada antecedente A de L **hacer**
 $\text{Descripción} = \text{Descripción} \oplus A$ /* donde \oplus denota añadir un antecedente */
si PORCENTAJEACERTOS(Descripción) > 80% **entonces**
 $\text{Descripciones} = \text{Descripciones} \cup \{\text{Descripción}\}$
si PORCENTAJEACERTOS(Descripción) = 100% **entonces**
salir del bucle
fin si
fin para
retornar(Descripciones)

Algoritmo 3.5.3 Eliminación de antecedentes redundantes. Para cada antecedente desde el penúltimo al primero se construye una nueva descripción en la que se elimina dicho antecedente; se evalúa la probabilidad de acierto sobre los ejemplos que cubre esa nueva regla y, si es mayor o igual que la de la descripción inicial, se añade al conjunto de descripciones.

Función ELIMINARANTREDUNDANTES(*Regla*, *Ejemplos*) : Cjto. Reglas
Descripciones = \emptyset
PorcentajeOriginal = PORCENTAJEACIERTOS(*Regla*)
MejorR = *Regla*
para cada antecedente *A* de *MejorR* desde el penúltimo al primero **hacer**
 R' = *MejorR* \ *A*
 si PORCENTAJEACIERTOS(*R'*) \geq *PorcentajeOriginal* **entonces**
 Descripciones = *Descripciones* \cup {*R'*}
 MejorR = *R'*
 fin si
fin para
retornar(*Descripciones*)

Algoritmo 3.5.4 Filtrado de reglas candidatas. De todas las descripciones obtenidas a partir de una regla en las fases anteriores sólo nos quedaremos con aquellas que tengan un nivel de impureza como máximo UMBRALFILTRADO puntos por encima del valor de la regla de mayor calidad. El valor por defecto de este umbral es de 10 puntos.

Función FILTRADOCANDIDATAS(*Candidatas*, *Ejemplos*) : Cjto. Reglas
Umbral = MEJORNIVELIMPUREZA(*Candidatas*) + UMBRALFILTRADO
Filtradas = \emptyset
para cada regla *R* \in *Candidatas* **hacer**
 si CALIDAD(*R*) es superior a *Umbral* **entonces**
 /* Para que la calidad de *R* sea superior a un umbral, su nivel de impureza ha de ser menor que ese umbral */
 Filtradas = *Filtradas* \cup {*R*}
 fin si
fin para
retornar(*Filtradas*)

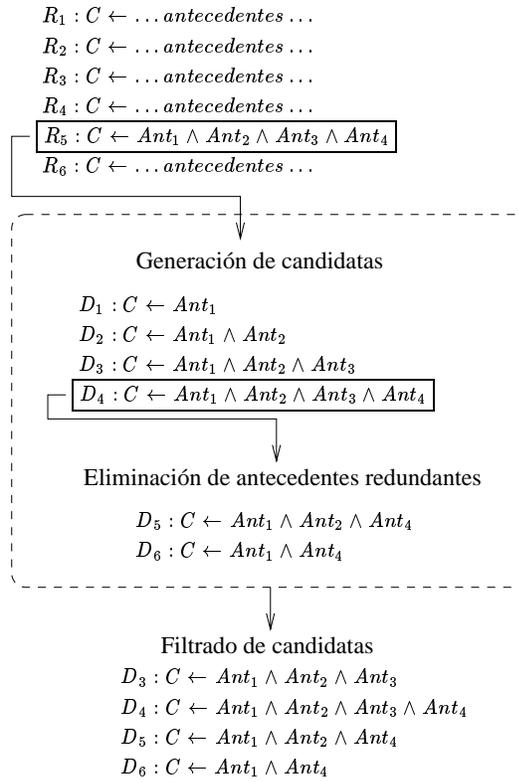


Figura 3.10: Fases de la cualificación de una regla. Suponiendo que se va a cualificar la regla R_5 , cuyos antecedentes ya están ordenados por nivel de impureza, la generación de reglas candidatas daría lugar a las descripciones D_1 a D_4 . Si la descripción D_4 resulta ser la que tiene mayor calidad (menor nivel de impureza) entonces sobre ella se aplica la fase de eliminación de antecedentes redundantes. En este ejemplo suponemos que es eliminado Ant_3 y, posteriormente, Ant_2 , dando lugar a las descripciones D_5 y D_6 . La fase de filtrado de candidatas toma todas las descripciones D_i generadas y elimina aquellas cuyo nivel de impureza sobrepasa un cierto umbral. En este caso sólo D_3 , D_4 , D_5 y D_6 consiguen pasar. En el conjunto original de reglas se sustituye, pues, la regla R_5 por estas cuatro descripciones.

Capítulo 4

Post-proceso de reglas

4.1 Introducción

Tras haber obtenido un conjunto de reglas fruto de la unión de los distintos subconjuntos inducidos por el proceso iterativo de generalización y cualificación expuesto en el Capítulo 3, INNER trata ahora de reducir el número de reglas resultante sin perder precisión en la clasificación, aplicando lo que se podría llamar un post-proceso de la teoría aprendida. Este proceso, cuya descripción en pseudocódigo puede verse en el Algoritmo 4.1.1, consta de varias etapas que se van a detallar en este capítulo.

Algoritmo 4.1.1 Post-proceso de reglas. Las reglas inducidas sufren un proceso de extensión, una selección en función del rendimiento y una nueva extensión. Por último se aplica, en algunos casos, un último proceso de inflado. La extensión previa a la selección y la extensión posterior son ligeramente distintas, tal como se detalla en la sección 4.2.

```
Función POSTPROCESAR(Teoría, Ejemplos) : Cjto. Reglas
EXTENSIÓNINICIAL(Teoría, Ejemplos) /* Primera fase de extensión */
SELECCIÓN(Teoría, Ejemplos)
EXTENSIÓNFINAL(Teoría, Ejemplos) /* Segunda fase de extensión */
si no hay regla por defecto entonces
    INFLADO(Teoría, Ejemplos) /* Un inflado final a las reglas */
fin si
```

```
Procedimiento EXTENSIÓNINICIAL(Teoría, Ejemplos)
EXTENSIÓN(Teoría, Ejemplos, SIN_INTERSECCIONES)
```

```
Procedimiento EXTENSIÓNFINAL(Teoría, Ejemplos)
EXTENSIÓN(Teoría, Ejemplos, CON_INTERSECCIONES)
```

4.2 Extensión de reglas

Una de las etapas del post-proceso aplicado por INNER consiste en tratar de extender algunas reglas en la dirección de otras de su misma clase que estén *suficientemente próximas* a ella. Se pretende así llegar a *pegar* reglas que seguramente han sido obtenidas en el mismo ciclo de generalización y que, por haber partido de instancias situadas en una misma región del espacio de atributos, han cubierto distintas partes de ésta cuando podía haberse cubierto con una sola regla un poco más extensa.

La extensión de reglas se aplica de dos maneras ligeramente diferentes: como se puede ver en el Algoritmo 4.1.1, antes de la llamada a la función SELECCIÓN se efectúa una extensión en la que no se van a permitir intersecciones con reglas de otras clases mientras que, después de la selección de reglas, cuando ya se dispone de lo que va a ser prácticamente el conjunto final de clasificadores, se efectúa una extensión de reglas en la que sí se permiten intersecciones con reglas de otras clases, asignando una prioridad relativa a las reglas involucradas en las intersecciones, tal como se describe en la sección 4.4. Esta última extensión conducirá en muchos casos a la obtención de reglas que se solapan o que quedan anidadas dentro de otras.

El mecanismo de extensión, cuyo pseudocódigo puede verse en el Algoritmo 4.2.1, comienza construyendo una lista que denominaremos *lista de pares*, en la que cada elemento recoge una pareja de reglas para las que sería posible tratar de efectuar una extensión. También se recoge la distancia que las separa, que se utilizará como criterio para decidir si se tratará de hacer la extensión o no; la máxima distancia para permitir la extensión actúa como umbral en la función CRITERIOINTENTAREXTENSIÓN utilizada en el Algoritmo 4.2.2. Además, la distancia también sirve para ordenar los pares e intentar la extensión de las reglas más próximas antes que la de las más alejadas.

Para cada elemento de la lista de pares, que denotaremos por $[R_i, R_j|D]$, se va a tratar de extender la regla R_i sobre la regla R_j usando la función EXTENDER, descrita en el Algoritmo 4.2.3. Esta función construye una nueva regla a partir de la regla inicial R_i que va a crecer hasta cubrir, quizá por completo, la regla R_j sobre la que se está extendiendo. Si la calidad de la nueva regla, medida en términos de nivel de impureza, no es inferior a la de la regla original, entonces se sustituye ésta por su versión extendida; si la regla extendida resulta ser de peor calidad se desechan las modificaciones y se retoma la regla original. También puede ocurrir que, a pesar de ser reglas muy próximas, no haya posibilidad de extensión por razones geométricas que se explicarán más adelante, en cuyo caso la función EXTENDER no hace nada con la regla R_i y notifica de esta circunstancia retornando el valor FALSO.

El algoritmo toma, pues, una primera decisión respecto a si se debe probar la extensión o no de una regla hacia otra; esta decisión se toma en base a la *distancia* entre las reglas y a la *geometría* de estas que, como se verá más adelante, no sólo sirve como criterio adicional para decidir hacer una extensión sino que también va a indicar cómo ha de hacerse. Si se decide probar la extensión, la segunda decisión,

relativa a si se admite la extensión o no, se toma en función del nivel de impureza de la regla extendida.

En el resto de la sección se van a detallar los elementos clave en el proceso de extensión, mostrando cómo se mide la distancia entre dos reglas y, si se decide extender una de ellas, cómo hacerlo.

4.2.1 \mathcal{R} -HEOM: La distancia entre reglas

Hay que destacar que la distancia entre reglas, calculada por la función DISTANCIA, no cumple la propiedad simétrica si las reglas contienen antecedentes simbólicos, es decir, la distancia de R_i a R_j no tiene por qué ser la misma que la de R_j a R_i . Por esta razón hay que considerar una posible extensión de R_i hacia R_j independientemente de la extensión de R_j hacia R_i . Cada elemento de la lista de pares antes mencionada será de la forma $[R_i, R_j|D_{ij}]$ y representará la posibilidad de extensión desde R_i hacia R_j al ser D_{ij} menor que el umbral requerido. Nótese, no obstante, que puede no existir un elemento $[R_j, R_i|D_{ji}]$ si D_{ji} no supera el umbral.

Para el cálculo de la distancia de una regla a otra sólo se tienen en cuenta los antecedentes presentes en la primera. La distancia desde una regla R_i a una R_j se calculará como:

$$\mathcal{R}\text{-HEOM}(R_i, R_j) = \sqrt{\sum_{a=1}^{m_i} d_{\mathcal{R}}(Ant_a^i, Ant_a^j)^2} \quad (4.1)$$

donde

- m_i es el número de antecedentes de la regla R_i .
- Ant_a^i es el antecedente que establece una condición sobre el atributo a -ésimo en la regla R_i mientras que Ant_a^j es su homólogo en la regla R_j . La regla R_j puede no tener ese antecedente.

En la ecuación (4.1) se hace uso de $d_{\mathcal{R}}$ para medir la distancia entre los antecedentes que establecen una condición sobre el atributo a -ésimo de las dos reglas R_i y R_j ; esta función se define como:

$$d_{\mathcal{R}}(Ant_a^i, Ant_a^j) = \begin{cases} 0 & \text{si } Ant_a^j \text{ es nulo} \\ D_{\mathcal{R}}(Ant_a^i, Ant_a^j) & \text{si } Ant_a^i \text{ es simbólico} \\ E_{\mathcal{R}}(Ant_a^i, Ant_a^j) & \text{si } Ant_a^i \text{ es continuo} \end{cases} \quad (4.2)$$

donde

$$D_{\mathcal{R}}(Ant_a^i, Ant_a^j) = \min_v \{ \mathcal{F}(T_a^i[v]) / T_a^j[v] = 0 \} \quad (4.3)$$

$$E_{\mathcal{R}}([x_1^i, x_2^i], [x_1^j, x_2^j]) = \begin{cases} 0 & \text{si } Ant_a^i \text{ y } Ant_a^j \text{ intersecan} \\ \frac{\min\{|x_1^i - x_2^j|, |x_2^i - x_1^j|\}}{\max_a - \min_a} & \text{en otro caso} \end{cases} \quad (4.4)$$

Algoritmo 4.2.1 Extensión de reglas. Para cada par de reglas de la misma clase se calcula la distancia que las separa y se construye una lista ordenada con aquellos pares de reglas que están suficientemente próximas como para intentar una extensión. Si la extensión se efectúa y una de las reglas queda completamente cubierta por la otra, se elimina. Además, cada vez que se extiende una regla hay que actualizar la lista de pares para mantener el orden ya que puede haber cambiado la distancia entre algunas reglas.

Procedimiento EXTENSIÓN(*Reglas, Ejemplos, Intersecciones?*)
para cada clase *C* **hacer**
 ListaPares = CONSTRUIRLISTAPARES(*C, Reglas*)
 $[R_i, R_j|D]$ = PRIMERELEMENTO(*ListaPares*)
 mientras $[R_i, R_j|D] \neq$ ULTIMOELEMENTO(*ListaPares*) **hacer**
 SeExtiende R_i = EXTENDER($R_i, R_j, Ejemplos, Intersecciones?$)
 SeElimina R_j = CONTENIDA(R_j, R_i)
 si *SeElimina* $R_j \vee SeExtiendeR_i$ **entonces**
 si *SeElimina* R_j **entonces**
 Reglas = *Reglas* $\setminus R_j$
 fin si
 /* Actualizar lista de pares */
 para cada elemento $[Rg_i, Rg_j|Dist]$ de *ListaPares* **hacer**
 si *SeElimina* $R_j \wedge (Rg_i = R_j \vee Rg_j = R_j)$ **entonces**
 /* Borrar elementos en los que aparezca la regla eliminada */
 ListaPares = *ListaPares* $\setminus [Rg_i, Rg_j|Dist]$
 si no si *SeExtiende* $R_i \wedge (Rg_i = R_i \vee Rg_j = R_i)$ **entonces**
 /* Actualizar distancia en elementos con la regla extendida */
 NuevaDist = DISTANCIA(Rg_i, Rg_j) /* Actualizar distancia */
 ListaPares = (*ListaPares* $\setminus [Rg_i, Rg_j|Dist]$) $\cup [Rg_i, Rg_j|NuevaDist]$
 fin si
 fin para
 si *SeExtiende* R_i **entonces**
 ORDENAR(*ListaPares*)
 $[R_i, R_j|D]$ = PRIMERELEMENTO(*ListaPares*)
 fin si
 fin mientras
 fin para

Algoritmo 4.2.2 Construcción de la lista de reglas para extender. Dada una clase C , esta función retorna una lista de elementos que contienen, cada uno, una regla R_i y una regla R_j , ambas de la clase C y la distancia D_{ij} desde la regla R_i a la regla R_j . Esta lista se va a utilizar para tratar de extender R_i sobre R_j por lo que en ella no deben figurar reglas separadas por una distancia superior a un determinado umbral, ni la regla por defecto, si es que existe.

