POLITEXT

A. Moreno - E. Armengol - J. Béjar
L. Belanche - U. Cortés - R. Gavaldà
J.M. Gimeno - B. López - M. Martín
M. Sànchez

Aprendizaje automático

EDICIONS UPC

POLITEXT 36

Aprendizaje automático

Antonio Moreno - Eva Armengol - Javier Béjar
Lluís Belanche - Ulises Cortés - Ricard Gavaldà
Juan Manuel Gimeno - Beatriz López - Mario Martín
Miquel Sànchez

EDICIONS UPC

Diseño de la cubierta: Manuel Andreu

© Edicions UPC, 1994
Edicions de la Universitat Politècnica de Catalunya, SL
Jordi Girona Salgado 31, 08034 Barcelona
Tel. 934 016 883 Fax. 934 015 885
Edicions Virtuals: www.edicionsupc.es
e-mail: edupc@sg.upc.es

Producción: Servei de Publicacions de la UPC
y CPET (Centre de Publicacions del Campus Nord)
La Cup. C. Gran Capità s/n, 08034 Barcelona

Depósito legal: B-5.473-94
ISBN: 84-7653-460-4

Contenido

Lista de Figuras

Lista de Tablas

"Quem lê deixa de viver.
Fazei agora por que o façais.
Deixai de viver, e lede.
O que é a vida? "

"Poesia", Fernando Pessoa

Capítulo 1 Introducción

> "Para empezar con las máquinas que aprenden: un sistema organizado puede definirse como aquel que transforma un cierto mensaje de entrada en uno de salida, de acuerdo con algún principio de transformación. Si tal principio está sujeto a cierto criterio de validez de funcionamiento, y si el método de transformación se ajusta a fin de que tienda a mejorar el funcionamiento del sistema de acuerdo con ese criterio, se dice que el sistema *aprende*."
>
> **N. Wiener**

1.1 Aprendizaje animal y automático

No es habitual comenzar un libro sobre el tipo de aprendizaje que puede realizar una máquina, al que se llamará aquí *aprendizaje automático*, estudiando los procesos de aprendizaje que son observables en la naturaleza, aunque existen algunos precedentes (por ejemplo, **[GALL91]** y **[PEAR87]**). La razón es que si se quiere buscar un marco cognitivo para explicar el fenómeno del aprendizaje, parece razonable referirse a aquellas conductas observables en los animales –y que son identificables como *aprendizaje*– que pueden ayudar a explicar de manera más completa un proceso tan complejo como el que nos ocupa.

No basta sólo con intentar explicar *qué* y *cómo* se aprende en términos de procesos generales tales como asociaciones, abstracción de prototipos, pruebas de hipótesis, inducción, razonamiento analógico, asimilación, generalización o diferenciación. Hay razones para afirmar que existe cierto sesgo en las estructuras cognitivas que se emplean en cada una de las tareas mencionadas y que dicho sesgo depende directamente de características específicas del dominio donde *algo* ha ser aprendido.

1.1.1 Aprendizaje animal

Cuando los organismos se ajustan o adaptan al conjunto de estímulos que provienen del entorno, es decir, reciben información y la almacenan con el fin de reutilizarla en situaciones o patrones de estímulos semejantes, se puede decir que *aprenden*. En particular nos referimos a los animales ya que, desde el punto de vista del aprendizaje, y a diferencia de las plantas, son

móviles y activos. De hecho, se puede afirmar que los animales son máquinas que presentan una conducta predatoria y tienen que moverse para localizar su alimento y conseguirlo o, al menos, están dotados de órganos especializados para ello, como es el caso de muchos seres marinos que no se *mueven* pero son capaces de filtrar el agua que absorben para alimentarse. Este énfasis en el *movimiento* de los animales es fundamental para explicar la existencia de diferentes tipos de aprendizaje. Es posible imaginar dos situaciones muy diferentes:

- El animal se mueve de manera aleatoria; en este caso el mismo movimiento debe acercarle al medio en el cual pueda proveerse de todo lo necesario para su subsistencia (agua, oxígeno, comida, *etc*) y asegurar su desarrollo y la supervivencia de una proporción significativa de los seres de su especie.

- Si un movimiento aleatorio no es suficiente debe existir un movimiento dirigido para lo que debe haber órganos especiales de detección de objetos en el entorno y guiar al animal hacia ellos, de forma directa o indirecta. La mayor precisión de estos movimientos depende de la evolución causada por la presión generada por la competencia con otras especies por unos recursos acotados en un determinado hábitat ecológico.

Una parte significativa de la conducta de muchos animales parece estar fuertemente arraigada e influenciada por la experiencia, de tal forma que puede ser descrita como innata, instintiva o, simplemente, no aprendida. Es decir, muchos animales están de alguna forma *programados* de manera innata y, cuando perciben alteraciones en su entorno y cambian sus patrones de conducta – como resultado de esta percepción – se puede decir que aprenden. Desde este punto de vista, el aprendizaje[1] puede ser definido como la organización (o reorganización) de la propia conducta (ante una situación o un patrón de estímulos) como resultado de una experiencia individual. La definición anterior indica un mínimo de características que un fenómeno debe presentar para poder ser clasificado como un ejemplo de aprendizaje y evitar en la medida de lo posible la confusión que puede causar intentar definirlo a partir de la manera en que el proceso de aprendizaje se ha realizado. Esta definición permanece inmutable aún en caso de necesitar explicar este proceso en situaciones en las que la conducta se ve continuamente modificada por la adquisición de nuevos conocimientos. Algunos autores han sugerido la necesidad de explicar el aprendizaje animal en términos conductistas y después en términos cognitivos, si es posible.

Frecuentemente en la literatura, se considera al aprendizaje como un proceso *adaptativo*, es decir, que se manifiesta mediante cambios suaves, incrementales. Se considerará aquí la adaptatividad del aprendizaje como la medida de ajuste de una conducta. Además, se asume que lo aprendido permanece *en memoria* durante períodos relativamente largos, de manera que asegura la observabilidad de la conducta aprendida (como algo relativamente estable).

Lo que un animal puede aprender no depende solamente de su capacidad para ello, ya que existen muchas otras restricciones y limitaciones que moldean esta capacidad (*e.g.* las limitaciones en sus sistemas sensoriales). Así, dadas estas limitaciones, un animal está más predispuesto a reaccionar a un tipo de estímulos que a otros, puede aprender más de éstos que de aquellos. Aún más, todos los estímulos a los que un individuo responde en un cierto contexto pueden no ser efectivos para producir una conducta de aprendizaje en otros contextos.

[1] Las formas de definir el aprendizaje e interpretar las conductas observadas dependen fundamentalmente de la manera de considerar la naturaleza del *aprendiz*.

Como consecuencia, el aprendizaje debe considerarse como una de las características más *apreciables* en un sistema.

1.1.2 Tipos de aprendizaje animal

Resulta casi imposible dividir los diversos tipos de aprendizaje en categorías mutuamente exclusivas, que puedan ser definidas exactamente y cubran todo el espectro del aprendizaje animal. Aquí no se pretende ser exhaustivo y tan sólo se busca dar una visión de conjunto del estado del arte.

Habituación Es un tipo de aprendizaje que consiste en una respuesta que decae ante un conjunto de estímulos repetidos (o contínuos) no asociados a ningún tipo de recompensa o refuerzo. La habituación se puede caracterizar como asociada a un estímulo específico y su relativa permanencia la distingue de manifestaciones temporales como la fatiga o la adaptación sensorial (*e.g.* la adaptación a los espacios con poca luz o a los muy ruidosos). La habituación implica una tendencia a borrar todo tipo de respuesta ante un estímulo que no tiene importancia para la supervivencia. A pesar de ser el tipo más simple de aprendizaje resulta muy efectivo, especialmente en los organismos más simples, ya que sirve como filtro a conjuntos de estímulos que no son relevantes evitando la innecesaria especialización de algún órgano. No obstante, la habituación como mecanismo de aprendizaje está presente en todos los tipos de organismos, independientemente de su complejidad. Sin embargo, los mecanismos que subyacen en el proceso de habituación se vuelven más elaborados cuando los organismos devienen más complejos.

Aprendizaje asociativo Frecuentemente en los entornos en los que los animales se mueven, un evento permite precedir, con cierta confianza, la ocurrencia (o no ocurrencia) de otro. La aparición de ciertos rasgos en el paisaje puede indicar el cambio de estación, el cambio de comportamiento de algunos individuos de especie puede anunciar la temporada de celo, o la ingestión de alguna planta (o animal) puede causar alguna enfermedad (o producir consecuencias benéficas). Un animal que *conoce* esas relaciones puede sacar provecho anticipándose a esos eventos y así comportarse apropiadamente. Pero, ¿cómo se adquiere ese conocimiento?

A pesar de que el concepto de asociación –la conexión entre un estímulo y una respuesta que no ha existido antes en la conducta de un organismo– tiene una larga historia que puede ser trazada fácilmente, y a pesar de que en la década de 1880 ya se aplicaba este concepto en los estudios experimentales sobre al aprendizaje humano, los estudios psicológicos modernos del aprendizaje animal asociativo no comenzaron hasta el final del siglo `XIX`. En ese momento un grupo de psicólogos rusos comenzó a dar las primeras explicaciones sobre cómo las conductas adquiridas, y, probablemente, también las heredadas, pueden ser modificadas y adaptadas mediante su asociación a un nuevo estímulo durante el proceso de entrenamiento (aprendizaje).

Condicionamiento Los estudios de I. Pavlov sobre la digestión, usando perros, le convirtieron en el investigador ruso más influyente en el final del siglo pasado y sus experimentos dieron lugar a la formulación de la teoría del reflejo condicionado, o condicionamiento clásico.

Esencialmente, la noción de condicionamiento clásico denota el proceso mediante el cual un animal adquiere la capacidad de responder a un estímulo determinado con la misma acción refleja con que respondería a otro estímulo condicionante (refuerzo o recompensa) cuando ambos estímulos se presentan concurrentemente (o sobrepuestos en una secuencia) un cierto número de veces.

Aprendizaje mediante prueba y error Este tipo de aprendizaje se identificó al observar la conducta de ciertos animales que obtienen recompensas (*i.e.* comida) después de realizar con éxito ciertas tareas como p.e. gatos intentando escapar de una caja tras presionar alguna palanca, ratas lanzadas en medio de una bañera que han de identificar alguna marca para nadar hacia sitios secos, *etc.* En esas situaciones los animales permanecen siempre activos y su atención se fija primero aquí y luego allá probando todas las posibilidades imaginables hasta que de manera más o menos accidental resuelve con éxito la tarea y obtiene la recompensa. Esto a pesar de no existir una relación entre las acciones realizadas y la superación de la prueba. El aprendizaje mediante prueba y error requiere entonces la existencia del refuerzo (o recompensa) para animar la selección de la respuesta adecuada de entre una variedad de conductas en una situación determinada, hasta que finalmente se establece una relación entre el estímulo o situación y una respuesta correcta para obtener una recompensa.

En este caso el refuerzo está precedido por el estímulo y la respuesta requerida, lo que no ocurre forzosamente en el condicionamiento clásico. A este tipo de aprendizaje se le ha dado muchos otros nombres, tales como condicionamiento operante, condicionamiento instrumental, *etc.*

Aprendizaje latente El aprendizaje latente es un tipo de aprendizaje asociativo que tiene lugar en ausencia de recompensa. Un experimento clásico es el realizado con ratas que son dejadas en libertad en un laberinto durante varios días sin ningún tipo de recompensa. Cuando su aprendizaje es comparado con otro grupo que no ha estado en el laberinto y comienza a ser recompensado inmediatamente, los resultados del primer grupo son sorprendentes: aprenden más rápidamente y con menos errores que el segundo grupo. De aquí se desprende que el primer grupo aprendió *algo* durante su estancia en el laberinto que permanece *latente* hasta que es *necesitado*.

Imitación La imitación ha sido frecuentemente considerada como una evidencia de la existencia de conductas altamente reflexivas, a pesar de que diversos fenómenos son incluídos bajo la etiqueta de imitación.

Uno de los tipos de imitación más comunes es el denominado facilitación social (*social facilitation*) que describe un patrón de conducta ya existente en el repertorio de un individuo, ya que éste realiza cuando la misma conducta es realizada por otros miembros de su especie. Por ejemplo, en los humanos, bostezar.

Pero la verdadera imitación, que implica copiar una conducta, acción o expresión nueva o que resulta imposible de aprender si no es copiada de otro individuo, se presenta especialmente en los humanos y en algunos chimpancés y monos. En particular, uno puede imaginar como un ejemplo de este tipo de aprendizaje el que ocurre cuando un individuo es entrenado para realizar un salto con pértiga. En otros animales, como

los felinos y otros cazadores, el aprendizaje de cómo matar certeramente una presa es realizado mediante la imitación de los padres y reforzado mediante los juegos[2].

Si bien, tal como se ha definido, la imitación[3] significa una copia consciente de una conducta, acción o expresión realizada por otro individuo, también está asociada a un intento de obtener un provecho de la experiencia de otro.

Impronta La impronta[4] es un ejemplo ilustrativo de la manera en que un rango específico de estímulos es capaz de elicitar una respuesta pudiendo ser limitado y refinado mediante la experiencia.