Función CONSTRUIRLISTAPARES($C, Reglas$) : Lista de $[R_i, R_j|D_{ij}]$

```

para cada  $R_i \in Reglas$  hacer
  si CLASE( $R_i$ ) =  $C \wedge$  NUMANTECEDENTES( $R_i$ ) > 0 entonces
    para cada  $R_j \in Reglas$  hacer
      si CLASE( $R_j$ ) = CLASE( $R_i$ )  $\wedge$   $R_i \neq R_j \wedge$  NUMANTECEDENTES( $R_j$ ) > 0
        entonces
           $Dist =$  DISTANCIA( $R_i, R_j$ )
          si CRITERIOINTENTAREXTENSIÓN( $Dist$ ) entonces
             $ListaPares = ListaPares + [R_i, R_j|Dist]$ 
          fin si
        fin si
      fin para
    fin si
  fin para
  ORDENAR( $ListaPares$ ) /* Ordenar de menor a mayor distancia entre reglas */
retornar( $ListaPares$ )

```

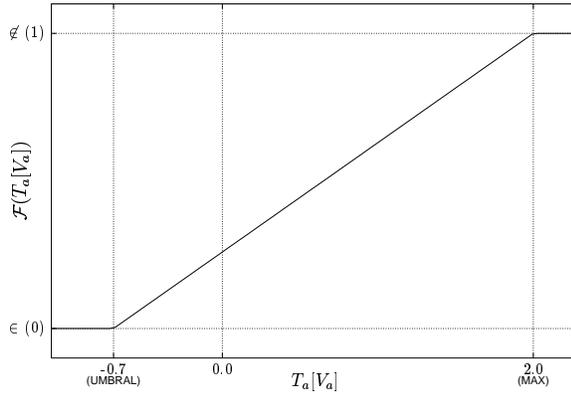


Figura 4.1: Función de pertenencia de un símbolo a un antecedente. Esta función sirve para medir lo lejos que está un valor simbólico de pertenecer a un antecedente.

Antecedentes simbólicos

Como se puede observar en la ecuación (4.3), la distancia entre un antecedente simbólico de la regla R_i y su homólogo de la regla R_j se calcula como el mínimo de los valores obtenidos al aplicar la función \mathcal{F} a los valores en aquellas entradas de su tabla de diferencias de los símbolos que *pertenecen* al antecedente de R_j . La función \mathcal{F} se denomina *función de pertenencia* y pretende indicar cuan lejos está un símbolo de ser incluido en el antecedente simbólico. Se define como $(1 - \mathcal{G})$, donde \mathcal{G} es el *grado de pertenencia* ya introducido en la página 35 al hablar del proceso de regularización de antecedentes simbólicos. La representación gráfica de esta función puede verse en la Figura 4.1.

Con el ánimo de aclarar un poco más el cálculo de la distancia usando \mathcal{R} -HEOM, vamos a poner un ejemplo: supongamos dos reglas de clase C con un único antecedente que establece una condición sobre el atributo *color*:

$$R_i : C \leftarrow color \in \{azul\}$$

$$R_j : C \leftarrow color \in \{rojo, verde\}$$

Las tablas de diferencias de estas reglas son las siguientes:

	COLOR		
	rojo	verde	azul
R_i	$-0.68 \Rightarrow \mathcal{F} = 0.007$	$0.24 \Rightarrow \mathcal{F} = 0.348$	$-0.95 \Rightarrow \mathcal{F} = \mathbf{0}$
R_j	$-0.98 \Rightarrow \mathcal{F} = \mathbf{0}$	$-0.82 \Rightarrow \mathcal{F} = \mathbf{0}$	$1.25 \Rightarrow \mathcal{F} = 0.722$

Para calcular $D_{\mathcal{R}}(\text{COLOR}_{R_i}, \text{COLOR}_{R_j})$, es decir, la distancia desde el antecedente *color* de R_i al de R_j , se toman los valores que resultan de aplicar la función \mathcal{F}

en aquellas entradas de R_i tales que, en R_j el valor de \mathcal{F} es 0. De todos ellos, nos quedaremos con el menor. En este caso, pues, la distancia desde el antecedente *color* de R_i al de R_j se calcula usando las entradas *rojo* y *verde* de R_i , y el resultado viene dado por $\min\{0.007, 0.348\}$, o sea, 0.007. Para finalizar, la distancia de la regla R_i a R_j es:

$$\mathcal{R}\text{-HEOM}(R_i, R_j) = \sqrt{0.007^2} = 0.007$$

Sin embargo,

$$\mathcal{R}\text{-HEOM}(R_j, R_i) = \sqrt{0.722^2} = 0.722$$

El razonamiento seguido para plantear de esta forma el cálculo de distancias entre antecedentes simbólicos tiene mucho que ver con el procedimiento de generalización descrito en el Capítulo 3. Se puede ver en el ejemplo anterior que, aunque el único valor contenido en el antecedente de R_i es *azul*, si se observa la entrada correspondiente al *rojo* en la tabla de diferencias, se aprecia que el mecanismo de generalización ha estado a punto de sugerir también la inclusión de ese color; podemos pensar que el rojo “casi” está incluido y, siendo así, cabe pensar que la distancia sobre la que habría que extender R_i para alcanzar a R_j sea “casi” cero. Sin embargo, al calcular la distancia desde R_j a R_i se observa que el único valor presente en R_i , el *azul*, ha sido claramente rechazado por la regla R_j , luego la distancia a recorrer para extender R_j sobre R_i ha de ser razonablemente mayor que la opuesta.

Antecedentes continuos

La distancia entre antecedentes continuos se calcula, tal como puede verse en la ecuación (4.4), como la distancia Euclídea normalizada entre los extremos más próximos de los intervalos que definen cada antecedente; si los intervalos de los antecedentes intersecan, entonces la distancia es 0. Cuando dos reglas tienen únicamente antecedentes continuos la distancia entre ellas cumple la propiedad simétrica.

4.2.2 Los antecedentes extensibles

Como ya se ha mencionado anteriormente, la distancia que separa dos reglas no es criterio suficiente para decidir si se debe o no extender una de ellas. Si analizamos desde un punto de vista geométrico distintas situaciones que se pueden presentar podemos encontrar casos como los que ilustra la Figura 4.2 donde, a pesar de que la distancia que separa las reglas es la misma, no en todos los casos es aconsejable la extensión.

Se necesita, pues, otro criterio adicional para decidir si se trata de extender una regla o no. Este criterio va a depender del *grado de coincidencia*, Γ , entre los antecedentes de las reglas, definido para antecedentes simbólicos y continuos en las

Algoritmo 4.2.3 Extensión de una regla sobre otra. El peso de este algoritmo recae en el cálculo de los atributos extensibles, que devuelve un conjunto de antecedentes de la regla R_i que pueden ser extendidos. Si el conjunto es vacío, no se va a practicar ningún tipo de extensión.

Función EXTENDER($R_i, R_j, Ejemplos, Intersecciones?$) : CIERTO | FALSO

$SeExiende =$ FALSO

$Extensibles =$ ANTECEDENTESEXTENSIBLES(R_i, R_j)

si $Extensibles \neq \emptyset$ **entonces**

$C =$ CLASE(R_i)

$R' =$ CREAMREGLA("C ← ")

para cada antecedente A de R_i **hacer**

si $A \in Extensibles$ **entonces**

$A' = A$ extendido sobre R_j

$R' = R' \oplus A'$ /* donde \oplus denota *añadir un antecedente**/

si no

$R' = R' \oplus A$

fin si

fin para

si $Intersecciones? \vee INTERSECC(R') = \emptyset$ **entonces**

si CALIDAD($R', Ejemplos$) \geq CALIDAD($R_i, Ejemplos$) **entonces**

$R_i = R'$

$SeExiende =$ CIERTO

fin si

fin si

fin si

retornar($SeExiende$)

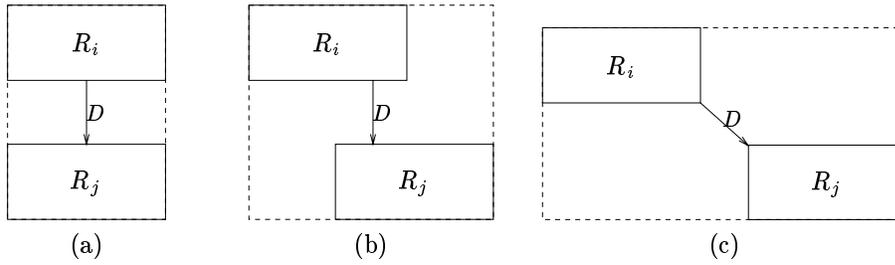


Figura 4.2: Al tratar de extender R_i sobre R_j puede suceder en algunos casos, como el (b) y el (c), que la regla tenga que crecer mucho, cubriendo gran parte del espacio a su alrededor. En el peor de los casos eso llevará a R_i a cubrir ejemplos de otras clases que pueden estar en sus proximidades y que seguramente le impidieron crecer por la zona sobre la que ahora pretende extenderse. En el mejor de los casos, si alrededor no hay ejemplos de ninguna clase, la regla extendida no estará adaptándose correctamente al espacio ocupado por los ejemplos de su clase. Por tanto, sólo se aconseja la extensión en el caso (a).

ecuaciones (4.5) y (4.6) respectivamente, como:

$$\Gamma(Ant_a^i, Ant_a^j) = \frac{\sum_v \mathcal{G}(T_a^i[v]) \cdot \mathcal{G}(T_a^j[v])}{\sum_v \mathcal{G}(T_a^i[v])} \quad (4.5)$$

$$\Gamma(Ant_a^i, Ant_a^j) = \begin{cases} \frac{\min\{x_2^i, x_2^j\} - \max\{x_1^i, x_1^j\}}{x_2^i - x_1^i} & \text{si } Ant_a^i \cap Ant_a^j \neq \emptyset \\ 0 & \text{si } Ant_a^i \cap Ant_a^j = \emptyset \end{cases} \quad (4.6)$$

El grado de coincidencia va a dictar cuales son los antecedentes que pueden ser extendidos en la regla R_i . En INNER las posibilidades de extensión se van a reducir a los dos casos mostrados en la Figura 4.3 que son:

- *Extender todos los antecedentes:* esto implica que la regla que está siendo extendida acaba cubriendo a la regla sobre la que se extiende. Este caso correspondería a una regla cuyos antecedentes son similares a los de la regla sobre la que va a extenderse. Es muy posible que se llegue a esta situación tras la generalización en distintos ciclos de instancias tomadas de una misma región.
- *Extender un solo antecedente:* equivale a *incrustar* una regla en otra. En ocasiones la regla sobre la que se extiende también puede quedar completamente cubierta. Esta situación puede deberse a la propia geometría del problema que impone la necesidad de colocar dos reglas con esa forma.

El criterio seguido para la elección de antecedentes a extender es:

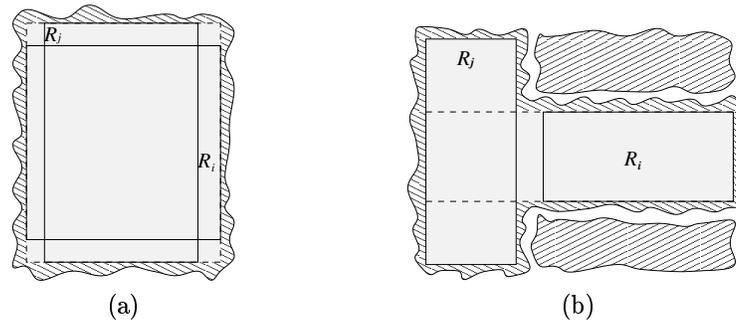


Figura 4.3: Selección de antecedentes a extender. En ocasiones la extensión se aplicará a todos los antecedentes, para acabar cubriendo a la segunda regla. Otras veces una regla puede extenderse a lo largo de un sólo eje o atributo para llegar a *incrustarse* en otra de su misma clase muy cercana.

Si el grado de coincidencia de todos los antecedentes de R_i con sus homólogos en R_j es alto, se extenderán todos los antecedentes de R_i . Si sólo uno de los antecedentes de R_i tiene un alto grado de coincidencia se extenderá sólo ese antecedente. En otro caso no habrá extensión de R_i , independientemente de lo próxima que pueda estar a R_j .

Se considerará alto aquel grado de coincidencia que supere un determinado umbral, establecido por defecto en el valor 0.9. El Algoritmo 4.2.4 correspondiente a la función ANTECEDENTESEXTENSIBLES implementa el criterio expuesto.

Uno de los elementos más importantes del algoritmo de extensión es, como se puede apreciar, la función EXTENDER. Esta función no sólo implementa el criterio adicional a la distancia que permite decidir si se va a extender o no una regla sobre otra, sino que además efectúa la extensión y comprueba si ésta es aceptable en función del nivel de impureza de la versión extendida de la regla.

Al extender una regla pueden producirse intersecciones con reglas de otras clases, en cuyo caso, el argumento adicional *Intersecciones?* de la función EXTENDER será decisivo para permitir o no la extensión. Concretamente, antes del proceso de selección de reglas no se permitirá que la extensión de una regla dé lugar a intersecciones o solapamientos con reglas de otras clases.

Esta primera fase de extensión, menos agresiva que la que se hace tras el proceso de selección, pretende agrupar o compactar aquellas reglas que están en regiones del espacio de atributos claramente caracterizadas por pertenecer a una determinada clase. En estas regiones seguramente hay varias reglas porque fueron generalizadas en el mismo ciclo iterativo del algoritmo general, de forma que el crecimiento de unas estorbaba al de otras y ninguna de ellas llegó a cubrir correctamente la región. La

Algoritmo 4.2.4 Antecedentes susceptibles de ser extendidos. Esta función retorna un conjunto de antecedentes de la regla R_i que, eventualmente, puede ser vacío. En otro caso el conjunto tendrá todos los antecedentes de R_i o sólo uno de los antecedentes, aquel cuyo grado de coincidencia no supere un umbral determinado.

Función ANTECEDENTESEXTENSIBLES(R_i, R_j) : Cjto. Antecedentes

$Extensibles = \emptyset$

$YaEncontréUno = \text{FALSO}$

para cada antecedente A_i de R_i **hacer**

$A_j = \text{homólogo de } A_i \text{ en } R_j$

si existe A_j **entonces**

$Coincidencia = \text{GRADOCOINCIDENCIA}(A_i, A_j)$

si $Coincidencia < \text{UMBRALCOINCIDENCIA}$ **entonces**

si $YaEncontréUno$ **entonces**

retornar(\emptyset) /* más de uno poco coincidente \Rightarrow ninguno extensible */

si no

$Extensibles = \{A_i\}$

$YaEncontréUno = \text{CIERTO}$

fin si

si no

si $\text{NO}(YaEncontréUno)$ **entonces**

$Extensibles = Extensibles \cup \{A_i\}$

fin si

fin si

fin si

fin para

retornar($Extensibles$)

extensión previa al proceso de selección trata de corregir estas situaciones, haciendo más explícita la clasificación de estas regiones y aligerando la carga de trabajo para la selección que se va a aplicar a continuación, ya que, generalmente, quedarán menos reglas en juego.

En la segunda fase de la extensión, una vez aplicada la selección, se procede de una forma más atrevida, permitiendo que una regla se pueda solapar con reglas de otras clases.

4.3 Selección de reglas

Tras obtener un conjunto de reglas que prácticamente puede considerarse como definitivo, INNER intenta simplificar el número total de reglas aplicando un proceso de *selección* o poda que va a tratar de quedarse con un subconjunto de las reglas obtenidas hasta el momento sin perder precisión sobre el conjunto de entrenamiento. Al igual que el proceso de cualificación descrito en la sección 3.5, la selección también se ha tomado del sistema ABANICO, si bien la implementación en INNER es una versión simplificada en algunos aspectos¹ con respecto al mecanismo original descrito en [Ranilla, 98], y ampliada en otros como la asignación explícita de prioridades a las reglas que se solapan.

La poda que efectúa el proceso de selección no hace una exploración exhaustiva de todos los posibles subconjuntos de reglas, ya que el coste computacional es muy elevado. En su lugar se utiliza un heurístico basado, una vez más, en el nivel de impureza.

El mecanismo de selección, cuyo pseudocódigo puede verse en el Algoritmo 4.3.1, consta de cinco fases que se describen en el resto de esta sección.

Algoritmo 4.3.1 Selección de reglas.

Procedimiento SELECCIÓN(*Teoría, Ejemplos*)
 ELIMINARRUIDOSAS(*Teoría, Ejemplos*)
 BUSCARMEJORNIVELIMPUREZA(*Teoría, Ejemplos*)
 ELIMINARREDUNDANTES(*Teoría, Ejemplos*)
 ASIGNARPRIORIDADES(*Teoría, Ejemplos*)
 INCLUIRREGLADEFECTO(*Teoría, Ejemplos*)

¹El mecanismo de selección de ABANICO puede operar en *modo calidad* y en *modo ponderado*; el primero garantiza que el conjunto de reglas seleccionado tiene un rendimiento sobre el conjunto de entrenamiento igual o mejor que antes de la poda, mientras que el segundo efectúa una poda más agresiva, reduciendo generalmente el número de reglas resultantes con respecto al modo calidad, pero perdiendo también algo de precisión. En INNER sólo se implementa el modo *calidad*.

Eliminación de reglas ruidosas

La primera etapa de la selección de reglas está relacionada con el tratamiento del ruido. En esta fase se pretende detectar si un problema es ruidoso y, en caso de que así sea, eliminar las reglas ruidosas.

Para decidir si un problema se considera ruidoso se va a tener en cuenta el número de *fallos en aplicación nítida*² de las reglas, denominados *errores internos*, así como su nivel de impureza. Se considera que hay ruido si

“... los errores internos de las reglas con niveles mejores que el de la peor regla de entre las mejores de cada clase son superiores a un cierto umbral. Se admiten tantos fallos como clases tenga el problema.” [Ranilla, 98]

Como se puede ver en el Algoritmo 4.3.2, para cada clase se busca la regla de mejor calidad, siempre en términos de nivel de impureza; de entre las mejores así obtenidas, una por cada clase, se toma como umbral o valor de corte el nivel de impureza de la peor de ellas y, volviendo al conjunto completo de reglas, se contabilizan los fallos en aplicación nítida de todas aquellas reglas de calidad igual o superior (nivel de impureza igual o inferior). Si la suma de fallos supera el número de clases presentes en el problema, entonces se considera que hay ruido.

Falta precisar ahora cuáles serán las reglas consideradas como ruidosas. Teniendo en cuenta que el ruido es un fenómeno aleatorio que afecta a casos puntuales dentro del conjunto de entrenamiento, vamos a considerar como ruidosas aquellas reglas que acierten en pocos casos. Precizando un poco más, se consideran ruidosas

“... las reglas cuyo número de aciertos sea menor o igual que un porcentaje del número de ejemplos de entrenamiento de la clase menos abundante.” [Ranilla, 98]

Búsqueda del nivel de impureza con mejor rendimiento

Esta etapa pretende encontrar una primera aproximación al subconjunto de reglas con mejor rendimiento sobre los ejemplos de entrenamiento. El planteamiento que se hace en esta fase es el de evaluar la precisión de distintos subconjuntos para quedarnos con el que menos errores cometa.

Para garantizar una solución óptima es necesario evaluar $2^K - 1$ subconjuntos distintos de reglas, donde K es el número de reglas de que se dispone; esto supone un coste importante³ desde el punto de vista computacional, así que sólo se van a considerar algunas combinaciones de reglas. El heurístico utilizado para reducir el espacio de búsqueda se basa, una vez más, en el nivel de impureza.

²Ejemplos de una clase cubiertos por una regla de otra clase.

³Este mismo problema ya se había planteado en el proceso de cualificación presentado en la sección 3.5 para decidir qué antecedentes había que eliminar en cada regla.

Algoritmo 4.3.2 Eliminación de reglas ruidosas. Para determinar si el problema tiene ruido se calculan los fallos de clasificación de aquellas reglas cuya calidad es superior o igual a la de un determinado valor de *corte*. Si el problema se considera ruidoso, entonces se eliminarán las reglas que tengan un número de aciertos (en aplicación nítida) menor o igual al 5% del número de ejemplos de la clase minoritaria.

Procedimiento ELIMINARRUIDOSAS(*Teoría, Ejemplos*)
Corte = PEORMEJORCALIDADEN(*Teoría, Ejemplos*)
ErroresInternos = 0
para cada regla $R \in Teoría$ **hacer**
 si CALIDAD($R, Ejemplos$) no es inferior al *Corte* **entonces**
 ErroresInternos = *ErroresInternos* + FALLOS($R, Ejemplos$)
 fin si
fin para
HayRuido = (*ErroresInternos* > número de clases del problema)
si *HayRuido* **entonces**
 F_R = FACTORRUIDO /* por defecto 5% */
 N_m = número de ejemplos de la clase minoritaria
 $UmbralAciertos$ = $F_R \cdot N_m$
 SinRuidosas = \emptyset
 para cada regla $R \in Teoría$ **hacer**
 si ACIERTOS($R, Ejemplos$) > *UmbralAciertos* **entonces**
 /* no es ruidosa */
 SinRuidosas = *SinRuidosas* \cup { R }
 fin si
 fin para
 Teoría = *SinRuidosas*
fin si

Algoritmo 4.3.3 Búsqueda del nivel de impureza con mejor rendimiento. Inicialmente se construye una lista con los niveles de impureza de las reglas obtenidas hasta el momento. Habitualmente esta lista tendrá menos elementos que reglas hay en el conjunto *Teoría* ya que, si hay más de una regla con el mismo nivel de impureza, éste solo se introduce una vez en la lista.

Procedimiento BUSCARMEJORNIVELIMPUREZA(*Teoría*, *Ejemplos*)

ListaNiveles = lista de los niveles de impureza de las reglas de *Teoría*

Conjunto = *Teoría*

Valoración = EVALUAR(*Conjunto*, *Ejemplos*)

para cada nivel de impureza $N \in ListaNiveles$ ordenada de peor a mejor **hacer**

CjtoCandidato = \emptyset

para cada $R_i \in Conjunto$ **hacer**

si CALIDAD(R_i) es superior a N **entonces**

CjtoCandidato = *CjtoCandidato* \cup R_i

fin si

fin para

ValActual = EVALUAR(*CjtoCandidato*, *Ejemplos*)

si *ValActual* superior a *Valoración* **entonces**

Valoración = *ValActual*

Conjunto = *CjtoCandidato*

fin si

fin para

Teoría = *Conjunto*

Se persigue establecer un corte o umbral en un nivel de impureza y eliminar aquellas reglas de peor calidad que ese umbral; para elegir donde colocar el corte se ordenarán las reglas según su nivel de impureza y se establecerá como valor de corte el peor de ellos; ese corte se irá desplazando de forma iterativa por los distintos niveles de impureza presentes en el conjunto de reglas tal como muestra la Figura 4.4. En cada iteración se evaluará el rendimiento de las reglas que superan el corte. Al final del proceso nos quedaremos con el subconjunto que haya tenido mejor rendimiento. El Algoritmo 4.3.3 muestra como se efectúa el proceso.

Eliminación de reglas redundantes

Al conjunto de reglas obtenido en la etapa anterior se aplica un proceso de refinamiento eliminando reglas que resulten redundantes. En esta fase también cabe la posibilidad de hacer una exploración exhaustiva evaluando todos los posibles subconjuntos; sin embargo, la solución propuesta en ABANICO, y adoptada también en INNER, utiliza la medida de nivel de impureza de las reglas para reducir el conjunto de posibilidades a evaluar.

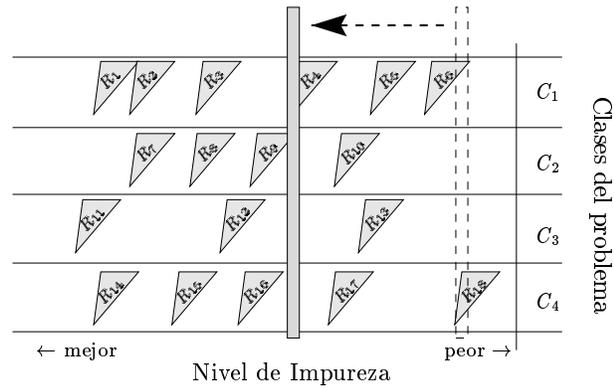


Figura 4.4: Buscando el mejor corte por nivel de impureza. En cada iteración se va *subiendo el listón* y se evalúa el rendimiento de las reglas que lo superan. Al final del proceso nos quedaremos con el subconjunto que haya dado el mejor rendimiento.

Este proceso ordena las reglas de menor a mayor calidad y recorre el conjunto así ordenado evaluando, para cada regla, la precisión que se consigue sobre los ejemplos de entrenamiento si ésta es eliminada; si la precisión *no disminuye*, entonces se elimina la regla y se prosigue sin ella; en caso contrario se vuelve a añadir la regla al conjunto y se prueba con la siguiente, tal como puede verse en el Algoritmo 4.3.4.

Algoritmo 4.3.4 Eliminación de reglas redundantes.

Procedimiento ELIMINARREDUNDANTES(*Teoría, Ejemplos*)

Valoración = EVALUAR(*Teoría, Ejemplos*)

ORDENAR(*Teoría*) /* Las reglas se ordenan de peor a mejor calidad */

para cada regla $R \in Teoría$ **hacer**

$Teoría = Teoría \setminus R$

$ValNueva = EVALUAR(Teoría, Ejemplos)$

si $ValNueva$ no es superior a $Valoración$ **entonces**

$Teoría = Teoría \cup R$

fin si

fin para

Asignación de prioridades explícitas

Una vez eliminadas las reglas redundantes, el sistema original de selección utilizado en ABANICO comprueba si es aconsejable o no introducir una regla por defecto.

Sin embargo, INNER añade antes un paso que consiste en asignar de forma explícita valores de prioridad a las reglas que se solapan. Esta etapa va a permitir a la fase siguiente decidir con facilidad qué reglas son eliminables y cuales no en caso de añadir una regla por defecto.

El procedimiento de asignación de prioridades comprueba la calidad de la región de intersección para cada clase de las reglas implicadas, asignando mayor prioridad a aquella cuya clase dé mejor resultado. Evidentemente, sólo tiene sentido hablar de prioridades entre reglas de distintas clases, ya que en intersecciones entre reglas de una misma clase no importa cual de ellas se aplique, el resultado será el mismo.

Aunque las prioridades se asignan justo antes de comprobar si se añade una regla por defecto, la descripción detallada del procedimiento y su justificación no se va a dar ahora, sino que, por conveniencia, se explicará en la sección 4.4, al hablar de la segunda fase de extensión de reglas, en la que adquiere especial relevancia el tratamiento de intersecciones; baste aquí decir que aquellas reglas que se solapan con otras de clase diferente van a tener una prioridad explícita a partir de este momento, que servirá para decidir cuál se aplica en la intersección.

Inclusión de regla por defecto

Para reducir aún más el conjunto de reglas es posible sustituir algunas (o quizá todas) las reglas de una clase por una sola regla que actuará como *regla por defecto*, y que se usará para clasificar los ejemplos que no estén cubiertos por ninguna otra regla. En este caso las reglas clasificarán a lo sumo los ejemplos que cubren explícitamente, dejando que la regla por defecto clasifique el resto. Por esta razón, incluir una regla por defecto es una tarea un tanto delicada ya que los ejemplos descubiertos suelen pertenecer a más de una clase; esto sugiere que la clase de la regla por defecto debe ser la que tenga más ejemplos descubiertos en el conjunto de entrenamiento. En cualquier caso, si la precisión en la clasificación se resiente, no se incluirá regla por defecto.

Al igual que en ABANICO, la inclusión de una regla por defecto sólo se va a considerar en problemas en los que todos los atributos sean de tipo simbólico.

Al incluir la regla por defecto para una clase, algunas de las reglas de esa clase pueden ser eliminadas, pues quedan contenidas en la regla por defecto. Sin embargo, no todas son eliminables, en particular aquellas que intersecan con reglas menos prioritarias de otras clases no deben ser eliminadas. Para seleccionar con eficiencia las reglas eliminables es necesario haber establecido previamente los valores de prioridad entre reglas de forma explícita, tal como se comentaba en la etapa anterior. Cabe destacar también que, al igual que en ABANICO, la inclusión de una regla por defecto sólo se va a considerar en problemas en los que todos los atributos sean de tipo simbólico.