A pesar de que la impronta no es diferente a otras formas de aprendizaje, tiene sin embargo algunas propiedades que la hacen diferente. Es un proceso mediante el cual un neonato [5] muy dependiente de los padres, para obtener comida y calor, tiene que mantener contacto visual con ellos y puede, de forma accidental, desarrollar una preferencia por cualquier cosa (animal u objeto) diferente de su especie. Esta preferencia se muestra siguiendo la conducta del objeto elegido. A pesar de ser un fenómeno que eminentemente se presenta en las aves, también puede llegar a ocurrir en los mamíferos, pero en éstos su incidencia es menor debido al mayor desarrollo de otros sentidos, especialmente el olfato.

Una de las características más relevantes de la impronta es que su duración se restringe al llamado *período sensitivo*, que es relativamente corto, quizás unas pocas horas[6]. En situaciones experimentales se ha logrado modificar la duración de este período, que aparece tan pronto como se han desarrollado lo suficiente las habilidades motoras para seguir un objeto (los polluelos son atraídos preferentemente por objetos móviles). Sin embargo, en laboratorio, se han modificado estas preferencias mediante el refuerzo adecuado.

Resumiendo, parece ser que la impronta consiste, a grandes trazos, en desarrollar una familiaridad hacia los objetos móviles. Las recompensas, tales como comida o contacto con el objeto o modelo en movimiento, no son necesarias. La impronta es una herramienta valiosa para la comprensión de la génesis de los aspectos perceptuales de la conducta, pues lo aprendido durante el período sensitivo puede revelarse en otros contextos[7]. Esto parece indicar una relación entre la impronta y el desarrollo de la percepción. Una vez creada la familiaridad, vía la impronta, ante la ausencia de objetos el ave muestra una conducta de búsqueda.

Un aspecto que con frecuencia no se ha considerado suficientemente para diferenciar las habilidades de aprendizaje entre individuos es que el efecto de refuerzo que causan la comida, el agua, etc., no es una propiedad intrínseca del estímulo sino del propio animal. Esto es, cada individuo de una especie asigna un valor a dichos estímulos en función de su utilidad para la

[2] No se insistirá aquí en la importancia que tienen los juegos en el desarrollo de algunos animales y la íntima relación de éstos con el aprendizaje.

[3] Un caso aparte es el caso de la reprodución de canciones (notas) por las aves mediante la imitación.

[4] Se ha escogido el término *impronta* para traducir el término inglés *imprinting* que, a su vez es la traducción de la voz alemana *Prägung*. La primera referencia escrita sobre este fenómeno se debe a Plinio el Viejo.

[5] Particularmente aquellas aves que recién nacidas abandonan el nido tras romper el huevo.

[6] En algunas especies de patos y pollos este período es efectivo de 13 a 16 horas después de romper el cascarón.

[7] Los investigadores están interesados en el impacto de la impronta en el desarrollo de las actitudes sociales, sexuales y paternales en la vida adulta del individuo.

supervivencia. En particular, la capacidad de adaptación de estos valores puede ser observada en los animales en cautividad, que parecen moverse bajo el estímulo de la *novedad* que les causa su entorno. La *exploración* se ha mostrado como una de las conductas más gratificantes para los individuos y que más influye en el aprendizaje. Esta conducta tiende a cesar cuando no aparecen nuevos cambios (estímulos externos) en el entorno. Esto sugiere que la conducta de exploración, que un estímulo novedoso elicita, termina como resultado de ciertos cambios internos interpretables como que el estímulo ha perdido su novedad.

1.1.3 Aprendizaje automático

El aprendizaje se refiere, como se ha visto, a un amplio espectro de situaciones en las cuales el *aprendiz* incrementa su conocimiento o sus habilidades para cumplir una tarea. El aprendizaje aplica inferencias a determinada información para construir una representación apropiada de algún aspecto relevante de la realidad o de algún proceso.

Una metáfora habitual en el área del aprendizaje automático – dentro de la Inteligencia Artificial – es considerar la resolución de problemas [8] como un tipo de aprendizaje que consiste – una vez resuelto un tipo de problema – en ser capaz de reconocer la situación problemática y reaccionar usando la estrategia *aprendida*. Actualmente la mayor distinción que se puede trazar entre un animal y un mecanismo de resolución de problemas es que ciertos animales son capaces de mejorar su actuación, en un amplio conjunto de tareas, como resultado de haber *solucionado* un cierto problema.

Se asume, en este enfoque, que un agente autónomo[9] debe tener la capacidad de realizar una misma tarea de varias maneras, si es posible, y dependiendo de las circunstancias. Debe ser capaz de tomar decisiones sobre cuál es el curso más apropiado que debe seguir la resolución de un problema y modificar estas decisiones cuando las condiciones así lo requieran. Por esto, uno de los objetivos centrales de este área es construir sistemas (agentes) que sean capaces de adaptarse – dinámicamente y sin un entrenamiento previo – a situaciones nuevas y aprender como resultado de resolver el problema (o problemas) que estas situaciones presentan.

El aprendizaje automático, también llamado *aprendizaje artificial* **[PLAZ92]**, es un área de intéres muy desarrollada en la IA. En otras áreas afines como la biología **[DAWK89]**, la psicología y la filosofía también se ha investigado la naturaleza de la habilidad de aprender referida a sistemas biológicos y al hombre en particular.

Comprender el aprendizaje –por ejemplo, el proceso de aprendizaje humano – de manera que permita reproducir aspectos de dicha conducta utilizando un ordenador es una meta muy ambiciosa. Aún cuando algunos investigadores han explorado también esta posibilidad utilizando como modelo otros animales, los resultados son todavía pobres.

Como ya se ha visto, a*prendizaje* es un término muy general que denota la forma, o formas, en la cual un animal (o una máquina) aumenta su conocimiento y mejora sus capacidades de actuación (*performance*) en un entorno. De esta manera, el proceso de aprendizaje puede ser visto como un generador de cambios en el sistema que aprende – que por otra parte ocurren lentamente, adaptativamente – y que pueden ser revocados o ampliados. Estos cambios se

[8] Esta metáfora tan rica se debe a H.Simon [SIMO89]

[9] Se utiliza el término agente autónomo para indicar un individuo, un programa, un artefacto, un robot, *etc* que esté bajo observación cuando realiza una tarea cognitiva identificable. En nuestro caso, aprender.

refieren no sólo a la mejora de las capacidades y habilidades para realizar tareas sino que también implican modificaciones en la representación de hechos conocidos.

En este contexto, se dice que un *sistema que aprende de forma automatizada (o aprendiz)* es un artefacto (o un conjunto de algoritmos) que, para resolver problemas, toma decisiones basadas en la experiencia acumulada – en los casos resueltos anteriormente – para mejorar su actuación. Estos sistemas deben ser capaces de trabajar con un rango muy amplio de tipos de datos de entrada, que pueden incluir datos incompletos, inciertos, ruido, inconsistencias, *etc.*

Nuestra primera caracterización del proceso de aprendizaje automático es:

$$Aprendizaje = Selección + Adaptación$$

Visto así, el *aprendizaje automático* es un proceso que tiene lugar en dos fases. Una en la que el sistema elige (*selecciona*) las características más relevantes de un objeto (o un evento), las compara con otras conocidas – si existen – a través de algún proceso de cotejamiento (*Pattern Matching*[10]) y, cuando las diferencias son significativas, *adapta* su modelo de aquel objeto (o evento) según el resultado del cotejamiento. La importancia del *aprendizaje*, como se ha dicho, reside en que sus resultados habitualmente se traducen en mejoras en la calidad de actuación del sistema. Un sistema artificial que *aprende* puede emplear técnicas muy diversas para aprovechar la capacidad de cómputo de un ordenador, sin importar su relación con los procesos cognitivos humanos. Estas técnicas incluyen métodos matemáticos muy sofisticados (ver el capítulo 2, dedicado al aprendizaje inductivo), métodos de búsqueda en grandes bases de datos, *etc.*, que requieren la creación (o modificación) de estructuras de representación del conocimiento adecuadas para agilizar la identificación de los hechos relevantes.

Una de las motivaciones más importantes en el diseño y construcción de sistemas de aprendizaje automático reside en el hecho de que en muchos dominios la experiencia es escasa, y la codificación del conocimiento que la describe es limitada, fragmentaria y, por lo tanto, incompleta o casi inexistente. Además, dotar a un agente de todo el conocimiento necesario es una tarea muy compleja, costosa, que toma mucho tiempo y en la cual la eliminación de los posibles errores introducidos es difícil y requiere una atención especializada. En el caso de los humanos son necesarios 5 ó 6 años para aprender las habilidades motoras básicas y los rudimentos del lenguaje, y entre 12 a 20 años para manipular conceptos complejos, aprender un oficio, las convenciones culturales e históricas, *etc.* Además, el aprendizaje en los humanos es *personalizado*.

Una línea de investigación importante es aquella que explora la *calidad* de lo aprendido en función de aquello que el *aprendiz* sabe: cómo lo que es sabido por el agente puede conducir, vía inferencia **[CUMM91]**, circumscripción **[NUÑE91a]**, *etc.*, a la obtención de nuevo conocimiento. Otra de las motivaciones es el intento de aprovechar la capacidad de cálculo de los ordenadores como una ayuda a la toma de decisiones (*decision-making*). Si además el sistema es capaz de aprender a partir de los casos tratados, como hace un humano, entonces el valor de la herramienta aumenta considerablemente.

[10] Pattern Matching también puede ser traducido como cotejamiento de esquemas.

1.2 Reseña histórica

Una breve reseña histórica del desarrollo de las investigaciones sobre el aprendizaje automático en IA es la siguiente:

1. Entusiasmo inicial (1955-1965)

   - Aprendizaje sin conocimiento de respaldo
   - *Neural Modelling*
   - Aprendizaje evolutivo

   Entre los hitos más significativos de este período se hallan los *perceptrones* **[MINS67], [MINS69a]**, las nociones de autoorganización, autoestabilización y el cotejamiento de patrones como herramientas básicas en los procesos de aprendizaje. Los sistemas desarrollados en esta época se caracterizan por la carencia, casi absoluta, de conocimiento de respaldo o conocimiento inicial. El sistema *ANALOGY* **[EVAN68]**, que será estudiado en el capítulo 3, debe ser considerado como una transición.

2. Etapa obscura (1965-1976)

   - Adquisición simbólica de conceptos **[WINS70]**
   - Adquisición del lenguaje

   En la segunda época los problemas se trasladaron esencialmente a la obtención de esquemas de representación con el fin de asegurar la correcta adquisición de *nuevos* conocimientos (*i.e.* saber cuáles son los ítems de conocimiento tras un proceso de aprendizaje). Asociado a esta tendencia se diseñaron estructuras, llamadas jerarquías, para organizar los conceptos (ver la sección 2.5). Este tipo de aprendizaje consiste en la inferencia y asimilación de nuevo material compuesto de conceptos, leyes generales, procedimientos, etc. Estos sistemas descansan sobre la existencia de un "profesor" humano que supervisa el aprendizaje.

3. Renacimiento (1976-1986)

   - Exploración de diferentes estrategias
   - *Knowledge-intensive Learning*
   - Aplicaciones exitosas

   En esta época se explora cómo combinar técnicas simbólicas de aprendizaje para mejorar la "calidad" de lo aprendido. Se explora con el uso de grandes cantidades de conocimiento de respaldo. Aparecen las primeras aplicaciones "reales".

4. Desarrollo (1986-Actualidad)

   - Aprendizaje conexionista **[RUME86c]**
   - Sistemas multiestrategia
   - Comparaciones experimentales

- Expectativas de aplicaciones prácticas relevantes
- *Nouvelle AI*

Una posible enumeración de las áreas de investigación más activas actualmente en este campo es la siguiente:

- Modelos neurales y modelos cognitivos **[ARBI91]**
- Modelos computacionales teóricos **[VALI84]**
- Algoritmos de aprendizaje evolutivos **[KOZA92]**
- Sistemas autónomos

Un agente tiene la capacidad de *aprender* cuando de forma autonóma es capaz de realizar nuevas tareas, adaptarse a los cambios de su entorno, o mejorar su actuación en tareas ya conocidas. La pregunta entonces no es saber si el aprendizaje automático (o artificial) es posible o no, sino cuáles son los métodos que efectivamente pueden conducir al aprendizaje (*i.e.* ¿cuáles son los procesos? ¿sobre qué partes de las estructuras de conocimiento operan? *etc.*).

1.2.1 Paradigmas del aprendizaje automático

Según el tipo de selección y adaptación (transformación) que un sistema realiza sobre la información disponible es posible identificar varios paradigmas del aprendizaje automático. Esta clasificación ha evolucionado rápidamente en la última década.

- **Aprendizaje deductivo**

  Este tipo de aprendizaje se realiza mediante un secuencia de inferencias deductivas usando hechos o reglas conocidos. A partir de los hechos conocidos nuevos hechos o nuevas relaciones son lógicamente derivadas. En este tipo de sistemas la monotonicidad de la teoría definida por la base de conocimientos es importante.

- **Aprendizaje analítico** Los métodos usados en este tipo de aprendizaje intentan formular generalizaciones después de analizar algunas instancias en términos del conocimiento del sistema. En contraste con las técnicas empíricas de aprendizaje – que normalmente son métodos basados en las similitudes – el aprendizaje analítico requiere que se proporcione al sistema un amplio conocimiento del dominio. Este conocimiento es usado para guiar las cadenas deductivas que se utilizan para resolver nuevos problemas. Por tanto, estos métodos se centran en mejorar la eficiencia del sistema, y no en obtener nuevas descripciones de conceptos, como hace el aprendizaje inductivo (p.e. **[MITC86]**).