La Figura 4.5 muestra un ejemplo con reglas que se pueden eliminar y otras que no se deben eliminar, si se incluye la regla por defecto.

Algoritmo 4.3.5 Inclusión de regla por defecto.

```

Procedimiento INCLUIRREGLADEFECTO(Teoría, Ejemplos)
si todos los atributos del problema son simbólicos entonces
  Descubiertos = Ejemplos \ COBERTURA(Teoría, Ejemplos)
  si Descubiertos ≠ ∅ entonces
    Valoración = EVALUAR(Teoría, Ejemplos)
    C = CLASEMÁSABUNDANTEEN(Descubiertos)
    Rdef = CONSTRUIRREGLA("C ← ")
    Eliminables = CALCULARELIMINABLES(Teoría)
    Teoría = (Teoría \ Eliminables) ∪ {Rdef}
    ValNueva = EVALUAR(Teoría, Ejemplos)
    si ValNueva no es superior a Valoración entonces
      /* no se incluye la regla por defecto, se recupera el cjto. original */
      Teoría = (Teoría ∪ Eliminables) \ Rdef
    fin si
  fin si
fin si

```

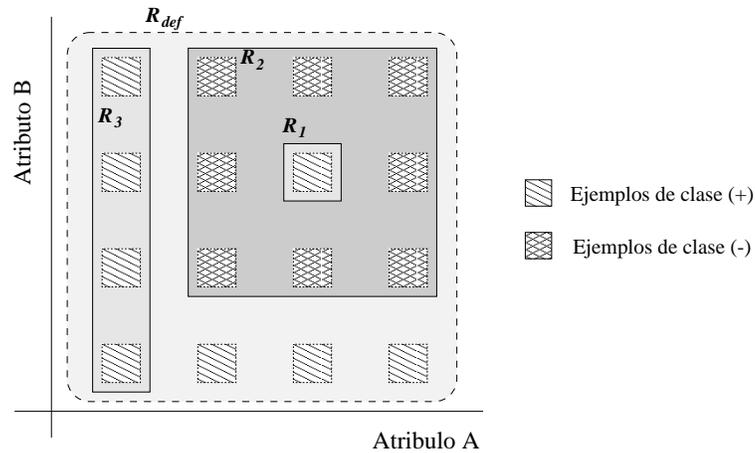


Figura 4.5: Reglas eliminables por inclusión de regla por defecto. En la situación representada tenemos dos reglas R_1 y R_3 de clase (+) y una regla R_2 de clase (-). Puesto que quedan ejemplos de clase (+) descubiertos se plantea la posibilidad de incluir una regla por defecto, R_{def} , de clase (+); en caso de incluirla puede verse claramente que podemos prescindir de la regla R_3 pero no de la regla R_1 , ya que sirve para clasificar las excepciones dentro del área cubierta por una regla de otra clase, la R_2 .

4.4 Extensión con solapamientos

Tras haber seleccionado el conjunto final de reglas se aplica un nuevo proceso de extensión en el que se va a permitir que las reglas puedan crecer y solaparse con reglas de otras clases. Cuando se producen solapamientos INNER asigna de forma explícita valores de prioridad a las reglas implicadas, estableciendo un orden entre ellas que coloca a unas *encima* de otras, siendo más prioritaria una regla cuanto más arriba esté; cuando haya que clasificar un ejemplo situado en la intersección se aplicará la regla de mayor prioridad.

La asignación de prioridades depende únicamente de la calidad de la zona de intersección que se medirá, como siempre, en términos de nivel de impureza, manteniendo la coherencia en todo el sistema INNER. Así, para dos reglas que se solapen, R_i y R_j , de clases (+) y (-) respectivamente, se calculará el nivel de impureza de la intersección suponiendo que va a ser clasificada por la regla R_i ; luego se calculará el nivel de impureza suponiendo que va a ser clasificada por la regla R_j . El menor nivel de impureza indica mayor calidad en la clasificación y por tanto cuál debe ser la regla más prioritaria. La Figura 4.6 ilustra un ejemplo en el que se plantea la situación descrita.

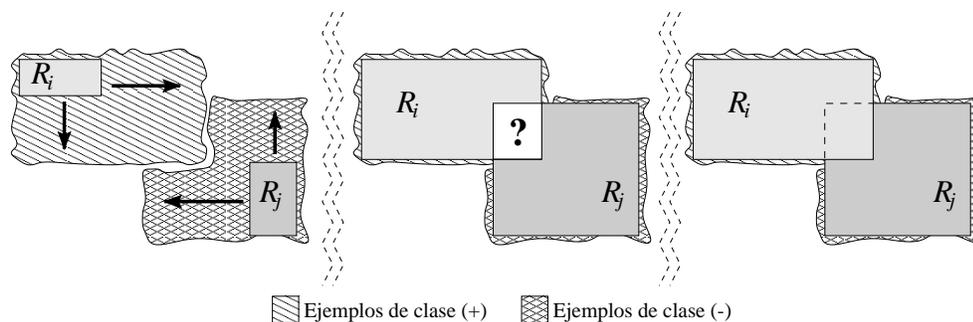


Figura 4.6: Asignación explícita de prioridad. Cuando dos reglas de clases diferentes se solapan, la zona de intersección es susceptible de ser clasificada por ambas. En este ejemplo y a la vista de la distribución del conjunto de entrenamiento, es evidente que debe la regla R_i la de mayor prioridad. El nivel de impureza así lo detecta y asigna correctamente las prioridades.

Cuando la intersección involucra varias reglas se procede de la misma manera, con el matiz de que se asignan las prioridades por orden de calidad comenzando por la mejor regla, la de menor nivel de impureza.

Para evitar situaciones que, en ocasiones, podrían resultar muy enrevesadas, el mecanismo de asignación de prioridades aplica una *simplificación* que, como puede

verse en el Algoritmo 4.4.1, consiste básicamente en asignar la prioridad a una regla sin necesidad de usar el procedimiento general, sino teniendo en cuenta la propiedad transitiva de la prioridad entre reglas. Así,

Una regla será más prioritaria que otra con la que interseca si ya es transitivamente más prioritaria. De la misma forma, una regla será menos prioritaria que otra con la que interseca si ya es transitivamente menos prioritaria.

donde una regla R_i es *transitivamente más prioritaria* con respecto a otra R_j si:

- a) interseca con R_j y es más prioritaria que ella, según el mecanismo general de asignación de prioridades.
- b) no interseca con R_j pero interseca con otra regla R_k que es transitivamente más prioritaria que R_j .

Esta simplificación tiene como consecuencias más destacables:

- *No se replantean las prioridades entre reglas cuyas intersecciones ya han sido resueltas previamente.* Así, si dos reglas se solapan y una de ellas, digamos la menos prioritaria, se extiende aumentando la región de intersección, no se reconsidera la posibilidad de asignar distintas prioridades, y la regla extendida seguirá siendo la menos prioritaria.
- *No se pueden producir situaciones de “caja de cartón”.* La Figura 4.7 muestra un ejemplo típico de solapamiento de reglas que INNER **no** permite.

Como consecuencia de la asignación de prioridades, el criterio de aplicación de reglas para clasificar un ejemplo queda como sigue:

Un ejemplo será clasificado por la regla más cercana. Si hay más de una se descartan todas aquellas que sean transitivamente menos prioritarias que otras. Si aún queda más de una regla aplicable se escogerá la más específica; en caso de empate aquella de mayor calidad y si persiste el empate, una al azar

Cabe mencionar aquí la exposición que se hace en [Wettschereck, Dietterich, 95] en contra del solapamiento de reglas en el sistema NGE [Salzberg, 90], donde se concluye que el solapamiento de reglas conduce al sistema a obtener peores resultados. Esto se debe al criterio arbitrario de selección de la regla a aplicar que utiliza NGE, que se decanta por la que menos área cubre para clasificar la zona de intersección. En este sentido INNER se asemeja más al sistema RISE [Domingos, 96] ya que utiliza un mecanismo de desempate más eficaz para elegir la regla a aplicar en los solapamientos, seleccionando la que mejor calidad ofrece. Esta calidad se mide en INNER en términos de nivel de impureza, mientras que RISE utiliza la estimación de Laplace, cuya

Algoritmo 4.4.1 Asignación de prioridades en las intersecciones. Para cada regla se calculan las intersecciones con reglas de otras clases y, una por una, se resuelven. El procedimiento RESOLVERINTERSECCIONES anota en la regla R su prioridad con respecto a cada regla con la que interseca. Esa prioridad se establece comprobando si ya existía una relación de prioridad transitiva entre ellas, en cuyo caso no se recalcula nada, simplemente se anota de forma explícita la prioridad; en otro caso se asigna mayor prioridad a la regla que clasifique la zona de intersección con mejor calidad.

Procedimiento ASIGNARPRIORIDADES(*Reglas*, *Ejemplos*)

ORDENAR(*Reglas*) /* de mejor a peor calidad */

para cada regla $R \in$ *Reglas* **hacer**

Intersecadas = REGLASINTERSECADAS(R , *Reglas*)

si *Intersecadas* $\neq \emptyset$ **entonces**

 RESOLVERINTERSECCIONES(R , *Intersecadas*, *Ejemplos*)

fin si

fin para

Procedimiento RESOLVERINTERSECCIONES(R , *Intersecadas*, *Ejemplos*)

ORDENAR(*Intersecadas*) /* de mejor a peor calidad */

para cada regla $R' \in$ *Intersecadas* **hacer**

 /* $R \prec R'$ es CIERTO $\Leftrightarrow R$ es transitivamente menos prioritaria que R' */

si $R \prec R'$ **entonces**

 ANOTARMENOSPRIORIDAD(R , R') /* R será menos prioritaria que R' */

si no si $R' \prec R$ **entonces**

 ANOTARMASPRIORIDAD(R , R') /* R será más prioritaria que R' */

si no

 /* Mecanismo general de asignación de prioridades */

$C =$ CLASE(R)

 /* Suponemos que $R \cap R'$ va a ser de clase C */

$Calidad_1 =$ CALIDADDEINTERSECCIÓN(R , R' , C , *Ejemplos*)

$C' =$ CLASE(R')

 /* y ahora suponemos que $R \cap R'$ va a ser de clase C' */

$Calidad_2 =$ CALIDADDEINTERSECCIÓN(R , R' , C' , *Ejemplos*)

si $Calidad_2$ es superior a $Calidad_1$ **entonces**

 ANOTARMENOSPRIORIDAD(R , R') /* R será menos prioritaria que R' */

si no

 ANOTARMASPRIORIDAD(R , R') /* R será más prioritaria que R' */

fin si

fin si

fin para

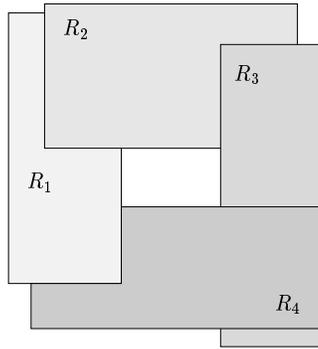


Figura 4.7: Situación de “caja de cartón”. La simplificación que hace uso de la transitividad de la prioridad de reglas impide que se pueda llegar a esta situación.

expresión se corresponde con la ecuación (4.7), seleccionando la regla que maximice su valor.

$$LAP(r) = \frac{p + 1}{p + n + c} \quad (4.7)$$

donde

- p es el número de ejemplos cubiertos correctamente.
- n es el número total de ejemplos cubiertos.
- c es el número de clases del problema.

El nivel de impureza efectivo

En cuanto una regla se solapa con otras y tiene una prioridad explícita cabe la posibilidad de refinar la medida utilizada para evaluar su calidad. En todo momento se ha utilizado para este fin el nivel de impureza [Ranilla, Mones, Bahamonde, 98] descrito de forma resumida en la sección 3.4.1. Sin embargo, ahora se puede precisar más ya que, para cualquier regla R_i sabemos que los ejemplos contenidos en las intersecciones con reglas de mayor prioridad no van a ser clasificados por R_i , así que podemos prescindir de ellos para calcular su calidad. Introducimos así el *nivel de impureza efectivo*, que se calcula como el nivel de impureza clásico pero prescindiendo de los ejemplos cubiertos por reglas más prioritarias.

Aunque es cierto que las intersecciones entre reglas se pueden producir desde las primeras etapas del proceso de inducción, esta medida de calidad no se utiliza hasta el momento en que se asignan prioridades de forma explícita, en los últimos pasos del proceso de selección. Utilizar el nivel de impureza efectivo antes sería arriesgado,

pues se estaría evaluando la calidad de una regla en función de otras reglas con las que interseca que podrían desaparecer en las fases de poda posteriores; la calidad así calculada no correspondería con la de la regla que al final se obtiene. El nivel de impureza es un estimador de calidad más pesimista que el nivel de impureza efectivo, ya que supone que la regla clasifica a todos los ejemplos que cubre, aunque sean de otras clases.

El uso del nivel de impureza efectivo va a permitir a INNER efectuar extensiones con solapamientos que el nivel de impureza no permitiría, como puede verse en la Figura 4.8.

4.5 Inflado final de reglas

El último proceso que se aplica al conjunto de reglas inducido pretende hacer un último y pequeño ajuste utilizando el mismo procedimiento de inflado que se usa para generalizar inicialmente las instancias y que ha sido detallado en el Capítulo 3. La única diferencia es que, en este momento la calidad de las reglas no se mide en términos de nivel de impureza sino de impureza efectiva, puesto que ya se ha establecido la prioridad entre las reglas de distintas clases que se solapan.

Este proceso de inflado final va a permitir que, en ocasiones, algunas reglas puedan llegar a cubrir ejemplos que no fueron cubiertos en el proceso de generalización, porque el nivel de impureza no lo aconsejaba, ni en el proceso de extensión de reglas, por no estar a medio camino entre dos reglas que se pudiesen extender. Una situación como esta bien podría ser la que ilustra la Figura 4.9, donde se aprecia que este último ajuste puede aumentar la precisión en la clasificación y hacer que las reglas sean más explícitas.

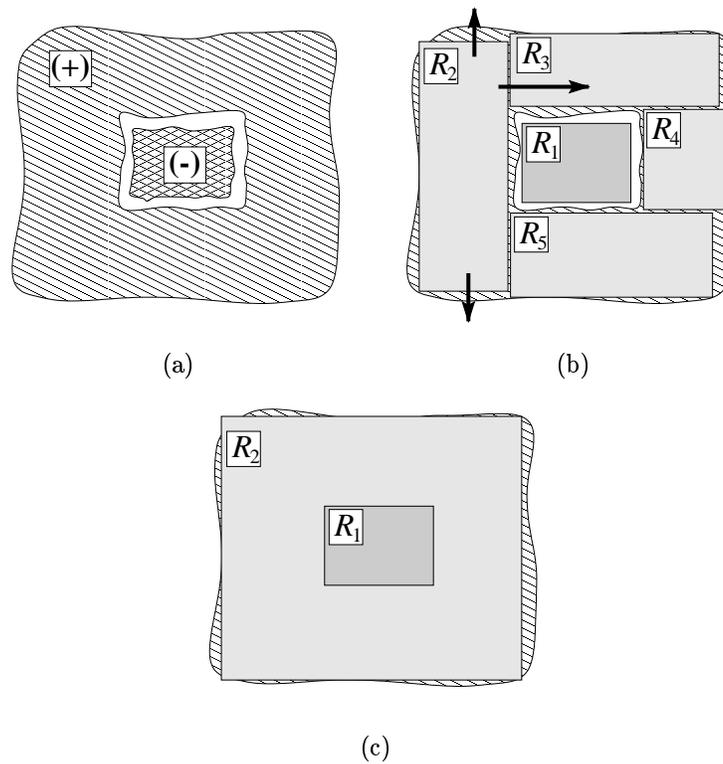


Figura 4.8: Extensiones con solapamiento. En un problema artificial donde los ejemplos de clase (+) y (-) se distribuyen como se muestra en la figura (a) es relativamente fácil llegar a obtener las reglas que se muestran en (b) tras los procesos de generalización y selección. En esta situación, el nivel de impureza efectivo permitirá extender las reglas de la clase (+) *por debajo* de R_1 . De esta forma la clase (+) se podrá quedar sólo con R_2 ya que, a pesar de cubrir ejemplos de clase (-), no los va a clasificar por estar en la intersección con otra regla más prioritaria, R_1 . Se obtienen así las dos reglas que pueden verse en la figura (c), que clasifican con la misma precisión que las cinco de partida.

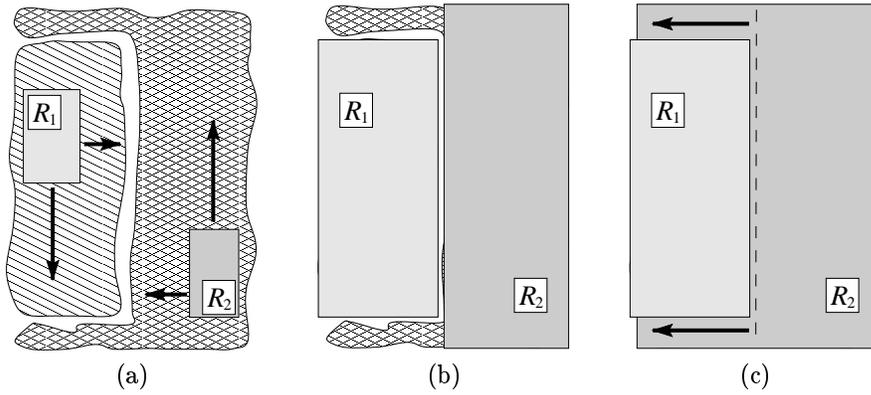


Figura 4.9: Inflado final con prioridad explícita en los solapamientos. La figura (a) corresponde a un problema en el que se generalizan dos reglas, una para cada clase, hasta obtener los resultados mostrados en (b). La regla R_2 no cubre todos los ejemplos de su clase porque eso implicaría cubrir también ejemplos de otra clase, así que el nivel de impureza desaconseja, durante la generalización, la expansión hacia el extremo izquierdo de la figura. En el inflado final, sin embargo, la regla puede crecer, recibiendo una prioridad inferior a R_1 como cabe esperar.

Capítulo 5

Resultados experimentales

5.1 Introducción

En este capítulo se presentan los resultados experimentales obtenidos con el sistema INNER ofreciendo una comparativa con los resultados obtenidos con el sistema RISE y el sistema C4.5 en la versión que genera reglas de clasificación (C4.5-rules), no sólo en cuanto a precisión sino también en cuanto a tamaño del conjunto de reglas aprendido.

En primer lugar se presenta y justifica el uso de un determinado grupo de problemas, con una pequeña descripción de cada uno de ellos. También se incluye en la Tabla 5.1 un resumen con las características más destacables de cada problema, tomada de [Holte, 93]. A continuación se comparan los resultados obtenidos en los experimentos con INNER, RISE [Domingos, 96] y C4.5 [Quinlan, 93a]. Se efectúan las comparaciones con estos dos sistemas por dos razones fundamentales: la similitud en la filosofía de RISE y la de INNER, teniendo en cuenta que RISE es un sistema reciente con una precisión muy elevada, y la bien merecida fama del ya clásico C4.5 de Quinlan.

5.2 Problemas usados en las pruebas

El grupo de problemas que se ha utilizado como banco de pruebas para INNER está compuesto por 19 conjuntos de ejemplos: 16 se corresponden con los *problemas de Holte* [Holte, 93] y los otros 3 son los *problemas del monje*, más conocidos en la literatura anglosajona por *the Monk's problems* [Wnek et al., 90].

Los problemas de Holte son un grupo de problemas muy conocidos y ampliamente utilizados en aprendizaje automático; Holte los ha utilizado en [Holte, 93] para comparar los resultados que obtuvo con sus sistema 1R frente a C4.5 y justificar que los resultados que se obtendrían en problemas del mundo real serían similares.

Holte afirma que un conjunto de ejemplos típicamente utilizado para labores de aprendizaje automático representa fielmente a un problema del mundo real si cumple dos condiciones:

1. Debe haber sido tomado de un dominio del mundo real y no haber sido construido artificialmente. Todos los problemas de Holte satisfacen este requisito.
2. Los ejemplos contenidos en el conjunto y los atributos usados para describirlos deben ser los que, de forma natural, se utilizan en el dominio del mundo real. Esto quiere decir que los problemas no deben ser *rediseñados* para hacer más fácil el proceso de aprendizaje. El único que no cumple esta condición es el problema denominado CH que, según se indica en [Holte, 93], representa finales de partidas de ajedrez de forma especialmente adecuada para el sistema ID3 de Quinlan y, como consecuencia de ello, para el C4.5.

Teniendo en cuenta estas condiciones así como los procesos de construcción de los conjuntos, Holte asegura que, aunque no representan todo el rango de problemas que se pueden encontrar en el mundo real, pueden considerarse como representativos de la clase de problemas que a menudo se plantean. Por esta razón este grupo de problemas ha adquirido una cierta reputación, lo que ha motivado su uso como parte del banco de pruebas a que se ha sometido INNER.

Los tres conjuntos restantes conforman el grupo de problemas Monk's, tres problemas artificiales que fueron creados para evaluar el rendimiento de diferentes algoritmos de aprendizaje y que han adquirido también una fama considerable. Un importante estudio comparativo realizado en torno a este conjunto de problemas puede verse en [Thrun *et al.*, 91]

Todos los conjuntos de ejemplos utilizados en las pruebas han sido descargados del almacén de aprendizaje automático de la Universidad de California en Irvine (UCI) [Blake, Keogh, Merz, 98]. A continuación se describe brevemente en qué consiste cada problema.

BC/LY - Breast Cancer/Lymphography

Son dos de los tres problemas donados por el Instituto de Oncología del Centro Médico Universitario de Ljubljana, Yugoslavia, al almacén de la UCI. En el problema BC cada ejemplo contiene información de una paciente con cáncer de pecho y se pretende aprender a predecir si el tumor es de carácter recurrente o no. Los datos han sido donados por *M. Zwitter* y *M. Soklic*. La distribución por clases es:

BC			LY		
CLASE	Ejemplos	%	CLASE	Ejemplos	%
<i>no_recurrence_events</i>	201	70.28	<i>normal_find</i>	2	1.35
<i>recurrence_events</i>	85	29.72	<i>metastases</i>	81	54.73
	286		<i>malign_lymph</i>	61	41.22
			<i>fibrosis</i>	4	2.70
				148	

CH - Chess end-game

Este problema representa un final de partida de ajedrez entre las blancas, Rey y Torre, y las negras Rey y Peón en A4; mueven las blancas. Este problema suele aparecer en la literatura como KRKPA7. Cada ejemplo representa una descripción del tablero y las clases indican cuando las blancas pueden ganar (won) o cuando no pueden ganar (nowin). La distribución por clases es:

CLASE	Ejemplos	%
<i>won</i>	1669	52.22
<i>nowin</i>	1527	47.78
	3196	

GL - Glass Identification

En este problema se debe clasificar un trozo de cristal en una de las siete clases posibles. La distribución los ejemplos por clases es la que se muestra a continuación, donde se puede observar que no hay ejemplos de la cuarta clase:

CLASE	Ejemplos	%
(1) <i>building_windows_float</i>	70	32.71
(2) <i>building_windows_non_float</i>	76	35.51
(3) <i>vehicle_windows_float</i>	17	7.94
(4) <i>vehicle_windows_non_float</i>	0	0
(5) <i>containers</i>	13	6.07
(6) <i>tableware</i>	9	4.21
(7) <i>headlamp</i>	29	13.55
	214	

G2 - Glass Identification (2)

Es una variante del problema GL en la que se reduce el número de clases a tan solo dos. Para ello se combinan los ejemplos de la clase 1 y 3 en una sola clase y los de la

2 en otra, prescindiendo del resto. La distribución de ejemplos queda entonces como sigue:

CLASE	Ejemplos	%
(1) <i>float</i>	87	53.37
(2) <i>non_float</i>	76	46.63
	163	

HD - Heart Disease

Se pretende aprender a diagnosticar si un paciente padece o no una enfermedad cardíaca en función de 14 atributos¹. Los datos han sido recolectados en la *Cleveland Clinic Foundation* por *Robert Detrano*. La distribución de clases es la siguiente:

CLASE	Ejemplos	%
0	164	54.13
1	139	45.87
	303	

donde la clase 0 indica que el paciente no presenta dolencia alguna, mientras que la clase 1 indica la presencia de afección cardíaca. Originalmente las clases del problema se identificaban con la gravedad de la afección, oscilando los posibles valores entre 0 y 4 pero la práctica habitual es combinar todos los ejemplos de clase 1, 2, 3 y 4 en la clase 1, reduciendo el problema a discernir entre pacientes sanos y pacientes con dolencia cardíaca.

HE - Hepatitis Disease

Cada ejemplo está constituido por los valores de diferentes atributos relacionados con el diagnóstico de una dolencia hepática. El problema consiste en predecir si el paciente va a sobrevivir o no. La distribución de ejemplos es:

CLASE	Ejemplos	%
<i>die</i>	32	20.65
<i>live</i>	123	79.35
	155	

¹De los 76 que originalmente tenía la base de datos sólo se usan 14.

HO - Horse Colic

El problema consiste en determinar si las lesiones sufridas por un caballo precisan intervención quirúrgica (clase 1) o no (clase 2). Aunque los ejemplos de la base de datos original estaban compuestos por 28 atributos sólo se usan 22 de ellos. La distribución de ejemplos es:

CLASE	Ejemplos	%
1	232	63.04
2	136	36.96
	368	

HY/SE - Thyroid Disease

Ambos conjuntos están relacionados con el diagnóstico de enfermedades del tiroides. Proviene del *Garavan Institute* de Sidney, Australia. La distribución de ejemplos en estos problemas es como sigue:

HY			SE		
CLASE	Ejemplos	%	CLASE	Ejemplos	%
<i>negative</i>	3012	95.23	<i>sick-euthyroid</i>	293	9.26
<i>hypothyroid</i>	151	4.77	<i>negative</i>	2870	90.74
	3163			3163	

IR - Iris Flowers

Este es uno de los problemas más conocidos en el ámbito del aprendizaje automático y del reconocimiento de patrones. El conjunto se compone de 150 ejemplos de lirios que pertenecen a una de las variedades siguientes: *setosa*, *versicolor* o *virginica*. La primera publicación en la que se hace referencia a este problema es [Fisher, 36]. La distribución de clases es:

CLASE	Ejemplos	%
<i>Iris-setosa</i>	50	33.33
<i>Iris-versicolor</i>	50	33.33
<i>Iris-virginica</i>	50	33.33
	150	

LA - Labor Negotiations

En este problema se recogen todos los convenios colectivos firmados en Canadá en el sector comercio y servicios en empresas con más de 500 empleados (profesores,

enfermeras, policía, ...) durante 1987 y el primer trimestre de 1998. Cabe destacar de este problema que, además de constar de pocos ejemplos, un 35.75% (326 de los $16 \times 57 = 912$ posibles) de los atributos tienen un valor desconocido, lo que hace aumentar la dificultad. La distribución de clases es:

CLASE	Ejemplos	%
<i>bad</i>	20	35.09
<i>good</i>	37	64.91
	57	

MU - Mushroom Database

Conjunto que contiene la descripción de 23 especies diferentes de setas pertenecientes a las familias *Agaricus* y *Lepiota*. El problema consiste en aprender a clasificarlas correctamente en comestibles o venenosas. La distribución de clases es:

CLASE	Ejemplos	%
<i>Edible</i>	4208	51.80
<i>Poisonous</i>	3916	48.20
	8124	

Cabe destacar de este problema su gran tamaño con respecto al resto de problemas presentados en esta sección, así como su gran cantidad de valores desconocidos, 2480, concentrados, eso sí, en un único atributo, el número 11.

SO - Small Soybean Database

Este es un problema relacionado con el diagnóstico de enfermedades de la soja. En concreto, es un subconjunto bastante reducido del conjunto original, conocido como *Small Soybean Database*. En este problema las clases presentan la distribución que se muestra a continuación:

CLASE	Ejemplos	%
<i>D1</i>	10	21.28
<i>D2</i>	10	21.28
<i>D3</i>	10	21.28
<i>D4</i>	17	36.17
	47	

VO/V1 - Congressional Voting Records

Este conjunto recoge los votos de 435 congresistas en la segunda sesión del Congreso de los Estados Unidos en 1984. Cada ejemplo representa la votación de un congresista en 16 temas claves según el Congressional Quarterly Almanac. Se pretende descubrir el signo político de los congresistas, clasificándolos en *republicanos* o *demócratas*. El conjunto V1 contiene los mismos datos que el VO pero se ha eliminado la información del atributo más relevante [Buntine, Niblett, 92]. La distribución de los ejemplos por clases es:

VO/V1		
CLASE	Ejemplos	%
<i>democrat</i>	267	61.38
<i>republican</i>	168	38.62
	435	

Monk's

Se trata de tres problemas artificiales introducidos por [Wnek *et al.*, 90] para efectuar una comparativa entre distintas técnicas de aprendizaje como la generalización simbólica, las redes de neuronas artificiales y los algoritmos genéticos. Una amplia comparativa fue recopilada en [Thrun *et al.*, 91], teniendo también como base este mismo conjunto de problemas. El dominio de los tres problemas es el mismo, donde se trata de clasificar robots en dos clases diferentes en función de los valores que toman 6 atributos de tipo simbólico.