- **Aprendizaje analógico** Este tipo de aprendizaje, comentado en el capítulo 3, intenta emular algunas de las capacidades humanas más sorprendentes: poder entender una situación por su parecido con situaciones anteriores conocidas, poder crear y entender metáforas o resolver un problema notando su posible *semejanza* con otros vistos anteriormente adaptando (transformando) de forma conveniente la solución que se encontró para

esos problemas (p.e. **[CUMM91]**, **[WINS82]**, **[CREI88]**, **[EVAN68]**). Este tipo de sistemas requiere una gran cantidad de conocimiento. Algunos autores consideran que el aprendizaje analógico es una especialización del aprendizaje por explicación que será discutido en 3.

- **Aprendizaje inductivo**

  Es el paradigma más estudiado dentro del aprendizaje automático. Normalmente, estos sistemas carecen de una teoría del dominio, es decir, no conocen *a priori* los objetos con los que tratan o su cantidad. Trata problemas como inducir la descripción de un concepto a partir de una serie de ejemplos y *contraejemplos* del mismo (*i.e.* **[DIET81]**, **[MORE92]**), o determinar una descripción jerárquica o clasificación de un grupo de objetos (p.e. **[BÉJA92]**).

- **Aprendizaje mediante descubrimiento** El tipo de *Descubrimiento* es una forma restringida de aprendizaje en la cual un agente adquiere conocimientos sin la ayuda de un *profesor*. Este proceso ocurre cuando no existe ninguna "fuente" disponible que posea el conocimiento que el agente busca **[LENA84]**. Un tipo particular de *Descubrimiento* se lleva a cabo cuando un agente intenta agrupar objetos que supone del mismo conjunto.

- **Algoritmos genéticos**

  Los algoritmos genéticos están inspirados en las mutaciones y otros cambios que ocurren en los organismos durante la reproducción biológica de una generación a la siguiente y en el proceso de selección natural de Darwin. El problema principal que trata de resolver es el *descubrimiento de reglas* y la dificultad mayor con que se encuentra es la *asignación de crédito* a las mismas. Este último punto consiste en valorar positiva o negativamente las reglas según lo útiles que sean al sistema. Esta valoración será la que determine qué regla aplicar para resolver un problema determinado (p.e. **[HOLL92]**, **[DAVI87]**). Este tipo de aprendizaje se trata en el capítulo 7.

- **Conexionismo** Otra manera de concebir un sistema de aprendizaje automático es el denominado enfoque *conexionista*. En esta aproximación el sistema es una *red* de nodos interconectados, que tiene asociada una regla de propagación de valores, y cuyos arcos están etiquetados con pesos. Ante un conjunto de ejemplos el sistema reacciona modificando los pesos de los arcos. Se dice que el sistema *aprende* si *adapta* los pesos de las conexiones de tal manera que le lleven a dar la salida *correcta* ante todas (o la mayoría) de las entradas que se le ofrezcan **[ARBI91]**. Ver el capítulo 5.

Otra posible clasificación de los métodos de aprendizaje explorados en IA, considerando el tipo de estrategia y las ayudas que recibe un sistema de aprendizaje, es:

- **supervisados**

  La suposición fundamental de este tipo de método es que los ejemplos proporcionados como entrada son *necesarios* para cumplir las metas del aprendizaje. Es como aprender con un *profesor*. En este tipo de método se dan ejemplos y se especifica de qué concepto lo son.

- **no supervisados**

Son diseñados para desarrollar nuevos conocimientos mediante el *descubrimiento* de regularidades en los datos (*data-driven*). Estos métodos no están dirigidos por las metas (*goal-driven*).

- **mediante refuerzos**

  Este método de aprendizaje está a medio camino entre los dos anteriores. Al sistema se le proponen problemas que debe solucionar. El aprendizaje se realiza únicamente con una señal de refuerzo proporcionada por un profesor o por el entorno como indicador de si se ha resuelto correctamente el problema.

La figura 1.1 muestra una clasificación de los diferentes métodos de aprendizaje que puede ser considerada como apta para explicar al mismo tiempo el desarrollo histórico de la investigación en aprendizaje automático y para comprender las interrelaciones que existen entre los diferentes enfoques que han predominado en este área.

1.2.2 Medidas de actuación

En los siguiente capítulos se discutirán diferentes sistemas como ejemplos de los diferentes paradigmas de aprendizaje, diseñados en diversas etapas de la IA y, por consiguiente, con diferentes concepciones arquitéctonicas. Entonces, cabe preguntarse cómo se puede evaluar la actuación de un sistema para compararlo con otros. En este sentido hay pocos trabajos realizados aunque existen algunos intentos de sistematizar conjuntos de pruebas (*benchmarks*) que permitan desarrollar comparaciones **[THRU91]**. Las caracteristícas que se indican a continuación no son las únicas que pueden medirse pero dan una idea de la tendencia que se sigue en la comunidad:

Generalidad Una de las medidas de actuación de un sistema que aprende es la *generalidad* o alcance del método. Esta da idea de cuán fácil es adaptar el método a diferentes aplicaciones en dominios diversos. El ideal de generalidad es un sistema que pueda autoajustarse según los requerimientos de la tarea o del entorno. En el otro extremo se encuentran aquellos sistemas que sólo se aplican a un único dominio. El término medio es que los sistemas de aprendizaje trabajen aceptablemente bien en algunos dominios.

Eficiencia La eficiencia de un método puede ser medida como el coste temporal requerido para construir las estructuras de representación que permitan describir el objetivo (*target concept*) a partir de unas estructuras de representación iniciales.

Robustez Es la medida de la habilidad del sistema para trabajar con conjuntos de ejemplos de entrenamiento que pueden contener *ruido* e información parcialmente equivocada. Un sistema robusto también debe ser capaz de construir hipótesis que puedan ser modificadas ante la llegada de nuevas evidencias[11].

Eficacia Es una medida de la capacidad global del sistema y función de las medidas anteriores. La idea es generar un cierto orden entre los sistemas que permita asertar que el sistema X es más eficaz que otro Y.

[11] Esta característica implica una cierta no monotonía en las operaciones de inferencia.

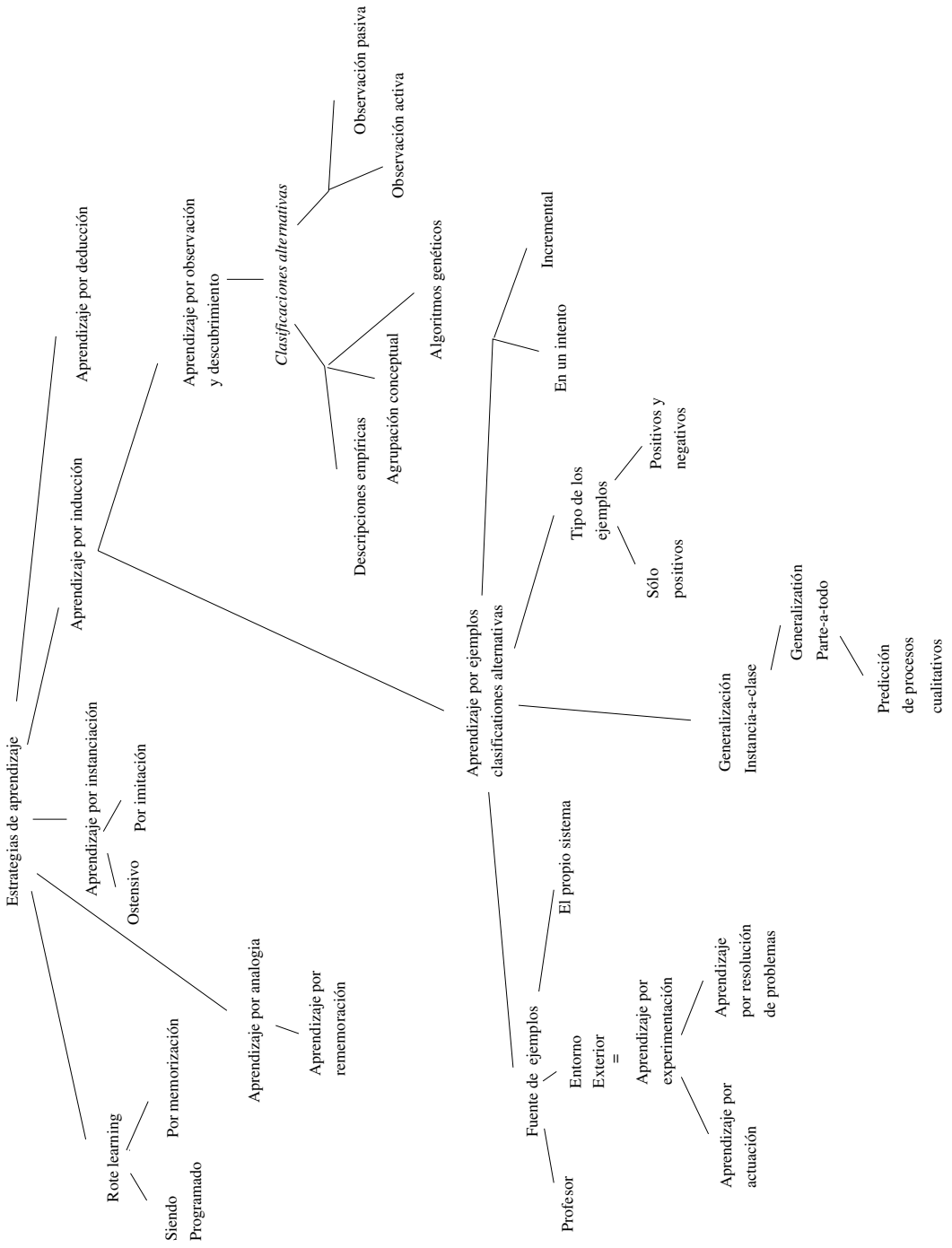


Figura 1.1: Clasificación de los métodos de aprendizaje

Otras medidas pueden estar relacionadas con la *facilidad de implementación*, pero requieren un conjunto de métricas asociadas propiamente a la corrección y complejidad del sistema y los tipos abstractos de datos que se emplean y que, en cierta medida, hay que relacionar con las características propias de las estructuras de representación.

Otros autores prefieren medidas operacionales [12] como la *validez* y la *abstracción* que dependen de la tarea que se tiene que realizar y el conocimiento de respaldo inicial. También se ha estudiado la *utilidad* en términos de la ganancia de conocimientos y la eficiencia del proceso.

La ganancia de conocimiento, medida como la diferencia del conocimiento actual y el inicial, es relativamente fácil de medir en algunos tipos de aprendizaje como por ejemplo en el inductivo donde el conocimiento inicial es nulo, o casi nulo. Lo mismo ocurre en el caso del aprendizaje analógico donde la restricción y limitación del dominio permiten conocer el estado inicial y final.

La eficiencia del proceso de aprendizaje no siempre resulta fácil de medir. Existen algunos intentos aislados como los apuntados para `PRODIGY` [MINT89] donde se evaluan la aplicabilidad de las reglas aprendidas, la frecuencia de su aplicación y la ganancia - en términos de eficiencia - de su aplicación. En el caso de métodos de aprendizaje SBL también existe algún intento de comparación, ver por ejemplo [MORE92]. Para el *conceptual clustering* existen algunas medidas sobre la actuación de los algoritmos [GENA89] y comparaciones entre diversos algoritmos [BÉJA93], [ROUR94].

1.3 Estrategias elementales de aprendizaje

Si hacemos referencia a la figura 1.1 las estrategias de aprendizaje más elementales son:

- Aprendizaje memorístico (*Rote learning*)
- Aprendizaje por instrucción (*Direct instruction*)
  - Aprendizaje por imitación (*Learning by imitation*)
  - Aprendizaje ostensivo (*Learning by being told*)

Este tipo de métodos depende en gran medida de la *calidad* de las descripciones suministradas[13]. Como consecuencia, requieren también buenos métodos para la comparación, total o parcial, entre descripciones.

En el *Aprendizaje memorístico* es posible resaltar dos características importantes: a) la buena organización al almacenar la información, y b) un acceso rápido a ésta. La *generalización* en este tipo de métodos es más bien pobre o inexistente. Habitualmente se emplea como método de inferencia la copia directa de los ítems de la base de conocimiento y se les usa así, repitiendo *lo sabido*. Un ejemplo típico (en los humanos) de estos métodos es el aprendizaje de las tablas de multiplicar.

Una estrategia de aprendizaje ligeramente más compleja es aquella que se sigue cuando se *aprende por instrucción*. Aquí la cantidad de inferencia necesaria es mayor, ya que se necesita

[12] También pueden ser consideradas como medidas cualitativas.
[13] Por un profesor que se considera justo (*fair*) y no engaña al aprendiz.

transformar el conocimiento a una forma operacional (quizá un algoritmo) antes de integrarlo en la base de conocimientos. Los humanos empleamos esta estrategia cuando un profesor presenta un conjunto de hechos de una manera estructurada e indica la finalidad de cada uno de ellos en un proceso sencillo. La más común, también llamada *Aprendizaje ostensivo*, es aquella en la que el profesor señala, con el dedo índice, un objeto y pronuncia una palabra que lo nombra[14]. Es trabajo del aprendiz el *asociar* el nombre con el objeto (y no con el dedo). El aprendizaje basado en ejemplos puede convertirse en una tarea compleja y merece un tratamiento especial, ver 2.2.1.