Atributos en Monk's	
Nombre	Posibles valores
<i>head_shape</i>	round, square, octogonal
<i>body_shape</i>	round, square, octogonal
<i>is_smiling</i>	yes, no
<i>holding</i>	sword, ballon, flag
<i>jacket_color</i>	red, yellow, green, blue
<i>has_tie</i>	yes, no

Cada problema define la función binaria que ha de ser aprendida así como el conjunto de entrenamiento que hay que utilizar, que será un subconjunto de los 432 ejemplos disponibles.

Monk's 1 La función a aprender es que un robot pertenece a una determinada clase (*good*) si la forma de la cabeza y la forma del cuerpo coinciden o si lleva chaqueta

roja; en otro caso, el robot no pertenece a la clase (*bad*). Esto es lo mismo que decir que $(head_shape = body_shape) \vee (jacket_color = red)$.

Monk's 1				
CLASE	Entrenamiento		Prueba	
	Ejemplos	%	Ejemplos	%
<i>good</i>	62	50.00	216	50.00
<i>bad</i>	62	50.00	216	50.00
	124		432	

Monk's 2 Este problema describe los robots de la clase *good* como aquellos en los que dos y sólo dos de sus atributos toman el primer valor posible, que es lo mismo que decir aquellos que cumplan dos y sólo dos de las siguientes condiciones: $head_shape = round$, $body_shape = round$, $is_smiling = yes$, $holding = sword$, $jacket_color = red$, $has_tie = yes$

Monk's 2				
CLASE	Entrenamiento		Prueba	
	Ejemplos	%	Ejemplos	%
<i>good</i>	105	62.13	290	67.13
<i>bad</i>	64	37.87	142	32.87
	169		432	

Monk's 3 En este problema un robot es de clase *good* si su chaqueta es verde y lleva espada o si su chaqueta no es azul y su cuerpo no es octogonal, esto es, $(jacket_color = green \wedge holding = sword) \vee (jacket_color \neq blue \wedge body_shape \neq octogonal)$

Monk's 3				
CLASE	Entrenamiento		Prueba	
	Ejemplos	%	Ejemplos	%
<i>good</i>	62	50.82	204	47.22
<i>bad</i>	60	49.18	228	52.78
	122		432	

Este problema presenta una dificultad adicional y es que el conjunto de entrenamiento tiene un 5% de ejemplos con ruido.

Banco de Pruebas

Prob.	Tamaño	Prec. Mín.	Valores Desc.	Atributos, número de valores distintos							Tot.
				cont.	2	3	4	5	6	>6	
BC	286	70.3	SI	—	3	2	—	1	1	2	9
CH	3196	52.2	NO	—	35	1	—	—	—	—	36
GL	214	35.3	NO	9	—	—	—	—	—	—	9
G2	163	53.4	NO	9	—	—	—	—	—	—	9
HD	303	54.5	SI	5	3	3	2	—	—	—	13
HE	155	79.4	SI	6	13	—	—	—	—	—	19
HO	368	63.0	SI	7	2	5	5	2	1	—	22
HY	3163	95.2	SI	7	18	—	—	—	—	—	25
IR	150	33.3	NO	4	—	—	—	—	—	—	4
LA	57	64.9	SI	8	3	5	—	—	—	—	16
LY ^a	148	56.7	NO	2	9	2	5	—	—	—	18
MU	8124	51.8	SI	—	5	1	5	1	2	7	22
SE	3163	90.7	SI	7	18	—	—	—	—	—	25
SO	47	36.2	NO	—	13	3	4	—	—	1	35
VO	435	61.4	SI	—	16	—	—	—	—	—	16
V1	435	61.4	SI	—	16	—	—	—	—	—	16
Monk's 1	432	50.0	NO	—	2	3	1	—	—	—	6
Monk's 2	432	67.1	NO	—	2	3	1	—	—	—	6
Monk's 3	432	52.8	NO	—	2	3	1	—	—	—	6

Tabla 5.1: Información de las características mas relevantes de cada uno de los problemas utilizados en las pruebas experimentales. La información mostrada de izquierda a derecha es: el nombre o código del problema, el número de ejemplos que lo componen, el porcentaje de ejemplos de la clase mayoritaria, si tienen o no valores desconocidos, el número de atributos continuos y el número de atributos simbólicos, indicando cuantos pueden tomar 2 valores, cuantos pueden tomar 3, etc. hasta cuantos puede tomar más de 6 valores; la última columna muestra el número total de atributos que describen cada ejemplo. Nótese por ejemplo que en el problema SO hay $13 + 3 + 4 + 1 = 21$ atributos que toman más de un valor, y $35 - 21 = 14$ atributos que toman un único valor.

^aEn [Holte, 93] se afirma que este problema tiene 141 ejemplos. Sin embargo el conjunto tomado del almacén de la UCI contiene 148.

5.3 Resultados

En esta sección se muestran los resultados obtenidos por INNER en los problemas descritos anteriormente y se establece una comparativa con los sistemas RISE y C4.5 tanto en precisión alcanzada como en complejidad de la teoría aprendida; esta complejidad se medirá en términos de número de reglas y número de premisas de las reglas.

Se han efectuado pruebas de *validación cruzada estratificada*, con 10 particiones y repitiendo el cada experimento 5 veces. Esto significa que se ha dividido cada conjunto de forma aleatoria en 10 partes aproximadamente iguales manteniendo la proporción de clases del conjunto original. A continuación se procede de forma iterativa, utilizando cada partición como conjunto de prueba de las reglas aprendidas sobre las otras 9 particiones, lo que da lugar a 10 experimentos diferentes; esto se repite 5 veces, obteniéndose los resultados de 50 experimentos distintos.

Para establecer una comparación entre los distintos sistemas en la que los efectos del azar quedasen completamente anulados se ha suministrado a la librería MLC++ [Kohavi *et al.*, 94], encargada de generar las distintas particiones para la validación cruzada, la misma semilla² en todos los casos, garantizándose así que los 50 pares de conjuntos entrenamiento/prueba han sido exactamente los mismos para todos los sistemas. Además, todas las pruebas fueron realizadas en la misma máquina para evitar diferencias que pudieran ser debidas al método de cálculo en coma flotante de distintos procesadores. En las tablas que se muestran en el resto de la sección se recoge la media de los resultados obtenidos en los 50 experimentos junto con el error estándar, que se calcula como:

$$E = \sqrt{\frac{\sum_i \frac{e_i^2}{n} - (\sum_i \frac{e_i}{n})^2}{n - 1}} \quad (5.1)$$

donde n es el número de experimentos (50) y e_i es el error cometido en cada uno de ellos, con $i = 1, 2, \dots, n$.

Una excepción a esta forma de proceder la presentan los problemas Monk's, puesto que ya establecen de antemano cuales son los ejemplos de entrenamiento y prueba que hay que utilizar; para los tres problemas se efectuó una ejecución con RISE y con C4.5, puesto que su resultado es determinístico. Sin embargo, al sistema INNER le afecta el orden de presentación de los ejemplos así como la elección de las instancias iniciales como reglas puntuales, aunque no mucho como se constatará más adelante; por esta razón se efectuaron 50 experimentos con 50 semillas diferentes³ para el generador de

²La semilla para el generador de las particiones fue 2032, el número local del teléfono del laboratorio donde se llevaron a cabo los experimentos. INNER utiliza un generador de números aleatorios para barajar los ejemplos antes de presentarlos al sistema así como para seleccionar en cada ciclo las instancias a convertir en reglas puntuales. Para INNER la semilla inicial fue en todos los casos 34259944, el D.N.I. del autor.

³Las semillas tomaron los valores 34259944 al 34259994.

números aleatorios con objeto de hacer una comparativa más rigurosa. Los resultados presentados para estos tres problemas son la media de los 50 experimentos.

Una de las comparativas, mostrada en la Tabla 5.2, contrasta el error cometido por los sistemas en los experimentos realizados. En este sentido INNER está a la altura de RISE y de C4.5, obteniendo un porcentaje de error algo inferior al del C4.5 y ligeramente superior al de RISE; prácticamente equidistante del obtenido, en media, por ambos sistemas. La Tabla 5.3 compara el tamaño de las soluciones obtenidas, donde el claro vencedor es INNER.

	INNER Error (%)	RISE Error (%)	C4.5 Error (%)	INNER vs. RISE	INNER vs. C4.5
BC	28.97 ± 1.21	28.76 ± 1.13	29.52 ± 1.26	0.21	-0.55
CH	6.61 ± 0.36	1.94 ± 0.09	1.02 ± 0.08	4.67	5.59
G2	21.76 ± 1.66	22.62 ± 1.37	22.44 ± 1.39	-0.86	-0.68
GL	37.65 ± 1.42	28.83 ± 1.52	33.97 ± 1.26	8.82	3.68
HD	17.20 ± 1.17	18.72 ± 0.92	20.37 ± 1.15	-1.52	-3.17
HE	19.60 ± 1.29	20.92 ± 1.38	21.43 ± 1.92	-1.32	-1.83
HO	15.66 ± 0.83	15.00 ± 0.75	16.96 ± 0.76	0.66	-1.30
HY	3.83 ± 0.21	1.79 ± 0.11	0.78 ± 0.06	2.04	3.05
IR	3.87 ± 0.63	4.01 ± 0.60	4.80 ± 0.76	-0.14	-0.93
LA	11.27 ± 1.72	8.14 ± 1.50	18.00 ± 1.87	3.13	-6.73
LY	25.10 ± 1.72	19.19 ± 1.63	23.79 ± 1.74	5.91	1.31
MU	1.48 ± 0.06	0.00 ± 0.00	0.01 ± 0.00	1.48	1.47
SE	7.14 ± 0.29	3.62 ± 0.17	2.30 ± 0.12	3.52	4.84
SO	0.80 ± 0.56	0.00 ± 0.00	2.60 ± 1.01	0.80	-1.80
V1 ^a	10.39 ± 0.63	10.90 ± 0.55	9.94 ± 0.65	-0.51	0.45
VO	4.70 ± 0.46	5.29 ± 0.41	4.83 ± 0.45	-0.59	-0.13
Monk's 1	0.00 ± 0.00	13.40 ± 1.64	8.33 ± 1.33	-13.40	-8.33
Monk's 2	31.70 ± 0.24	30.60 ± 2.22	33.80 ± 2.28	1.10	-2.10
Monk's 3	2.34 ± 0.18	7.20 ± 1.25	3.70 ± 0.91	-4.86	-1.36
Media	13.16	12.68	13.61	0.48	-0.45

Tabla 5.2: Resultados obtenidos con INNER, RISE y C4.5 en su versión generadora de reglas. Los valores de las dos últimas columnas de esta tabla se han obtenido como la diferencia entre el error cometido por INNER y el error cometido por el otro sistema objeto de la comparación; por tanto, los valores negativos indican que el error cometido por INNER es menor. Se observa que, en media, la diferencia entre el error cometido por INNER, RISE y C4.5 es pequeña.

^aEn las descripciones de los problemas V1 y VO se advierte que el significado del símbolo ? es el de la abstención del congresista en el voto y no que se desconozca cuál fue el voto emitido. Los valores mostrados en la tabla son los resultantes de haber utilizado el problema original. No obstante se ha probado a sustituir los interrogantes por un tercer valor que indicase abstención, pasando así a tener un par de problemas V1 y VO sin valores desconocidos. El resultado de INNER sobre ellos fue peor, obteniéndose un error medio del 11.31% en V1 y del 4.92% en VO.

	INNER		RISE		C4.5	
	Reglas	Anteced.	Reglas	Anteced.	Reglas	Anteced.
BC	9.12	22.58	125.72	866.94	8.66	19.16
CH	4.74	9.48	144.10	4145.04	26.70	99.40
G2	11.94	32.30	54.98	494.82	8.32	21.62
GL	18.66	52.80	68.24	614.16	13.72	48.92
HD	6.34	13.92	93.20	978.58	13.46	37.08
HE	5.46	9.80	69.58	819.76	8.36	21.94
HO	4.78	6.42	248.62	2497.14	5.74	10.74
HY	1.58	2.62	870.66	18620.74	6.28	12.72
IR	3.90	6.02	22.72	90.88	4.02	6.04
LA	5.52	6.96	36.72	272.56	4.08	6.08
LY	7.84	13.92	57.12	749.54	10.16	23.12
MU	4.12	3.24	23.36	281.66	16.84	25.54
SE	4.06	9.16	881.54	19845.47	12.54	40.14
SO	4.90	4.90	4.00	44.42	4.00	5.20
V1	5.32	14.42	109.74	1034.42	11.16	28.80
VO	3.10	6.16	87.44	712.88	6.20	13.96
Monk's 1	5.00	7.00	63.00	318.00	11.00	23.00
Monk's 2	26.90	92.40	88.00	414.00	13.00	35.00
Monk's 3	3.26	2.58	51.00	217.00	12.00	25.00
Media	7.19	16.67	163.14	2790.42	10.33	26.50
Sin HY y SE			79.27	855.99		

Tabla 5.3: En esta tabla se muestra el tamaño medio de la solución obtenida para cada problema. Este tamaño se mide en número medio de reglas y de antecedentes de éstas en los 50 experimentos realizados con cada problema. Los experimentos relacionados con los problemas Monk's no han sido validaciones cruzadas, ya que estos problemas definen el conjunto de entrenamiento y prueba a utilizar; no obstante, los datos pertenecientes al sistema INNER son el resultado de 50 ejecuciones distintas para minimizar el efecto (positivo o negativo) que el azar pudiese ejercer. Se observa una dramática reducción en el número de reglas y de antecedentes especialmente con respecto al sistema RISE. Cabe destacar también que los resultados con Monk's 1 han sido particularmente satisfactorios, ya que en las 50 ejecuciones se han obtenidos siempre las 5 reglas que definen exactamente la función binaria a aprender:

```

good ← jacket_color ∈ {red}
good ← head_shape ∈ {octagon}, body_shape ∈ {octagon}
good ← head_shape ∈ {square}, body_shape ∈ {square}
good ← body_shape ∈ {round}, head_shape ∈ {round}
bad ←

```

5.3.1 \mathcal{K} -INNER: Una variante menos explícita de INNER

El proceso de generalización de antecedentes simbólicos que incorpora INNER hace uso de un mecanismo para medir y modificar distancias que hemos denominado \mathcal{K} -HEOM y que se describe en la sección 3.2. El trabajo fundamental de este mecanismo es el de sugerir finalmente qué valores simbólicos deben estar en los antecedentes, convirtiéndose mediante la regularización en conjuntos clásicos de valores que definen *explícitamente*, junto con el resto de antecedentes, los ejemplos cubiertos por las reglas.

Una vez obtenidas las reglas finales se utiliza la métrica de solapamiento, de forma que durante la clasificación de ejemplos las distancias solo pueden ser 0 ó 1; sin embargo, durante el proceso de generalización las distancias entre valores y antecedentes simbólicos se mide con mayor precisión utilizando valores reales entre 0 y 1, a modo de grado de pertenencia del valor al antecedente.

Es razonable pensar que, si mantenemos y usamos en las reglas finales las tablas de diferencias utilizadas durante la generalización, tal vez la precisión pueda aumentar. Por ello se ha construido una variante del sistema INNER, que denominaremos \mathcal{K} -INNER, y que se diferencia de la versión original precisamente en que induce un conjunto de reglas junto con las tablas de diferencias de los antecedentes simbólicos que luego utiliza para clasificar los ejemplos. Los resultados obtenidos, mostrados en la Tabla 5.4, son sólo ligeramente mejores a los obtenidos con la versión original.

Para contrastar estadísticamente los resultados obtenidos se ha efectuado una prueba t-test de dos colas para muestras pareadas con los porcentajes de error cometido por INNER, RISE, C4.5 y \mathcal{K} -INNER. Esta prueba corrobora que no hay diferencia significativa en la precisión obtenida por todos estos sistemas ya que, para la hipótesis nula que se plantea “*hay diferencia significativa en la precisión de los sistemas, comparados dos a dos*”, se obtiene una probabilidad de equivocarse muy alta que obliga a rechazar dicha hipótesis. Concretamente en la prueba entre INNER y RISE ese valor es de 0.649 y entre INNER y C4.5 es de 0.582, valores muy superiores al 0.05 habitual que permitiría aceptar la hipótesis.

5.3.2 La influencia del parámetro de cobertura

Para decidir cuando parar los ciclos de selección y generalización, INNER utiliza a modo de umbral un porcentaje de ejemplos a cubrir. Cuando dicho umbral es rebasado en todas las clases del problema finaliza el proceso iterativo, dando paso al post-proceso de las reglas obtenidas. Por defecto el umbral se ha establecido de forma experimental en el 95%; parece razonable que sea un valor alto, ya que pretendemos que las reglas cubran muchos ejemplos, pero no es fundamental que sea estrictamente el 95% ya que otros valores en torno a éste dan resultados similares. Como muestra de ello se recogen en la Tabla 5.5 los resultados obtenidos para un umbral del 90% y del 100%, donde se puede comprobar que las diferencias son mínimas con respecto al

uso del umbral por defecto.

5.3.3 La influencia de la selección inicial de instancias

En lo que respecta a la selección inicial de instancias es evidente que la influencia de este proceso disminuye al aumentar el número de ciclos del proceso de generalización. Esta selección sería fundamental si sólo se efectuase un ciclo, puesto que la diferencia entre partir de una regla puntual estratégicamente situada y partir de otra situada en una región poco relevante para la clase llevaría al algoritmo a obtener unos buenos resultados o unas malas generalizaciones, respectivamente; pero el algoritmo repite los ciclos de generalización, aumentando la probabilidad de seleccionar instancias en regiones adecuadas, por lo que esta cuestión tampoco es clave para la obtención de los resultados mostrados en las tablas anteriores.

Como prueba de esto se muestran en la Tabla 5.6 los resultados obtenidos utilizando un sistema auxiliar especializado en la selección de *ejemplos paradigmáticos* denominado BETS [Del Coz *et al.*, 99]. La selección de instancias propuestas por este sistema se realiza en base a unos criterios de cobertura similares a los que se han planteado en INNER, con la diferencia de que en INNER esa cobertura es explícita mientras que en BETS la cobertura viene dada por el área de influencia de la instancia seleccionada. En la tabla se puede observar que los resultados obtenidos son sólo ligeramente superiores a los ofrecidos por INNER. Sin embargo hay que destacar que el sistema BETS permite que la cobertura umbral se alcance en menos ciclos, estableciéndose el límite en tan sólo tres (en lugar de los 5 que puede llegar a efectuar el sistema original), puesto que ya en el primer ciclo se consigue cubrir gran parte del espacio de atributos.

	\mathcal{K} -INNER Error (%)	\mathcal{K} -INNER vs. RISE	\mathcal{K} -INNER vs. C4.5	\mathcal{K} -INNER vs. INNER
BC	27.37 ± 1.19	-1.39	-2.15	-1.60
CH	6.63 ± 0.36	4.69	5.61	0.02
G2	21.76 ± 1.66	-0.86	-0.68	0.00
GL	37.65 ± 1.42	8.82	3.68	0.00
HD	17.07 ± 1.24	-1.65	-3.30	-0.13
HE	19.98 ± 1.39	-0.94	-1.45	0.38
HO	15.50 ± 0.76	0.50	-1.46	-0.16
HY	3.83 ± 0.21	2.04	3.05	0.00
IR	3.87 ± 0.63	-0.14	-0.93	0.00
LA	11.27 ± 1.72	3.13	-6.73	0.00
LY	22.40 ± 1.73	3.21	-1.39	-2.70
MU	1.47 ± 0.06	1.47	1.46	-0.01
SE	7.28 ± 0.28	3.66	4.98	0.14
SO	0.80 ± 0.56	0.80	-1.80	0.00
V1	10.72 ± 0.64	-0.18	0.78	0.33
VO	4.83 ± 0.45	-0.46	0.00	0.13
Monk's 1	0.51 ± 0.26	-12.89	-7.82	0.51
Monk's 2	31.74 ± 0.26	1.14	-2.06	0.04
Monk's 3	2.91 ± 0.19	-4.29	-0.79	0.57
Media	13.03	0.35	-0.58	-0.13

Tabla 5.4: Esta tabla muestra los resultados obtenidos con \mathcal{K} -INNER. Los valores de las columnas 2 a 4 se han obtenido como la diferencia entre el error cometido por \mathcal{K} -INNER y el error cometido por el otro sistema, así que los valores negativos indican que el error de \mathcal{K} -INNER es menor. Como se puede ver en la última columna, la “versión \mathcal{K} ” de INNER mejora sólo ligeramente los resultados obtenidos, a costa de perder claridad conceptual en las reglas resultantes.

	INNER-90 Error (%)	INNER-100 Error (%)	INNER vs. INNER-90	INNER vs. INNER-100
BC	28.97 ± 1.21	28.97 ± 1.21	0.00	0.00
CH	6.67 ± 0.36	6.66 ± 0.35	-0.06	-0.05
G2	21.88 ± 1.62	21.88 ± 1.66	-0.12	-0.12
GL	37.65 ± 1.42	37.65 ± 1.42	0.00	0.00
HD	17.33 ± 1.17	17.00 ± 1.19	-0.13	0.20
HE	19.47 ± 1.33	19.21 ± 1.29	0.13	0.39
HO	15.72 ± 0.82	15.33 ± 0.83	-0.06	0.33
HY	3.83 ± 0.21	3.83 ± 0.21	0.00	0.00
IR	4.00 ± 0.63	4.80 ± 0.66	-0.13	-0.93
LA	11.53 ± 1.89	11.27 ± 1.72	-0.26	0.00
LY	24.70 ± 1.76	25.50 ± 1.68	0.40	-0.40
MU	3.52 ± 0.34	1.50 ± 0.06	-2.04	-0.02
SE	7.16 ± 0.29	7.14 ± 0.29	-0.02	0.00
SO	0.80 ± 0.56	0.80 ± 0.56	0.00	0.00
V1	10.49 ± 0.61	10.30 ± 0.65	-0.10	0.09
VO	4.51 ± 0.43	4.79 ± 0.45	0.19	-0.09
Monk's 1	0.00 ± 0.00	0.00 ± 0.00	0.00	0.00
Monk's 2	30.99 ± 0.29	31.27 ± 0.23	0.71	0.43
Monk's 3	2.15 ± 0.18	2.54 ± 0.16	0.19	-0.20
Media	13.23	13.18	-0.07	-0.02

Tabla 5.5: En esta tabla se muestran los resultados obtenidos por INNER para dos valores diferentes del parámetro *umbral de cobertura por clase*, utilizado como criterio de parada en el mecanismo iterativo de generalización de instancias. Las dos primeras columnas contienen los resultados para los valores 90% y 100%, respectivamente. En las dos últimas columnas se muestra la diferencia entre los resultados obtenidos por el sistema con el valor por defecto, 95%, frente a los que se muestran en las dos primeras columnas. Los valores negativos indican que el sistema comete menos error usando el valor por defecto.

	BETS + INNER Error (%)	INNER vs. BETS + INNER
BC	27.37 ± 1.32	1.60
CH	6.80 ± 0.40	-0.19
G2	22.42 ± 1.42	-0.66
GL	36.31 ± 1.51	1.34
HD	17.88 ± 1.05	-0.68
HE	20.79 ± 1.39	-1.19
HO	14.58 ± 0.83	1.08
HY	3.39 ± 0.21	0.44
IR	5.60 ± 0.75	-1.73
LA	9.73 ± 1.57	1.54
LY	22.55 ± 1.66	2.55
MU	1.42 ± 0.06	0.06
SE	7.56 ± 0.29	-0.42
SO	0.40 ± 0.40	0.40
V1	10.48 ± 0.67	-0.09
VO	4.28 ± 0.45	0.42
Monk's 1	0.00 ± 0.00	0.00
Monk's 2	31.80 ± 0.37	-0.10
Monk's 3	2.78 ± 0.00	-0.44
Media	12.95	0.21

Tabla 5.6: Estos son los resultados obtenidos por INNER usando un mecanismo auxiliar de selección inicial de instancias denominado BETS. En la última columna y manteniendo la coherencia con tablas anteriores se muestra la diferencia entre el error cometido por INNER y el cometido por BETS + INNER, de forma que los valores negativos indican un menor error (mejor comportamiento) del sistema original. Como se puede apreciar en esta columna de comparación, la media de las diferencias es positiva, lo que significa que BETS + INNER consigue unos resultados ligeramente mejores que INNER sólo; el que este valor sea de tan sólo dos décimas sugiere que la selección de instancias iniciales no es algo crucial en la operación global del algoritmo implementado en INNER.

Capítulo 6

Conclusiones

En esta memoria se ha presentado un nuevo sistema de aprendizaje automático a partir de ejemplos denominado INNER. Este sistema es capaz de sintetizar el conocimiento “inmerso” en un conjunto de ejemplos descritos por atributos tanto simbólicos como numéricos, obteniendo un conjunto de reglas muy compacto y con una precisión similar a la de otros sistemas de reconocido prestigio. Estas reglas se obtienen mediante un proceso de generalización de instancias tomadas del propio conjunto de entrenamiento, haciéndolas crecer hasta cubrir regiones del espacio de atributos en las que se evidencia una acumulación de casos de una determinada clase.

En este proceso de generalización cabe destacar el mecanismo usado para la medida de las distancias denominado \mathcal{K} -HEOM, que incorpora un novedoso sistema para aprender a medir distancias en el terreno simbólico al tiempo que se generalizan las reglas.

Los resultados obtenidos mostrados en el Capítulo 5 corroboran en cierta medida lo que Holte establecía en [Holte, 93], donde indicaba que la sencillez de la solución obtenida puede en muchos casos brindar mejores resultados de clasificación que los que se obtiene con soluciones más complejas. Las soluciones de INNER son mucho menos complejas que las de otros conocidos sistemas de aprendizaje a los que, en ocasiones, supera en precisión.

El reducido número de reglas y condiciones que obtiene INNER se debe en parte al uso de los dos procedimientos importados de ABANICO para tal fin, a saber, la cualificación y la selección de reglas, ambos basados en el nivel de impureza como medida de calidad de las reglas. Estos procedimientos permiten reducir el tamaño de las soluciones sin perder precisión.

El tipo de solución que obtiene INNER es muy adecuado para poder ser validado o contrastado por expertos humanos puesto que:

- está en una sintaxis legible.

- suele consistir en pocas reglas con pocos antecedentes lo que hace que la solución sea conceptualmente clara.
- la clasificación de un ejemplo viene explicada por la propia regla que lo clasifica, que fue sugerida por la presencia de una acumulación importante de ejemplos de entrenamiento de la misma clase en sus proximidades.

Por ello, las reglas sintetizadas por el sistema INNER son adecuadas para incorporar a un asistente informático en tareas en las que sea necesario explicar por qué se toma una decisión, para que el usuario humano pueda contrastar con su experiencia y decidir, en último término, qué hacer.

Bibliografía

- [Aha, 90] AHA, D. W.: A Study of Instance-based Algorithms for Supervised Learning Tasks: Mathematical, Empirical, and Psychological Evaluations. Ph. D. Dissertation. University of California at Irvine (1990).
- [Alonso, García, Bahamonde, 97] ALONSO GONZÁLEZ, J.; GARCÍA ÁLVAREZ, A., y BAHAMONDE, A.: Un Sistema que Adecua las Reglas Aprendidas a las Críticas Recibidas sobre sus Actuaciones. Actas de la VII Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA-97, pp. 581-590. Torremolinos-Málaga, 12-14 noviembre 1997.
- [Alonso, 94] ALONSO, J.: Un Algoritmo de Optimización de Reglas de Clasificación. Memoria de Investigación. Tercer Ciclo en Ciencias de la Computación e Inteligencia Artificial. Universidad de Oviedo. Septiembre de 1994.
- [Bahamonde *et al.*, 97] BAHAMONDE, A., De la CAL, E. A., RANILLA, J., ALONSO, J.: Self-organizing Symbolic Learned Rules. Biological and Artificial Computation: From Neuroscience to Technology, (Mira, Moreno-Diaz, Cabestany Eds.), pp. 536-545. LNCS N° 1240, Springer-Verlag. Berlin (1997).
- [Bahamonde, Vela, Botana, 91] BAHAMONDE, A., VELA, C. R., BOTANA, F.: Generalización de Reglas de Clasificación. Actas de la IV Reunión Técnica de la Asociación Española para la Inteligencia Artificial, AEPIA-91, pp. 231-242. Madrid, (1991).
- [Bahamonde, 94a] BAHAMONDE, A.: Cálculo del desfase en una red de tráfico urbano organizada en rutas. Informe técnico, Centro de Inteligencia Artificial, Universidad de Oviedo en Gijón, (1994).
- [Bahamonde, 94b] BAHAMONDE, A.: Inductive Properties of Lattices. International Journal of Computer Mathematics. Vol. 54, 127-142, (1994).
- [Blake, Keogh, Merz, 98] BLAKE, C., KEOGH, E. & MERZ, C.J.: UCI Repository of machine learning databases. Irvine, CA: University of California, Depart-

ment of Information and Computer Science. (1998).