1.4 Organización del texto

Este texto ha sido desarrollado en su mayor parte por miembros del grupo de *Sistemas basados en el conocimiento y Aprendizaje* de la sección de *Inteligencia Artificial* del *Departament de Llenguatges i Sistemes Informàtics* de la *Universitat Politècnica de Catalunya*. También han colaborado en la elaboración del mismo R.Gavaldà, de la sección de *Informática Teórica* del mismo departamento, B.López (de la *Universitat Rovira i Virgili*) y E.Armengol, investigadora del *Institut d' Investigació en Intel.ligència Artificial* de Blanes, Girona. Su objetivo es servir de texto de la asignatura *Aprendizaje* de la Ingeniería en Informática, y como texto de consulta en varias asignaturas del programa de doctorado en Inteligencia Artificial del *Departament de Llenguatges i Sistemes Informàtics*.

La organización del texto es la siguiente. En este primer capítulo se ha hecho un recorrido histórico del área y se han introducido los paradigmas dominantes en ella. A continuación se han comentado las estrategias elementales de aprendizaje.

El capítulo 2 está dedicado al aprendizaje inductivo. Empieza con una clasificación de los diferentes tipos de aprendizaje inductivo, y desarrolla en detalle varias de las subáreas de este campo. Concretamente, se describen y comparan los principales métodos de adquisición de conceptos (inducir la descripción de un concepto a partir de ejemplos y contraejemplos del mismo). Después se desarrolla la teoría de los árboles de decisión, estructuras que sirven para representar los procesos de decisión involucrados en una clasificación. Con estas estructuras aparece el problema de la relevancia de los atributos: saber qué características son o no importantes en la descripción de un concepto. A este problema también se le dedica una sección. Para acabar el capítulo, se comentan los principales métodos en el área de la formación de conceptos. Este capítulo está basado parcialmente en las tesinas de A.Moreno y Ll. Belanche, y en el trabajo de investigación sobre métodos de aprendizaje no supervisado llevado a cabo por J.Béjar.

En el capítulo 3 se presentan los conceptos de razonamiento y aprendizaje por analogía, a partir del trabajo exploratorio en el área llevado a cabo por J.M.Gimeno. Se describen diferentes estrategias de analogía, como la transformacional y la derivacional, ilustrándolas con ejemplos concretos de sistemas que utilizan estas técnicas. También se comenta un modelo que unifica las diferentes formas de analogía.

En el capítulo 4 se desarrolla el aprendizaje deductivo, haciendo especial enfásis en EBL (*Explanation Based Learning*, aprendizaje basado en explicaciones). E. Armengol describe las características de este tipo de métodos, analiza sus problemas asociados y estudia diversos

[14] Esta acción es típica de una lección cuando se aprende una lengua.

sistemas desarrollados bajo este paradigma (STRIPS, EBG, SOAR, PRODIGY). El capítulo acaba con una comparación con otros tipos de aprendizaje automático.

En el capítulo 5 Ll. Belanche comenta la visión conexionista del aprendizaje. Se describe el modelo biológico en el que se inspira este paradigma, y se da una perspectiva histórica del campo. Tras dar las nociones básicas necesarias para la comprensión del capítulo se explican modelos neuronales concretos como el perceptrón, las redes de Hopfield o las memorias asociativas bidireccionales, describiendo con profundidad los algoritmos necesarios en estos modelos, como el algoritmo de *backpropagation* o el de *counterpropagation*.

M.Martín describe en el capítulo 6 el aprendizaje por refuerzo. Este tipo de aprendizaje se basa en un maestro que vaya dando respuestas (refuerzos) positivas o negativas a un sistema según el comportamiento que presente delante de un problema. Utilizando estos refuerzos, el sistema debe acabar aprendiendo el comportamiento adecuado para cada situación. Se da una visión global de este tipo de aprendizaje, y se describen con detalle las dos famílias principales de métodos: con refuerzo inmediato o con refuerzo retardado. Se explican los principales algoritmos en este campo, como son el algoritmo lineal de premio-castigo, las diferencias temporales o el *q-learning*.

En el capítulo 7 se desarrolla la teoría de los algoritmos genéticos por parte de J.M.Gimeno. Se describe lo que es un algoritmo genético y se razona el porqué de su funcionamiento correcto delante de determinados tipos de problemas. También se comenta un sistema concreto que utiliza algoritmos genéticos (GABIL). Se acaba el capítulo analizando los sistemas clasificadores y el tema de la programación genética.

En el capítulo 8, B. López explica sistemas que utilizan razonamiento basado en casos. En este tipo de aprendizaje los sistemas mantienen una base de conocimientos donde almacenan casos que han resuelto previamente, y utilizan esta información para resolver los problemas que se les planteen posteriormente.

Para acabar, en el capítulo 9 R.Gavaldà describe cómo se ha enfocado el tema del aprendizaje automático desde su vertiente teórica. Se dan las definiciones básicas para poder entender el capítulo y se describen los tres principales paradigmas en la teoría del aprendizaje algorítmico: el aprendizaje mediante preguntas, el aprendizaje PAC y el aprendizaje con errores acotados. Todos estos modelos se ilustran con algoritmos concretos de aprendizaje.

1.5 Resumen

El aprendizaje es un proceso cognitivo mediante el cual un agente *adquiere* conocimiento, o aumenta la calidad y/o cantidad de su conocimiento, o mejora sus habilidades para realizar una tarea. En algunos casos dicho conocimiento es poseído por otros agentes que pueden servir como *profesores*.

Adquirir un concepto o una conducta motora o intelectual está considerado como un caso estándar de los procesos de aprendizaje, pero ni los conceptos ni las conductas aprendidas han de ser *justificadas* por el agente para contar como aprendidas. En otras palabras, el agente (o el proceso) no tiene que *elaborar* una explicación del proceso ni presentar (o justificar) las suposiciones (si hizo alguna) utilizadas en el proceso. Aún más, tampoco debe (en muchos casos) preocuparse por la veracidad (o utilidad) de lo aprendido.

El aprendizaje puede ser entendido como una tarea orientada a la creación y mantenimiento de un modelo interno del mundo. En general, esta tarea tiene dimensiones gigantescas pero, afortunadamente, existen variantes muy simples que pueden ser estudiadas *fácilmente*. Entre éstas se pueden mencionar aquéllas que ven al aprendizaje como un tipo especial de mecanismo de resolución de problemas en el que las tareas de búsqueda[15] (en un espacio) y de representación del conocimiento son muy relevantes.

En este proceso es posible identificar los siguientes componentes:

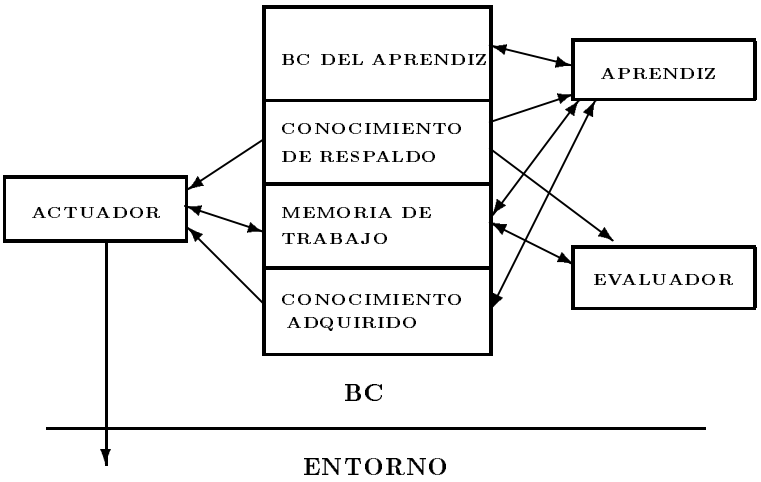


Figura 1.2: Proceso general de aprendizaje

- El **aprendiz** (o agente), que realiza la tarea de aprender.
- La **base de conocimientos**, que contiene el material (conocimiento) necesario para realizar una tarea determinada.[16].
- El **elemento de actuación**, aquello que actúa sobre el dominio a partir del contenido de la base de conocimientos del aprendizaje y del conocimiento de respaldo.
- El **mecanismo de evaluación**, que permite conocer el nivel de actuación del agente.
- El **contexto** (o entorno), de donde se obtienen las secuencias de entrenamiento.
- El **conocimiento de respaldo** contiene la representación de lo que se *sabe* sobre los objetos, sus relaciones, las acciones, sobre los eventos y sobretodo otros agentes. También puede incluir conocimiento, como heurísticas de resolución de problemas.

El agente (ver figura 1.2) debe disponer de al menos un algoritmo (o estrategia) de aprendizaje y de alguna capacidad de cómputo. Además, si le es posible aumentar su conocimiento su conducta y sus capacidades se verán ampliadas.

[15] Este es caso de los métodos de aprendizaje explicados en la sección 2.2

[16] Algunos autores prefieren decir: la información *necesaria*. Aún cuando no siempre esté disponible.

La representación del conocimiento por motivos de simplicidad está relegada a un segundo plano: se asume que el conocimento está representado o que es obtenible del entorno. Pero – como se ha dicho – de la calidad del conocimiento disponible depende la calidad de lo aprendido. En este sentido, dos aspectos a los que se enfoca cierta parte del esfuerzo de los sistemas de aprendizaje automático son:

1. Conocer el tipo de errores que pueden ocurrir en la estructura de representación.
2. Cómo detectar, localizar y corregir dichos errores.

Aparte de las clasificaciones que se han dado (ver la figura 1.1), uno puede intentar clasificar los métodos de aprendizaje según la cantidad de conocimiento que poseen. Entre los llamados *weak–methods* están los algoritmos genéticos o las redes neuronales (ver el capítulo 5). Estos métodos se caracterizan por ser extremadamente mecánicos. En el lado de los llamados *rich–knowledge methods*, se encuentran aquellos que se describen en el capítulo 2, basados en heurísticas de búsqueda y grandes cantidades de conocimiento de respaldo.

Capítulo 2 Aprendizaje inductivo

2.1 Definición

El proceso de aprendizaje inductivo consiste en la adquisición de nuevo conocimiento después de realizar inferencia inductiva (inducción) sobre los datos proporcionados por el entorno o por un maestro. Este proceso se puede caracterizar ([**NILS80**], [**MITC82**], [**MICH84a**]) como una búsqueda heurística en un espacio de estados (ver figura 2.1), donde:

- Los estados son descripciones simbólicas de mayor o menor generalidad. El estado inicial son los datos de entrada.
- Los operadores son reglas de inferencia, fundamentalmente *reglas de generalización* (pasan de una descripción simbólica a otra más general) y *reglas de especialización* (transforman una descripción en otra más particular).
- El estado final es una aserción con las siguientes propiedades:
  - **Implica los datos de entrada**. Esta condición garantiza que el resultado que se obtiene procede de un proceso inductivo, y no de realizar deducción sobre los datos disponibles.
  - **Satisface el conocimiento de respaldo del problema**. Por conocimiento de respaldo (*background knowledge*) se entiende el conocimiento que tiene el programa sobre el problema que está tratando de solucionar. Esta condición pide que la inducción obtenida no viole ninguna de las reglas que hayan de cumplir los objetos dentro del dominio que se esté tratando. Por ejemplo, si se está trabajando en el dominio del mundo de los bloques, uno de los hechos que podría tener el programa en su conocimiento de respaldo es que no es posible que haya ningún objeto encima de un círculo.
  - **Maximiza el criterio de preferencia** que se aplique para valorar la *calidad* de las descripciones encontradas. Este criterio puede ser encontrar la descripción más específica posible, o la más simple, o la que contenga menos descriptores, etc.

Las reglas de generalización que se usan son de dos tipos ([**MICH84a**]):

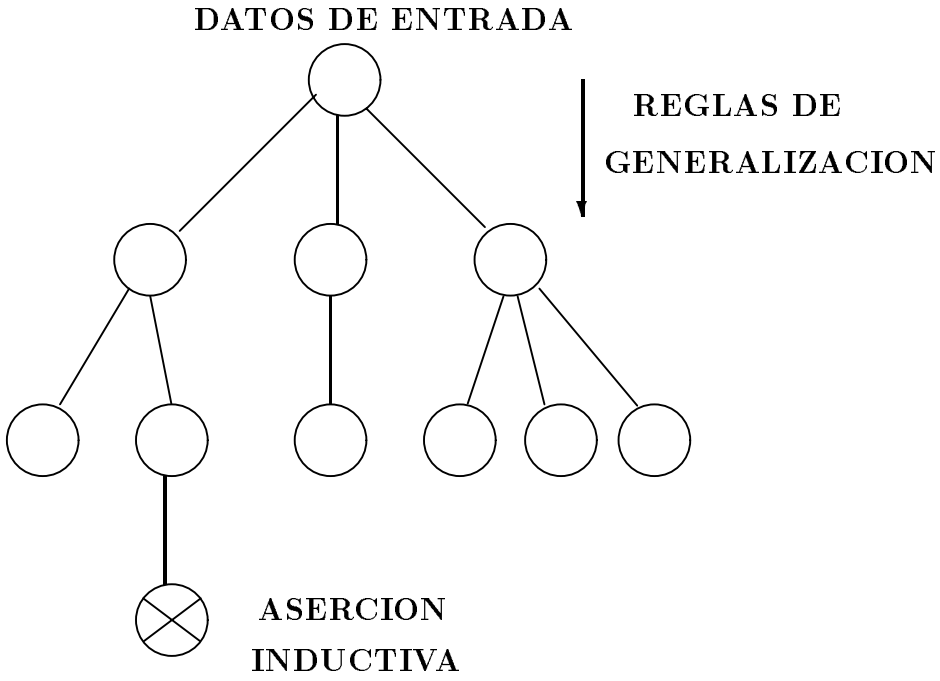


Figura 2.1: Inducción como búsqueda

- **Reglas de selección** Son aquellas reglas en las que todos los descriptores que aparecen en la expresión generalizada ya estaban presentes en las descripciones iniciales del concepto. Las más habituales son:

  - **Supresión de conjuntandos**
    Consiste en eliminar un elemento dentro de una conjunción, obteniendo de esta forma una expresión más general (a es más general que $a \wedge b$). P.e., es una generalización pasar de "*Hay un objeto pesado de color verde encima de la mesa*" a "*Hay un objeto verde encima de la mesa*".
  - **Adición de disyuntandos**
    Consiste en añadir un elemento dentro de una disyunción ($a \vee b$ es más general que a). P.e., se podría pasar de "*Hay un objeto verde encima de la mesa*" a "*Hay un objeto verde o azul encima de la mesa*".
  - **Cerrar intervalos**
    Si se tienen dos descripciones de la misma clase que difieren en el valor de un sólo descriptor lineal, se pueden reemplazar por una única descripción en la cual la referencia del descriptor sea el intervalo entre estos dos valores. Por ejemplo, si en una descripción se tiene "*Hay un objeto de peso 2*" y en otra "*Hay un objeto de peso 7*", se pueden generalizar a "*Hay un objeto con un peso entre 2 y 7*".
  - **Cambio de constantes por variables**
    Consiste en substituir alguna de las constantes que aparezcan en la descripción de un concepto por una variable cuantificada universalmente, obteniendo así una

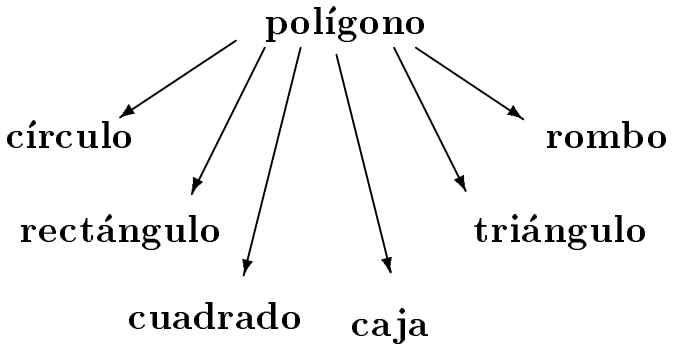


Figura 2.2: Árbol de generalización del atributo *forma*

expresión más general. Aplicando esta regla se podría pasar de "*Este libro de Ludlum es muy bueno*" a "*Todos los libros de Ludlum son muy buenos*".

- **Subir el árbol de generalización**
  Un atributo de tipo estructurado es aquel cuyo dominio se puede representar de forma jerárquica (con el llamado *árbol de generalización*). Si hay varias descripciones en las que un atributo de tipo estructural tiene diferentes valores, se pueden generalizar a una descripción en la que ese atributo tenga como valor el nodo más bajo del árbol de generalización que sea antecesor de esos valores. Por ejemplo, "*Hay un objeto cuadrado*" y "*Hay un objeto rectangular*" se pueden generalizar a "*Hay un polígono*", si se tiene un árbol de generalización como el que aparece en la figura 2.2.

- **Reglas constructivas** Estas reglas generan aserciones inductivas que contienen descriptores que no existían en las descripciones originales. Las más habituales son:
  - **Contar el número de objetos** que satisfacen una cierta condición. P.e., en el mundo de los bloques se podría incluir un nuevo descriptor unario **(Número-de-objetos-azules x)** que contara el número de bloques de color azul dentro de la escena tratada.
  - **Generar propiedades en una cadena de elementos** (objetos al principio, al final, en una posición determinada de la cadena). Por ejemplo, en el mundo de los bloques se pueden generar descripciones en las que aparezca el predicado **(CIMA x)** – que indica que **x** no tiene ningún objeto por encima – a partir de descripciones en las que sólo aparezca la relación **SOBRE**, relación binaria que indica que un objeto está sobre otro.
  - **Detectar dependencias** entre descriptores. P.e., si hay dos atributos *A1* y *A2* de tipo entero y son dependientes el uno del otro, se podrían crear nuevos atributos con valores como *A1* + *A2*, *A1* - *A2*, *A1* * *A2* o *A1* / *A2*. Esta técnica la utilizaba p.e. el sistema **Bacon** ([**LANG84**]).

Una descripción más amplia de diferentes reglas de generalización inductiva, tanto selectivas como constructivas, se puede encontrar en [**MICH93**].

Se puede describir un proceso de aprendizaje inductivo como un *método de búsqueda* (que incluye operadores y estado inicial), una *estrategia de control* (que incluye una heurística para recortar el espacio de búsqueda) y una *función de maximización*.

2.1.1 Tipos de aprendizaje inductivo

Se pueden distinguir **[MICH84a]** dos grandes tipos de aprendizaje inductivo:

- **Adquisición de conceptos** También se conoce como **aprendizaje a partir de ejemplos**. Se caracteriza porque hay un *profesor* que proporciona al programa la descripción de algunos objetos, ya clasificados en una o más clases (*conceptos*). La hipótesis que se induce puede ser vista como una regla de reconocimiento del concepto. Esto significa que si un objeto satisface las condiciones de la regla entonces representa al concepto dado.

  Algunos problemas tratados en este tipo de aprendizaje son:

  - Aprender la **descripción característica** de una clase de objetos, que especifica las propiedades comunes a todos los objetos conocidos de la clase (p.e. **[WINS70]**, **[HAYE78]**).
  - Aprender la **descripción discriminante** de una clase de objetos, que la distingue de un número limitado de clases diferentes (p.e. **[MICH80b]**).
  - Inferir **reglas de extrapolación** a partir de secuencias, capaces de predecir el siguiente elemento de una secuencia dada (p.e. **[DIET79]**).

- **Aprendizaje a partir de la observación** También es conocido como **generalización descriptiva**. Su objetivo es determinar una descripción general que caracterize un conjunto de observaciones.

  Algunos ejemplos de este tipo de aprendizaje son:

  - **Formular una teoría** que caracterice un conjunto de elementos (p.e. **[LENA84]**).
  - **Descubrir regularidades** en datos (p.e. **[LANG84]**).
  - **Determinar una descripción taxonómica** (clasificación) de una colección de objetos (p.e. **[MART91]**, **[BÉJA92]**). Este proceso también es conocido como *conceptual clustering*.

En el resto de este capítulo se describen en detalle técnicas utilizadas en algunos de estos campos. Dentro del campo de la adquisición de conceptos, se explican varios de los algoritmos clásicos empleados para obtener la descripción característica de un concepto a partir de instancias positivas (y a veces también negativas) del mismo. Estos algoritmos son los de Mitchell (espacio de versiones), Winston, Hayes-Roth, Vere y Michalski. A continuación se comentan los árboles de decisión, estructuras jerárquicas que permiten clasificar objetos a partir de los valores de sus atributos y de su clase correspondiente de la forma más eficiente posible, utilizando técnicas de Teoría de la Información. Finalmente, se describen las técnicas básicas de *conceptual clustering*, o clasificación de objetos descritos a través de pares (*atributo*, *valor*).

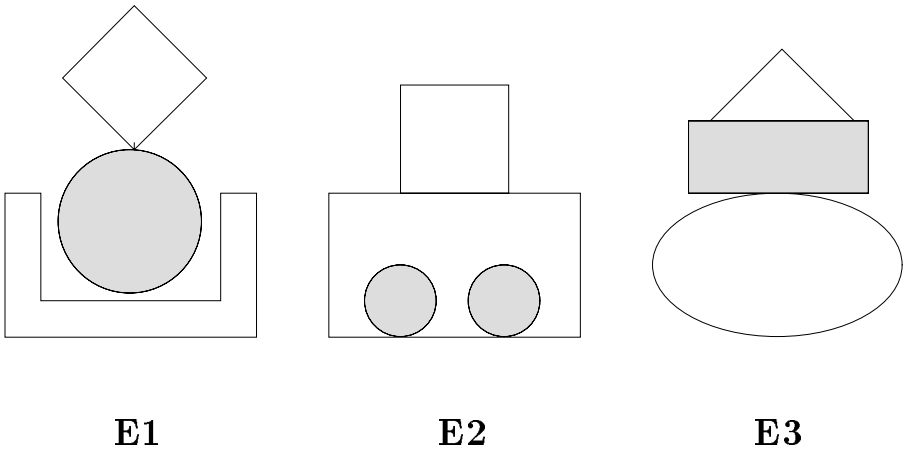


Figura 2.3: Ejemplo usado para la comparación de métodos

2.2 Métodos de adquisición de conceptos

En las siguientes secciones se describen algunos de los métodos clásicos de adquisición de conceptos. Todos ellos intentan inducir la descripción de una clase de objetos después de estudiar una serie de instancias positivas (y en algunos casos también negativas) de la clase.

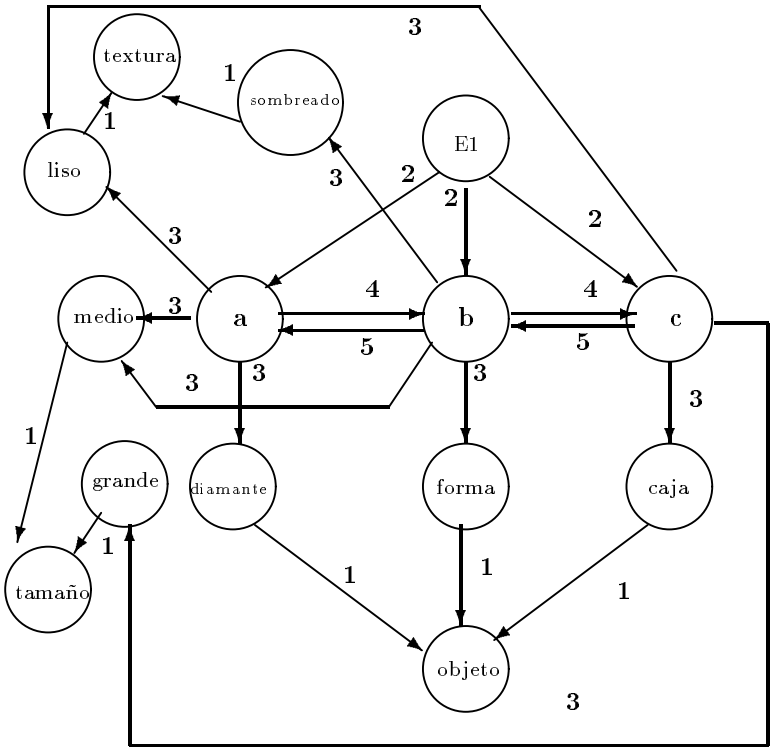
2.2.1 Método Winston

El método que se va a comentar a continuación fue desarrollado por Winston en la década de los 70 (**[WINS70]**, **[WINS75]**), y es reconocido como la base para toda una rama del aprendizaje automático conocida como **SBL** (*Similarity Based Learning*, aprendizaje basado en similitudes), y un clásico entre los métodos de aprendizaje. Este tipo de aprendizaje tiene por objetivo que la máquina aprenda la descripción de un concepto determinado después de haber visto una serie de ejemplos (y quizás también contraejemplos) del concepto objetivo. Se trata, por tanto, de un aprendizaje supervisado, guiado por un maestro que va mostrando a la máquina estos ejemplos y contraejemplos en el orden y forma más convenientes.

Una de las aportaciones fundamentales de este trabajo es la introducción de la noción de los **cuasiejemplos** (*near-misses*), que son una restricción sobre el tipo de ejemplos (o instancias) negativas que pueden ser suministradas al sistema. Así, dada una secuencia de entrenamiento compuesta de ejemplos positivos y negativos, estos últimos sólo pueden tener una diferencia significativa[1] con el concepto a aprender. Si éste es el caso los ejemplos negativos son llamados *cuasiejemplos*.

Los ejemplos negativos sirven, en general, para limitar la *extensión* del concepto que se está aprendiendo. Si además se tiene la certeza de que la diferencia entre los ejemplos positivos y

[1] En su último trabajo Winston **[WINS92]** denomina este tipo de aprendizaje *Learning by analyzing differences*.



1-Tipo-de **3-Tiene-la-propiedad**

2-Tiene-como-parte **4-Sobre**

5-Debajo

Figura 2.4: Descripción *a la Winston* del primer ejemplo

los cuasiejemplos es única, entonces el proceso de aprendizaje está dirigido, de alguna manera, a la búsqueda de esta diferencia.