[<http://www.ics.uci.edu/~mllearn/MLRepository.html>]

- [Blum, 97] BLUM, A.: Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, 26, 5-23 (1997).
- [Botana, Bahamonde, 95a] BOTANA, F. & BAHAMONDE, A.: Aprendizaje de Conceptos a partir de Representaciones Basadas en Grafos. *Informática y Automática*, Vol. 28, N° 2, 25-25, (1995).
- [Botana, Bahamonde, 95b] BOTANA, F. & BAHAMONDE, A.: SHAPE: A Machine Learning System from Examples. *International Journal of Human-Computer Studies*, 42, 137-155, (1995).
- [Botana, 92] BOTANA FERREIRO, F.: SHAPE: Sistema Heurístico de Aprendizaje a partir de Ejemplos, p. 99. Tesis doctoral dirigida por A. Bahamonde y A. Blanco Ferro. Departamento de Computación, Universidade da Coruña. (1992).
- [Breiman, Friedman, Olshen, 84] BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., & STONE, C. J.: *Classification and Regression Trees*. Belmont, Wadsworth International Group, (1984).
- [Buntine, Niblett, 92] BUNTINE, W. & NIBLETT, T.: A Further Comparison of Splitting Rules for Decision-Tree Induction. *Machine Learning*, 8, 75-85, (1992)
- [Cendrowska, 88] CENDROWSKA, J.: PRISM: An Algorithm for Inducing Modular Rules. *International Journal of Machine Studies*, 27, 349-370, (1988).
- [Clark, Niblett, 88] CLARK, P., & NIBLETT, T.: The CN2 Induction Algorithm. *Machine Learning*, 3, 261-284, (1988).
- [Cohen, Hirsch, 94] COHEN, W.W.; HIRSCH, H. (De.): *Machine Learning*. Proceedings of the Eleventh International Conference. July 10th-13th, 1994. Rutgers University, New Brunswick, New Jersey (1994).
- [Cost, Salzberg, 93] COST, S., & SALZBERG, S.: A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10, 57-78, (1993).
- [Cover, Hart, 67] COVER, T. M., & HART, P. E.: Nearest Neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27, (1967).
- [Del Coz *et al.*, 99] DEL COZ, J. J., LUACES, O., QUEVEDO, J.R., ALONSO, J., & BAHAMONDE, A.: Self-Organizing Cases to Find Paradigms. Proceedings of the International Work-Conference on Artificial and Natural Neural Networks,

IWANN '99, Alicante, Spain: Lecture Notes on Computer Science, Springer-Verlag, pp. 527-536, (1999).

[Domingos, 96] Pedro Domingos. Unifying Instance-Based and Rule-Based Induction. *Machine Learning*, 24, 141-168, (1996).

[Domingos, Pazzani, 97] DOMINGOS, P.; PAZZANI, M.: On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, Vol. 29, N. 2/3, 103-130, (1997).

[Fiol, 91] FIOLO ROIG, G.: Contribución a la adquisición inductiva de conocimiento. Tesis doctoral dirigida por José Miró Nicolau. Universidad de las Islas Baleares, (1991).

[Fisher, 36] FISHER, R.: The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7, 1, pp. 179-188, (1936).

[Frank, Witten, 98] FRANK E. & WITTEN I.H.: Generating accurate rule sets without global optimization. *Proc. International Conference on Machine Learning*. Morgan Kaufmann, (1998).

[Frank *et al.*, 99] FRANK E., WANG Y., INGLIS S., HOLMES G. and WITTEN I.H.: Inducing Model Trees for Continuous Classes. *Machine Learning*, Pendiente de publicación. (1999).

[ftp://ftp.aic.uniovi.es/publications/other_authors/weka/
/Wang-Witten-Induct-M5.zip]

[Fürnkranz, 97] Johannes Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review* 13(1):3-54, (1999).

[García, 97] GARCÍA ÁLVAREZ, A.: Evolución de Reglas de Producción: Un Sistema Basado en la Utilidad de las Clasificaciones. Proyecto fin de carrera dirigido por José Ranilla Pastor y Jaime Alonso González, ETSIIeII Gijón, Universidad de Oviedo, Mayo 1997.

[Hernández, López, Bahamonde, 97] HERNÁNDEZ, P.; LÓPEZ, S.; BAHAMONDE, A.: Artificial Neural Networks for the Computation of Traffic Queues. En *Biological and Artificial Computation: From Neuroscience to Technology*, (Mira, Moreno-Diaz, Cabestany Eds.), pp. 1288-1297. LNCS N° 1240, Springer-Verlag. Berlin (1997).

[Holte, 93] HOLTE, R. C.: Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11, 63-91 (1993).

- [Holte, Acker, Porter, 89] HOLTE, R. C., ACKER, L. & PORTER, B.: Concept Learning and the Problem of Small Disjuncts. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), pp. 813-818. (1989).
- [Kohavi *et al.*, 94] KOHAVI, R., JOHN, G., LONG, R., MANLEY, D., & PFLEGER, K.: MLC++: A machine learning library in C++. En Proceedings of the Sixth International Conference on Tools with Artificial Intelligence, pp. 740-743. IEEE Computer Society Press, (1994).
- [Kohavi, Sommerfield, Dougherty, 96] KOHAVI, R.; SOMMERFIELD, D.; DOUGHERTY, J.: Data mining using MLC++. A machine Learning Library in C++. En Tools With AI, IEEE Computer Society Press, 1996, 234-245.
[<http://www.sgi.com/Technology/mlc>]
- [Kohonen, 95] KOHONEN, T.: Self-Organizing Maps. p. xv+362. Springer Series in Information Sciences. Springer Verlag, Berlin, (1995).
- [Littlestone, 88] LITTLESTONE, N.: Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. Machine Learning, 2, 285-318 (1988).
- [Littlestone, Warmuth, 94] LITTLESTONE, N. & WARMUTH, M.K.: The Weighted Majority Algorithm. Information and Computation, 108(2), 212-261 (1994).
- [Luaces *et al.*, 98] LUACES, O., ALONSO, J., De la CAL, E. A., RANILLA, J., & BAHAMONDE, A.: Machine Learning Usefulness Relies on Accuracy and Self-maintenance. Proceedings of the 11th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (pp. 448-457). Benicassim, Spain: Lecture Notes on Artificial Intelligence, Springer-Verlag, (1998).
- [Luaces *et al.*, 99] LUACES, O., Del COZ, J. J., QUEVEDO, J.R., ALONSO, J., RANILLA, J. & BAHAMONDE, A.: Autonomous Clustering for Machine Learning. Proceedings of the International Work-Conference on Artificial and Natural Neural Networks, IWANN '99, Alicante, Spain: Lecture Notes on Computer Science, Springer-Verlag, pp. 497-506, (1999).
- [McCarthy, 80] MCCARTHY, J.: Circumscription - A form of Non-Monotonic Reasoning. Artificial Intelligence, 13, 27-39, (1980).
- [Michalski, Carbonell, Mitchell, 83] MICHALSKI, R. S., CARBONELL, J. G., & MITCHELL, T.M.: Machine Learning. An Artificial Intelligence Approach. Volume I, Tioga Pub., Palo Alto, California, (1983)

- [Michie, Spiegelhalter, Taylor, 94] MICHIE, D., SPIEGELHALTER, D. J. & TAYLOR, C. C., (Ed.): Machine Learning, Neural and Statistical Classification, p. xiv+289. Ellis Horwood, Hertfordshire, Gran Bretaña, (1994).
- [Mones, 96] MONES, R.: Atributos Artificiales y Discretización Supervisada en el Proceso de Segmentación de ABANICO. Proyecto fin de carrera dirigido por Jose Ranilla Pastor, EUITI Gijón, Universidad de Oviedo, Septiembre 1996.
- [Murthy, Kasif, 94] MURTHY, S. K., KASIF, S. & SALZBERG, S.: A System for Induction of Oblique Decision Trees. Journal of Artificial Intelligence Research, 2, 1-32, (1994).
- [Pellitero, 98] PELLITERO, M.: Aplicación Efectiva del Nivel de Impureza en Árboles de Decisión. Proyecto Fin de Carrera dirigido por José Ranilla Pastor, E.T.S.I.I. de Gijón, Universidad de Oviedo, Octubre 1998.
- [Quevedo, 97] QUEVEDO, J. R.: Diseño e Implementación Paralela de ABANICO sobre Plataformas Heterogéneas con PVM. Proyecto fin de carrera dirigido por José Ranilla Pastor, E.T.S.I.I. e I.I. Gijón, Universidad de Oviedo, Noviembre 1997.
- [Quevedo, 99] QUEVEDO, J. R.: Herramientas para el Uso de MLC++. Informe técnico, Centro de Inteligencia Artificial, Universidad de Oviedo en Gijón, (1999).
- [Quinlan, 83] QUINLAN, J. R.: Learning Efficient Classification Procedures and their Application to Chess End Games, en Machine Learning. An Artificial Intelligence Approach. Volume I, Tioga Pub., Palo Alto, California, pp. 463-482, (1983).
- [Quinlan, 87] QUINLAN, J. R.: Simplifying Decision Trees, International Journal of Man-Machine Studies, Vol. 27, pp. 221-234, (1987).
- [Quinlan, 92] QUINLAN, J. R.: Learning with Continuous Classes. Proceedings 5th Australian Joint Conference on Artificial Intelligence. World Scientific, Singapore. 343-348, (1992).
- [Quinlan, 93a] QUINLAN, J. R.: C4.5: Programs for Machine Learning, p. x+302, Morgan Kaufmann Publishers, San Mateo, California, (1993).
- [Quinlan, 93b] QUINLAN, J. R.: Combining Instance-based and Model-based Learning. En Proceedings Tenth International Machine Learning Conference, Amherst, MA, Morgan Kaufmann, (1993).
- [Rachlin *et al.*, 94] RACHLIN, J.; KASIF, S.; SALZBERG, S., AHA, D.W.: Towards a Better Understanding of memory-based Reasoning Systems. En Machine Learning. Proceedings of the Eleventh International Conference, 242-250, (1994).

- [Ranilla, 98] ABANICO: Aprendizaje Basado en Agrupación Numérica en Intervalos Continuos. Tesis doctoral dirigida por A. Bahamonde. Universidad de Oviedo. (1998)
- [Ranilla, Bahamonde, 98] RANILLA, J., & BAHAMONDE, A.: Fan: Finding Accurate iNductions. Technical Report, Artificial Intelligence Center, University of Oviedo at Gijón. November (1998).
[ftp://ftp.aic.uniovi.es/publications/Machine_Learning/Fanprn.ZIP]
- [Ranilla, Bahamonde, 95] RANILLA, J. & BAHAMONDE, A.: Segmentación de Valores Numéricos para el Aprendizaje a partir de Ejemplos. Actas de la VI Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA-95, pp. 225-234. Alicante (1995).
- [Ranilla, Mones, Bahamonde, 98] RANILLA, J.; MONES, R., y BAHAMONDE, A.: El Nivel de Impureza de una Regla de Clasificación Aprendida a partir de Ejemplos. Actas de la VII Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA-97, pp. 479-488. Torremolinos-Málaga, 12-14 noviembre 1997. También en Revista Iberoamericana de Inteligencia Artificial, 4, pp. 4-11, (1998) y en Novatica, (Vol. 131, 37-43, 1998).
- [Salzberg, 90] SALZBERG, S.: Learning with Nested Generalized Exemplars, p. xviii+159, Kluwer Academic Publishers, Boston, (1990).
- [Sestito, Dilon, 94] SESTITO, S. y DILON, T. S.: Automated Knowledge Acquisition, p. x+378, Prentice Hall, Englewood Cliffs, N.J., (1994).
- [Skalak, 94] SKALAK, DAVID B.: Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. En Machine Learning. Proceedings of the Eleventh International Conference, p. 293-301, (1994).
- [Spiegel, 70] SPIEGEL, M. R.: Estadística, p. x+357. McGraw-Hill, Atlacomulco, México, (1970).
- [Thrun *et al.*, 91] THRUN, S. B., BALA, J., BLOEDORN, E., *et al.*: The MONK's Problems - A Performance Comparison of Different Learning algorithms, Technical Report CS-CMU-91-197, Carnegie Mellon University, (1991).
- [Weiss, Indurkha, 95] WEISS, S.; INDURKHIA, N.: Rule-based Machine Learning Methods for Functional Prediction. Journal of Artificial Intelligence Research, JAIR, 3, 383-403, (1995).
- [Wettschereck, Aha, Mohri, 97] WETTSCHERECK, D., AHA, D. W., & MOHRI, T. A Review and Comparative Evaluation of Feature Weighting Methods for Lazy Learning Algorithms. Artificial Intelligence Review, 11, 273-314. Also available

as Technical Report AIC-96-006: Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence. (1997)

[Wettschereck, Dietterich, 95] WETTSCHERECK, D., DIETTERICH, T. An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, 19, 5-27 (1995).

[Wilson, Martinez, 97] WILSON, D. R. & MARTINEZ, T. R.: Improved Heterogeneous Distance Functions, *Journal of Artificial Intelligence Research*, vol. 6, no. 1, pp. 1-34, (1997).

[Wilson, Martinez, 99] WILSON, D. R. & MARTINEZ, T. R.: Reduction Techniques for Exemplar-Based Learning Algorithms, *aparecerá en Machine Learning Journal*, 1999.

[Winston, 94] WINSTON, P. H.: *Inteligencia Artificial*. Tercera Edición, p. xxv+805, Addison-Wesley Iberoamericana, Wilmington, Delaware, EE.UU., 1994 (traducción de la tercera edición en inglés de 1992).

[Wnek *et al.*, 90] WNEK, J., SARMA, J., WAHAB, A., & MICHALSKI, R.: Comparison Learning Paradigms Via Diagrammatic Visualization: A Case Study in Single Concept Learning Using Symbolic, Neural Net and Genetic Algorithm Methods. Technical Report, George Mason University, Computer Science Department, (1990).