El programa de Winston trabajaba en el dominio de objetos triédricos como los bloques, esferas, pirámides y objetos sencillos en general (el dominio de juguete de los bloques, ahora ya clásico en la Inteligencia Artificial). El primer problema que enfocó Winston fue cómo representar las escenas, y decidió usar redes semánticas, argumentando que son lo suficientemente sencillas y flexibles como para poder representar las escenas de forma adecuada. En estas redes semánticas cada objeto se representa en relación a otros objetos de la escena y a conceptos conocidos por el programa (p.e. *tamaño*). Se representan de la misma forma las relaciones entre objetos que las propiedades de los objetos. Como ejemplo, se puede ver en la figura 2.4 cómo se representaría la escena *E1* de la figura 2.3. El nodo *E1* representa toda la escena. Este primer ejemplo está formado por tres bloques (*a*, *b* y *c*). La descripción del ejemplo tiene dos partes:

- **Parte estructural**. En esta parte se describen las relaciones entre los objetos. En este

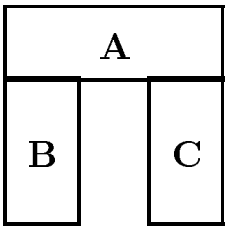


Figura 2.5: Arco

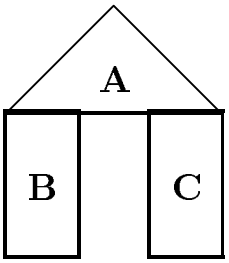


Figura 2.6: Arco con dintel triangular

ejemplo, a está *sobre* b y b está *sobre* c; la relación *debajo* se define por simetría.

- **Parte descriptiva**. Aquí se explicitan las propiedades de los objetos, es decir, los valores que tiene cada objeto en los atributos que se hayan considerado relevantes en el dominio tratado. En la figura 2.4 se dice, p.e., que el objeto a es un *diamante* (forma) *liso* (textura) *mediano* (tamaño).

Se ha de distinguir entre lo que es la **descripción de una escena** y lo que es el **modelo del concepto**. Un modelo es similar a una descripción normal, en el sentido de que tiene información sobre las diversas partes de una configuración, pero contiene muchas más cosas porque ha de indicar qué relaciones y propiedades han de cumplirse en cualquier ejemplo del concepto involucrado. Por ejemplo, se puede describir la figura 2.5 diciendo que A es un rectángulo que está soportado por los bloques B y C. Con esta descripción, ni la figura 2.6 ni la figura 2.7 serían reconocidas como arcos (en la figura 2.6 A no es un rectángulo, y en la figura 2.7 A no está soportado por B y C). En cambio, para cualquier humano la figura 2.6 sería un arco, ya que todos sabemos que la forma del dintel no es un aspecto importante en la definición de arco, mientras que sí es básico el hecho de que haya un objeto soportado por otros dos. Parece por tanto que una descripción ha de indicar qué relaciones son obligatorias y cuáles no son importantes antes de ser considerada como un modelo. Esto se puede lograr usando redes semánticas empleando relaciones tales como *soporta* (*support*) y *debe soportar* (*must-support*). El uso de este tipo de relaciones queda claro en la siguiente descripción del algoritmo **W** de Winston:

1. Tomar como modelo inicial la descripción de la primera instancia positiva del concepto. Llamar a esta descripción la **definición** del concepto.

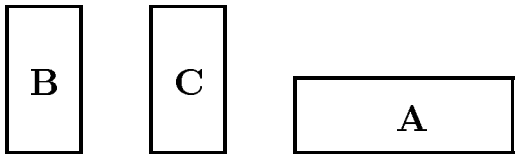


Figura 2.7: Ejemplo negativo de arco

2. Examinar la descripción de otras instancias positivas conocidas del concepto. *Generalizar* la definición del concepto para incluirlas.

3. Examinar las descripciones de los cuasiejemplos del concepto. *Restringir* (especializar) la definición del concepto para excluirlos.

- Los pasos 2 y 3 se van intercalando a medida que se van tratando ejemplos positivos y negativos del concepto objetivo.

El sistema posee un mecanismo de cotejamiento que permite comparar dos redes semánticas que son las respectivas descripciones estructurales del concepto y un ejemplo. Además, dependiendo de la información sobre el tipo de ejemplo (positivo o negativo), dirigirá su búsqueda para intentar *especializar* el concepto con ejemplos negativos o a *generalizarlo* en otro caso. Estas tareas se llevan a cabo sobre la parte estructural del concepto, así que sólo pueden realizarse mediante la modificación de las etiquetas que unen los nodos. El resultado de estas modificaciones es el modelo en evolución del concepto que se está aprendiendo. Con el fin de construir este modelo, Winston introduce dos heurísticas que actúan sobre las etiquetas: *require-link* (que exige que haya una determinada relación en la descripción) y *forbid-link* (que prohibe que haya una determinada relación en la descripción). Su aplicación es muy simple:

- *Require-link*: es una heurística empleada cuando el modelo del concepto que está siendo aprendido (en evolución) tiene una etiqueta k en un lugar donde un cuasiejemplo no. Entonces en la red semántica que representa el concepto esa etiqueta se transforma en *debe* (*must*).

- *Forbid-link*: esta heurística se aplica cuando un cuasiemplo tiene una etiqueta i en un lugar donde el modelo no. Entonces una etiqueta *no-debe* (*must-not*) se coloca en el modelo actual del concepto.

Se pueden resumir los elementos que componen este sistema de aprendizaje como:

- Un lenguaje de representación, casi siempre un sistema de frames[2].
- Un mecanismo dirigido de cotejamiento de estructuras.
- Un proceso de generalización.
- Un proceso de especialización.

[2]Una red semántica cuyos nodos son frames.

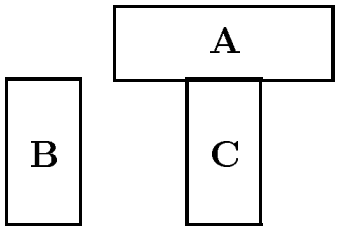


Figura 2.8: Cuasiejemplo de arco

Así los cuasiejemplos restringen la extensión del concepto, al hacer más rígidas las condiciones para que un objeto sea considerado como ejemplo de un concepto.

El algoritmo de *especialización* contiene los siguientes pasos:

1. Cotejar el modelo actual del concepto con el cuasiejemplo suministrado. Establecer las correspondencias existentes entre las partes, y localizar aquellas que no son iguales.
2. Determinar si existe una diferencia entre ambos:

   2.1 Si existe una única diferencia, determinar si el modelo actual del concepto o el cuasiejemplo posee una etiqueta que el otro no tiene.

   2.1.1 Si el modelo actual del concepto tiene una etiqueta que no existe en el cuasiejemplo, usar la heurística *require-link*. La interpretación de esta acción es: *Exigir necesariamente la existencia de esta etiqueta para identificar cualquier otra instancia positiva de este concepto.* Por ejemplo, si se adopta la descripción de la instancia de la figura 2.5 como modelo inicial de arco, al mostrarle al algoritmo el cuasiejemplo que se puede ver en la figura 2.8 se da cuenta de la necesidad de que el objeto A esté soportado por el objeto B. Por lo tanto, cambiaría una relación *soporta* por una relación *debe soportar*.

   2.1.2 Si el cuasiejemplo tiene una etiqueta que no está en el modelo actual del concepto, usar la heurística *forbid-link*. La interpretación de esta acción es: *Prohibir expresamente la existencia de esta etiqueta en los miembros de esta clase.* Siguiendo con el ejemplo de arco, si ahora el algoritmo encuentra el cuasiejemplo de la figura 2.9, entonces se daría cuenta de que la relación *tocar* que cumplen los objetos B y C se ha de prohibir en el modelo del concepto, luego en él aparecería una relación *no debe tocar*.

   2.2 Si no existe una única diferencia, ignorar el ejemplo negativo, ya que no es un cuasiejemplo. La interpretación de esta acción es: *Si existe más de una diferencia importante es difícil decidir sobre cuál actuar o en qué orden.* Por ejemplo, si cuando se tiene la descripción de arco de la figura 2.3 se muestra al algoritmo la instancia negativa de la figura 2.7, entonces no sabe qué aspecto concreto es el que hace que esa combinación de bloques no sea un arco (porque falta el soporte izquierdo, o porque falta el soporte derecho, o porque faltan ambos soportes ...), por lo que la única decisión que puede tomar sin riesgo de equivocarse es ignorar el ejemplo negativo presentado. Una posible alternativa consistiría en ir construyendo un árbol con las diferentes posibilidades. Al ir viendo posteriormente más ejemplos positivos

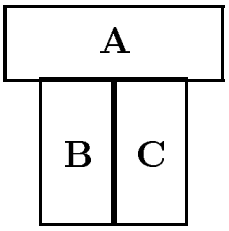


Figura 2.9: Otro cuasiejemplo de arco

y negativos, se deberían eliminar aquéllas que llevaran a alguna contradicción con el modelo del concepto.

El paso 2.2 introduce una nueva cuestión: el *orden* de presentación de los ejemplos. Del orden, en este tipo de aprendizaje, depende de manera directa la *calidad* de lo aprendido; si la secuencia no es la apropiada no se aprende el concepto. Así, en este tipo de aprendizaje, la participación del profesor es determinante.

El proceso de *generalización* permite extender el rango de objetos (instancias) que son cubiertos por el concepto que se está aprendiendo. Winston propone dos estrategias para hacer operativa esta acción. Estas estrategias son *climb-tree* (subir el árbol de generalización) y *enlarge-set* (aumentar el conjunto de posibles valores de un atributo). En el algoritmo de generalización que se presenta a continuación se puede ver en qué casos se aplican y cuál es su efecto:

1. Cotejar el modelo actual del concepto y un ejemplo positivo.
2. Para cada diferencia determinar su tipo.

   2.1 Si la diferencia consiste en que la etiqueta – perteneciente al modelo actual del concepto – apunta a una clase distinta. Si la clase pertenece a algún árbol de generalización (a alguna jerarquía), el modelo actual del concepto evoluciona hacia una generalización. Hay dos casos:

   2.1.1 Si las clases son parte del mismo árbol de generalización usar *climb-tree*. Esta estrategia tiene sentido cuando ambas representaciones son *estructuralmente* equivalentes, ya que en ese caso las diferencias hay que buscarlas en los valores de alguna propiedad. Si partiendo de la descripción inicial de arco (figura 2.3) se introduce el ejemplo positivo de la figura 2.6, entonces el algoritmo generaliza la forma del dintel al primer antecesor de *rectángulo* y *triángulo* en el árbol de generalización del atributo *forma* (ver figura 2.2), que es *polígono*.

   2.1.2 Si la nueva etiqueta no está en el árbol de generalización, eliminar la etiqueta. Esta estrategia se aplica cuando falla la estrategia de *climb-tree*. El sentido de esta eliminación (de una etiqueta) es que con la información actual (el conocimiento de respaldo actual) no es posible construir una abstracción y, entonces, quizás no es importante. Por ejemplo, si después de ver el arco de la figura 2.3 se muestra al algoritmo un ejemplo positivo de arco donde el dintel tiene forma de *pentágono* – como este valor no aparece en el árbol de generalización del atributo *forma* – el algoritmo considera que ese atributo no es relevante y se puede eliminar.

2.1.3 Aplicar *enlarge-set*. Esta estrategia funciona de la siguiente manera: supongamos que en la descripción inicial de arco, el dintel tiene color azul. Si se encuentra otra instancia positiva de arco donde el dintel es blanco, y no existe un árbol de generalización para el atributo *color*, entonces el algoritmo de Winston modificaría el modelo actual de arco para que el dintel pudiera ser azul o blanco. De esta forma se está generalizando el concepto porque se está ampliando el rango de posibles situaciones en las que se acepta una instancia como arco.

2.2 Si la diferencia es que la etiqueta falta en el modelo actual del concepto o en el ejemplo, entonces eliminar la etiqueta. Por ejemplo, si en el modelo de arco no se encuentra el atributo *color*, entonces no se tiene este atributo en cuenta aunque aparezca en ejemplos positivos del concepto vistos posteriormente.

2.3 Si la diferencia aparece en los valores asociados a una propiedad (o *slot*), entonces hay que restringir el rango (*close-interval*). Por ejemplo, si en un cierto momento en el modelo de arco está definido el atributo *color* pero no tiene ningún valor asociado, y se encuentra un ejemplo positivo donde este atributo tiene un valor determinado, entonces el algoritmo incluiría este valor concreto en el modelo del concepto.

2.4 En cualquier otro caso, ignorar la diferencia.

Este algoritmo permite estudiar algunas características importantes del proceso de cotejamiento entre dos ejemplos positivos. Como se asume que ambos ejemplos pertenecen a la misma clase la búsqueda de diferencias se dirige a encontrar formas que permitan identificarlos como miembros de la misma clase. Esto implica *flexibilizar* la definición actual del concepto para incluir la nueva instancia, representada por el ejemplo. Dicho de otra manera, primero se estudia la estructura, y luego los atributos que califican a los elementos de la estructura.

La estrategia de *enlarge-set* está concebida para permitir que el valor asociado a un atributo pueda pertenecer a un conjunto enumerado. La estrategia de *climb-tree* asume la existencia de una jerarquía que permite clasificar los objetos de un dominio como miembros de una clase. Dicha estructura, si existe *a priori* ha sido provista por el profesor, sino ha de construirse. Cuando es necesario realizar una *generalización* resulta indispensable tener conocimientos sobre el dominio. A esta información se le llama conocimiento de respaldo.

En el aprendizaje a partir de las descripciones estructurales de una secuencia de objetos es importante poder distinguir lo *importante* de lo *incidental*. Es decir, entre dos ejemplos pueden existir varias diferencias pero si tenemos el *conocimiento de respaldo* adecuado es posible decidir cuál es la más relevante y, en su caso, olvidar el resto. En general, en el método propuesto por Winston esta tarea la realiza el *profesor*, mediante la elección de una buena secuencia de ejemplos y cuasiejemplos, y de esta manera *sesga* el proceso.

Este sistema nos muestra algunas facetas importantes del proceso de aprendizaje que pueden ser resumidas con varios principios básicos:

- Si existe cualquier duda sobre lo que hay que aprender, es mejor no aprender (2.1.2) y (2.4).
- Cuando algo es una instancia positiva de un concepto y éste no se le parece, entonces crear una excepción (*principio de no alteración*). Los sistemas vivos y, por extensión,

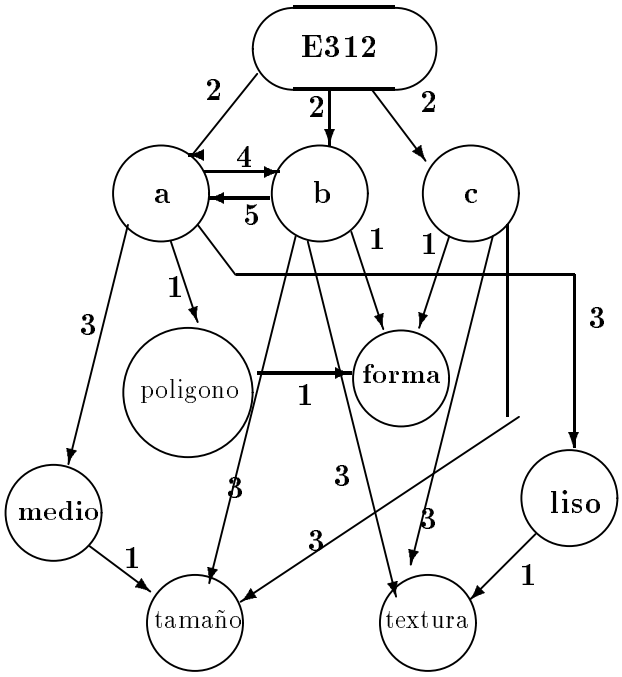


Figura 2.10: Generalización *a la Winston*

los artificiales, son *conservadores*, así que son reacios a cambiar totalmente un concepto ya aprendido.

- El aprendizaje se realiza en pasos muy pequeños, secuencialmente, refinando las ideas. Esta es la llamada ley de Martin[3].

- El algoritmo funciona como una búsqueda del tipo primero-en-profundidad (*depth-first*). Como se ha visto esta estrategia tiene el inconveniente de ser muy sensible al orden de entrada de los ejemplos.

Para comparar el método de Winston con otros métodos de adquisición de conceptos se utilizará el ejemplo de la figura 2.3, tomado de **[DIET81]**. En ese artículo se contrastan los métodos de adquisición de conceptos de Buchanan (**[BUCH78]**), Hayes-Roth (**[HAYE77]**),

[3] *You can't learn anything unless you almost know it already.* No puedes aprender algo a menos que casi lo sepas antes de que te lo enseñen.

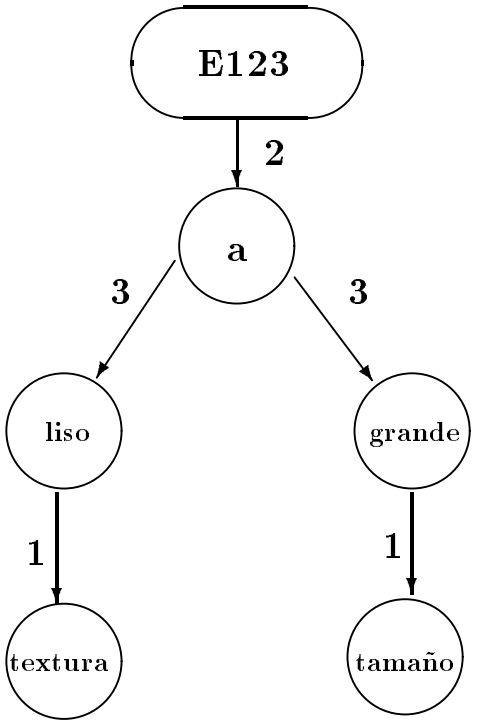


Figura 2.11: Otra generalización *a la Winston*

Vere ([**VERE75**]), Winston ([**WINS75**]) y Michalski ([**DIET81**]). Se empezará la comparación viendo los resultados que obtiene Winston en el ejemplo de la figura 2.3, donde se presentan tres instancias positivas de un concepto, y ningún contraejemplo.

En el método de Winston la generalización a que se llega depende del orden en que se le presentan los ejemplos. En las figuras 2.10 y 2.11 se pueden ver dos de las generalizaciones a las que llega el programa de Winston a partir de las tres instancias positivas mostradas en la figura 2.3.

La primera generalización se puede parafrasear de la siguiente forma: *Hay un polígono de tamaño medio, que no está sombreado, sobre otro objeto que tiene tamaño y textura. Hay también otro objeto con tamaño y textura.* La segunda generalización es equivalente a decir: *Hay un objeto grande que no está sombreado.*

Un aspecto interesante de la primera generalización encontrada con el método de Winston es que dice que *hay un* **polígono** *de tamaño medio.* La palabra *polígono* aparece porque una de las reglas de generalización empleadas por Winston es la regla de subida por el árbol

de generalización (*climbing generalization tree rule*, [**MICH84a**]). En este caso específico la jerarquía de generalización se refiere a los valores que puede tener el atributo *forma*, y sólo tendría un nivel, como se puede ver en la figura 2.2.

Así, si en el nuevo ejemplo y en el modelo actual del concepto hay dos nodos que realizan la misma función (representan al mismo objeto dentro del concepto, se podría decir) y tienen valores diferentes en un atributo, estos valores específicos serán substituídos en la generalización por el valor más bajo en el árbol de generalización que sea antecesor de ambos valores. En este caso, dados dos valores diferentes cualesquiera en el atributo *forma* siempre serían substituídos por el único antecesor que tienen, que es *polígono*.

2.2.2 Método Hayes-Roth

En el trabajo de Hayes-Roth y McDermott ([**HAYE77**], [**HAYE78**]) sobre aprendizaje inductivo se intenta encontrar las generalizaciones conjuntivas más específicas (en su nomenclatura **maximal abstractions** o **interference matches**) a partir de un conjunto de instancias positivas. La estructura que usan para representar tanto estas instancias como las generalizaciones la llaman **parameterized structural representations** (PSRs). Las instancias del concepto objetivo que se usan en esta sección se describirían de la siguiente forma:

```
E1: {{caja:a}{circulo:b}{rombo:c}
      {liso:a}{sombreado:b}{liso:c}
      {grande:a}{medio:b}{medio:c}
      {sobre:b, debajo:a}{sobre:c, debajo:b}}

E2: {{rectangulo:d}{circulo:e}{circulo:f}{cuadrado:g}
      {grande:d}{pequeno:e}{pequeno:f}{medio:g}
      {liso:d}{sombreado:e}{sombreado:f}{liso:g}
      {sobre:g, debajo:d}{fuera:d, dentro:e}{fuera:d, dentro:f}}

E3: {{elipse:h}{rectangulo:i}{triangulo:j}
      {liso:h}{sombreado:i}{liso:j}
      {grande:h}{medio:i}{medio:j}
      {sobre:i, debajo:h}{sobre:j, debajo:i}}
```

Cada uno de los componentes de esta representación es un **case frame**, compuesto de **case labels** (*pequeño, círculo*) y de parámetros (*a*, *b*). Se asume que todos los *case frames* están conectados de forma conjuntiva. La generalización se hace de la siguiente manera: el primer conjunto de generalizaciones conjuntivas, G_1, se inicializa con el primer ejemplo de la entrada. Dado un nuevo ejemplo y el conjunto de generalizaciones obtenido en el paso i-ésimo G_i, G_{i+1} se obtiene haciendo un cotejamiento parcial (**interference match**) entre cada elemento de G_i y el ejemplo de entrenamiento actual. Este cotejamiento intenta encontrar la asociación uno-a-uno más larga de parámetros y *case frames*. Esto se hace en 2 pasos:

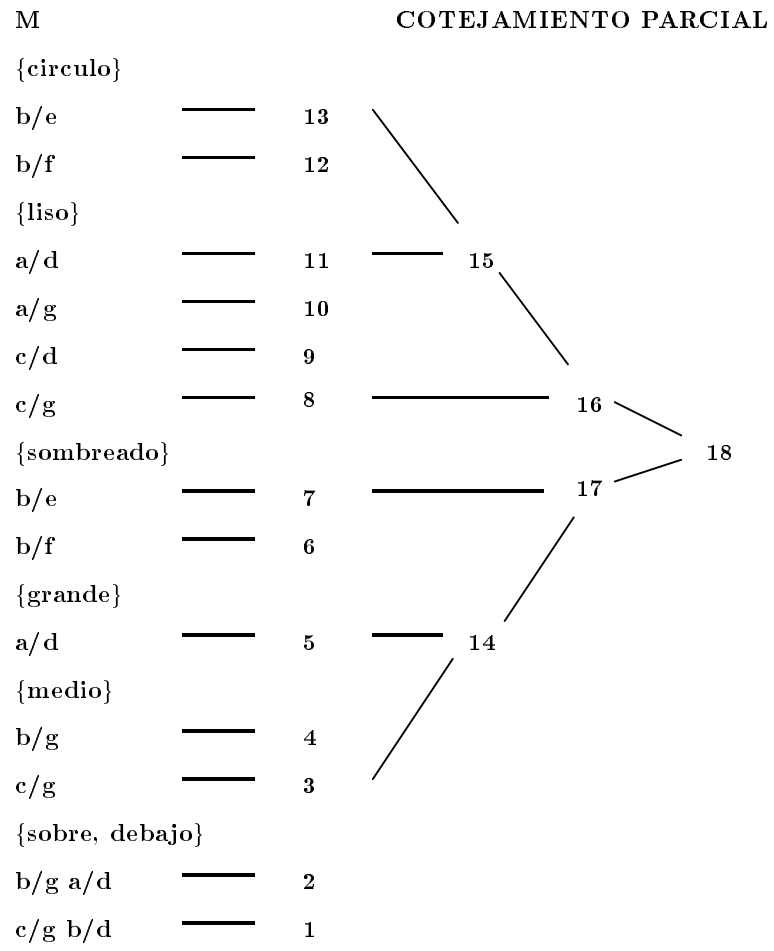


Figura 2.12: Parte del grafo de posibles vinculaciones de Hayes-Roth

1. Hacer el cotejamiento de los *case frames E1* y *E2* de todas las formas posibles para obtener un conjunto M, cada elemento del cual será un *case frame* y una lista de correspondencias entre parámetros que permite hacer *matching* con ambos *case frames*.

   El conjunto M que se obtendría a partir de los 2 primeros ejemplos descritos anteriormente sería el siguiente:

```
M = {{circulo:((b/e)(b/f))},
     {liso:((a/d)(a/g)(c/d)(c/g))},
     {sombreado:((b/e)(b/f))}
     {grande:((a/d))}
     {medio: ((b/g)(c/g))}
     {sobre,debajo: ((b/g a/d)(c/g b/d))}}
```

2. Seleccionar un subconjunto de las correspondencias entre parámetros de M de tal forma que todos los parámetros se puedan vincular de forma consistente. Esta selección se hace con un recorrido en anchura del espacio de posibles vinculaciones, podando los nodos que no sean prometedores.

Una vinculación *consistente* significa no vincular un mismo parámetro de una instancia con varios parámetros de otra instancia. Un trozo del grafo que se obtendría a partir de este conjunto M se puede ver en la figura 2.12. Cada número de ese grafo representa uno de los nodos que se generarían en el proceso de generalización. El nodo 18 es una vinculación que lleva a una generalización conjuntiva. Concretamente vincula a a d (para obtener v1), b a e (para obtener v2) y c a g (para obtener v3), produciendo la conjunción:

```
{{circulo:v2}
 {liso:v1}{sombreado:v2}{liso:v3}
 {grande:v1}{medio:v3}}.
```

Esta conjunción se interpretaría como "*Hay un círculo sombreado, un objeto grande no sombreado y un objeto mediano que tampoco está sombreado*".

El algoritmo de Hayes-Roth encuentra las siguientes generalizaciones a partir de los 3 ejemplos considerados en este capítulo :

- {{**sobre:v1, debajo:v2**}{**medio:v1**}{**liso:v1**}}
  - *Hay un objeto no sombreado de tamaño medio encima de otro objeto*
- {{**sobre:v1, debajo:v2**}{**medio:v1**}{**grande:v2**}{**liso:v2**}}
  - *Hay un objeto de tamaño medio encima de un objeto grande que no está sombreado*
- {{**medio:v1**}{**liso:v1**}{**grande:v3**}{**liso:v3**}{**sombreado:v2**}}
  - *Hay un objeto no sombreado de tamaño medio, un objeto grande no sombreado y un objeto sombreado*

2.2.3 Método Vere

En su trabajo sobre aprendizaje inductivo (**[VERE75]**), Vere también intenta encontrar las generalizaciones conjuntivas más específicas (en su terminología **maximal conjunctive generalizations** o **maximal unifying generalizations**) de un conjunto de instancias positivas de un concepto. Cada ejemplo se representa como una conjunción de **literales**, donde cada literal es una lista de constantes (**términos**) entre paréntesis. Por ejemplo, las tres instancias que se usan a lo largo de este capítulo se representarían de la siguiente manera:

```
EJ1: (caja a)(circulo b)(rombo c)
      (grande a)(medio b)(medio c)
      (liso a)(sombreado b)(liso c)
      (sobre b a)(sobre c b)

EJ2: (circulo d)(circulo e)(rectangulo f)(cuadrado g)
      (pequeno d)(pequeno e)(grande f)(medio g)
      (sombreado d)(sombreado e)(liso f)(liso g)
      (sobre g f)(dentro d f)(dentro e f)

EJ3: (elipse h)(rectangulo i)(triangulo j)
      (grande h)(medio i)(pequeno j)
      (liso h)(sombreado i)(liso j)
      (sobre i h)(sobre j i)
```

Aunque se parezca a la manera de representar los ejemplos de Hayes-Roth con *case frames* es bastante diferente, porque Vere trata todos los símbolos de igual manera. No le da semántica alguna a esta representación, no distingue entre nombres de propiedades (p.e. **grande**) y objetos concretos como **a** o **g**. Este hecho llevará a una serie de problemas que se comentarán posteriormente.

El algoritmo que utiliza para generalizar un par de ejemplos es el siguiente:

1. Creación del conjunto *MP*, que contiene todos los pares de literales que hagan **matching**. Dos literales hacen *matching* si tienen el mismo número de constantes y al menos un término común en la misma posición. En el ejemplo, si se consideran las 2 primeras instancias y se sigue este proceso el conjunto resultante sería:

```
MP={((circulo b),(circulo d))
    ((circulo b),(circulo e))
    ((grande a),(grande f))
    ((medio b),(medio g))
    ((medio c),(medio g))
    ((liso a),(liso f))
    ((liso a),(liso g))
    ((sombreado b),(sombreado d))
    ((sombreado b),(sombreado e))
    ((liso c),(liso f))
    ((liso c),(liso g))
    ((sobre b a),(sobre g f))
    ((sobre c b),(sobre g f))}
```

2. Selección de todos los posibles subconjuntos de *MP* de forma que ningún literal de un ejemplo esté emparejado con más de un literal en otro ejemplo. Cada uno de estos

subconjuntos formará una generalización de los ejemplos iniciales al final del proceso. En **[DIET81]** ya se hace notar que este paso puede ser muy costoso, ya que el espacio de posibles subconjuntos de *MP* es muy grande (exponencial con el número de elementos). Con el conjunto *MP* que se acaba de mostrar existen cientos de subconjuntos posibles que cumplen la propiedad pedida.

En este segundo paso se puede llegar a expresiones que no se podían alcanzar en ninguno de los métodos que se han comentado hasta ahora. Por ejemplo, algunos de los subconjuntos posibles a partir del conjunto *MP* visto anteriormente serían:

```
S1 = {((medio b)(medio g))
      ((liso a)(liso g))}

S2 = {((liso a)(liso f))
      ((liso c)(liso f))}
```

Como se puede apreciar, se está *ligando*, de alguna manera, dos objetos de un ejemplo con un sólo objeto del segundo ejemplo (**a** y **b** con **g** en el primer caso y **a** y **c** con **f** en el segundo). Como se verá al final del proceso, este hecho hace que en las generalizaciones obtenidas por Vere haya vinculaciones de variables del tipo **many-to-one**, y no **one-to-one** como se habían descrito hasta ahora. En la opinión de Dietterich y Michalski (y de la mayoría de los científicos que se dedican al aprendizaje inductivo), *normalmente este tipo de generalizaciones no tienen sentido, y su generación incontrolada es computacionalmente costosa.*

3. Cada subconjunto de los obtenidos en el paso 2 se extiende añadiéndole nuevos pares de literales. Un nuevo par p se añade a un subconjunto S de *MP* si cada literal de p está relacionado con algún otro par q de S por una constante común en la misma posición. Por ejemplo, si en un subconjunto S tenemos el par *((cuadrado b),(cuadrado d))*, se podría añadir el par *((sobre a b),(dentro e d))* porque el tercer elemento de *(sobre a b)* es el segundo de *(cuadrado b)* y el tercer elemento de *(dentro e d)* es el segundo elemento de *(cuadrado d)*.

   Si en el segundo paso el espacio de posibles subconjuntos era grande, en este tercer paso todavía lo es más. En **[DIET81]** se comenta que en ninguno de los trabajos publicados por Vere (**[VERE75]**, **[VERE77]**, **[VERE78]**, **[VERE80]**) se describe claramente cómo se efectúan los pasos 2 y 3 de este algoritmo de generalización, pero no debe ser con una búsqueda exhaustiva porque sería muy ineficiente.

4. El conjunto resultante de pares se convierte en una conjunción de literales, uniendo cada par para que forme un literal. Los términos que no hacen matching se transforman en nuevos términos, que pueden ser vistos formalmente como variables. Por ejemplo *((círculo a),(círculo c))* pasaría a ser *(círculo v1)*.

   Esta forma de crear literales y el hecho de no distinguir predicados de constantes hará que se puedan generar cosas extrañas. Antes se ha visto que, en la fase de extensión de los pares de literales, se podían añadir cosas como *((sobre a b),(dentro e d))*. Al hacer ahora la generalización de este par para formar un literal, se obtendría un literal como *(v1 v2 v3)*, que no tiene demasiado sentido tampoco.

En el ejemplo de la figura 2.3 que se está considerando para ir comentado los diversos métodos Vere obtiene muchas generalizaciones, algunas de las cuales son las siguientes:

- **(SOBRE v1 v2) (MEDIO v1) (GRANDE v2) (LISO v2) (LISO v3) (SOMBREADO v4) (v5 v4)**

  Hay un objeto de tamaño medio sobre un objeto grande no sombreado. Otro objeto no está sombreado. Hay un objeto sombreado.

- **(SOBRE v1 v2) (LISO v1) (MEDIO v1) (v9 v1) (v5 v3 v4) (SOMBREADO v3) (v7 v3) (v6 v3) (LISO v4) (GRANDE v4) (v8 v4)**

  Hay un objeto no sombreado de tamaño medio sobre otro objeto. Hay dos objetos relacionados de alguna forma tal que uno es sombreado y el otro es grande y no está sombreado.

- **(SOBRE v1 v2) (MEDIO v1) (LISO v2) (GRANDE v2) (v5 v2) (SOMBREADO v3) (v7 v3) (LISO v4) (v6 v4)**

  Hay un objeto de tamaño medio sobre un objeto grande no sombreado. Hay un objeto sombreado y hay un objeto no sombreado.

Como se puede apreciar a partir de estos resultados, la aparición de literales como *(v5 v4)* o de hasta 7 variables diferentes como en la última generalización lleva a bastante confusión, pero si se eliminan los literales *vacíos* (los que sólo contienen variables), las generalizaciones obtenidas son muy similares a las obtenidas con otros métodos.

2.2.4 Método Michalski-Dietterich

En esta sección se va a comentar el método de determinación de las generalizaciones conjuntivas más específicas descrito por Michalski y Dietterich en **[DIET81]**. Ellos describen los ejemplos de entrada en el lenguaje VL_{21}, que es una extensión de la lógica de predicados de primer orden. Cada ejemplo es una conjunción de **selectores**, que normalmente contienen un descriptor de predicados (con variables como argumentos) y una lista de los valores que el predicado puede tener. Otra forma de los selectores son predicados n-arios entre corchetes, que se interpretan de la forma habitual. Los tres ejemplos de la figura 2.3 se representarían de la siguiente forma:

E1: ∃ v1,v2,v3 [tamaño (v1) = grande] [tamaño (v2) = medio] [tamaño (v3) = medio] [forma (v1) = caja] [forma (v2) = círculo] [forma (v3) = rombo] [textura (v1) = liso] [textura (v2) = sombreado] [textura (v3) = liso] [sobre (v2,v1)] [sobre (v3,v2)] E2: ∃ v4,v5,v6,v7 [tamaño (v4) = pequeño] [tamaño (v5) = pequeño] [tamaño (v6) = grande] [tamaño (v7) = medio] [forma (v4) = círculo] [forma (v5) = círculo] [forma (v6) = rectángulo] [forma (v7) = cuadrado] [textura (v4) = sombreado] [textura (v5) = sombreado] [textura (v6) = liso] [textura (v7) = liso] [dentro (v4,v6)] [dentro (v5,v6)] [sobre (v7,v6)] E3: ∃ v8.v9.v10 [tamaño (v8) = grande] [tamaño (v9) = medio] [tamaño (v10) = pequeño] [forma (v8) = elipse] [forma (v9) = rectángulo] [

forma (v10) = triángulo] [textura (v8) = liso] [textura (v9) = sombreado] [textura (v10) = liso] [sobre (v9,v8)] [sobre (v10,v9)]

En este método se tratan de forma diferente los descriptores unarios (o **descriptores de atributos**) y los no unarios (o **descriptores estructurales**). La idea es primero buscar generalizaciones plausibles en el espacio estructural, y después buscar en el espacio de atributos para llenar los detalles de estas generalizaciones. Básicamente lo hacen así para reducir el espacio de búsqueda de las generalizaciones, al tener en cuenta al principio tan sólo el aspecto estructural de los ejemplos.

La parte de obtención de las generalizaciones en el método que se está examinando funciona de la siguiente forma [**DIET81**]. El algoritmo hace una búsqueda del tipo *beam search* ([**RUBI77**]) en el espacio estructural. Esta búsqueda es una forma de buscar primero el mejor (*best-first search*) en la cual se mantiene un conjunto de las mejores descripciones candidatas que se hayan obtenido hasta el momento.

Primero se eliminan todos los descriptores unarios de los ejemplos, quedándose de esta manera sólo con la parte estructural de los mismos. Se escoge un ejemplo de forma aleatoria y se toma como B_0, el conjunto inicial de generalizaciones. En cada paso, primero se eliminan de B_i las generalizaciones *menos prometedoras*. El criterio para evaluar las generalizaciones lo puede dar el usuario, y el programa también tiene algunos criterios incorporados, como maximizar el número de ejemplos cubiertos por una generalización o maximizar el número de selectores en una generalización, por ejemplo.

Después se comprueba si alguna de las generalizaciones de B_i cubre todos los ejemplos. Si es así, se pasan de B_i al conjunto C, donde se almacenan las generalizaciones conjuntivas candidatas.

Finalmente, B_i se generaliza a B_{i+1} cogiendo cada elemento de B_i y generalizándolo de todas las maneras posibles eliminando un selector. La búsqueda finaliza cuando el conjunto C llega a un tamaño determinado. El conjunto C contiene generalizaciones conjuntivas de los ejemplos de la entrada, algunas de las cuales son el máximo de específicas.

Una vez se ha construído el conjunto de generalizaciones candidatas, cada una de ellas se ha de completar encontrando valores para sus descriptores de atributos. Cada generalización se usa para definir un espacio de atributos en el que se hace una *beam search* similar a la realizada en el espacio estructural.

Entre todas las generalizaciones conjuntivas producidas por la primera fase del algoritmo puede haber algunas que no sean lo más específicas posibles. En [**DIET81**] se afirma que en la mayoría de los casos estas generalizaciones se vuelven el máximo de específicas cuando se llenan los atributos en la segunda fase del algoritmo.

Algunas de las generalizaciones obtenidas por este método usando los ejemplos de la figura 2.3 son las siguientes :

- **$\exists$ v1,v2 [sobre (v1,v2)] [tamaño (v1) = medio] [forma (v1) = polígono] [textura (v1) = liso] [tamaño (v2) = medio $\bigvee$ grande] [forma (v2) = rectángulo $\bigvee$ círculo]**

  Existen 2 objetos en cada ejemplo tal que uno es un polígono de tamaño medio no sombreado que está encima del otro, que es un círculo o un rectángulo de tamaño medio

o grande.

- **$\exists$ v1,v2 [sobre (v1,v2)] [tamaño (v1) = medio] [forma (v1) = círculo $\bigvee$ cuadrado $\bigvee$ rectángulo] [tamaño (v2) = grande] [forma (v2) = caja $\bigvee$ rectángulo $\bigvee$ elipse] [textura (v2) = liso]**

  Existen dos objetos tales que uno de ellos es un círculo, rectángulo o cuadrado de tamaño medio que está sobre el otro, que es una caja, rectángulo o elipse grande y no sombreado.

- **$\exists$ v1,v2 [sobre (v1,v2)] [tamaño (v1) = medio] [forma (v1) = polígono] [tamaño (v2) = medio $\bigvee$ grande] [forma (v2) = rectángulo $\bigvee$ elipse $\bigvee$ círculo]**

  Existen 2 objetos tales que uno de ellos es un polígono de tamaño medio que está sobre el otro, un rectángulo, elipse o círculo de tamaño medio o grande.

- **$\exists$ v1 [tamaño (v1) = pequeño $\bigvee$ medio] [forma (v1) = círculo $\bigvee$ rectángulo] [textura (v1) = sombreado]**

  Existe un objeto, que es un círculo o rectángulo, sombreado y de tamaño medio o pequeño.

Salta a la vista rápidamente que la principal diferencia de las generalizaciones obtenidas con este método respecto a las de otros métodos reside en las descripciones disyuntivas que obtiene (p.e. objetos que son *rectángulos, elipses o círculos*). Puede haber casos en que sean interesantes estos tipos de descripciones, pero en este ejemplo tan sencillo ya se puede ver que produce generalizaciones con interpretaciones un tanto artificiales y difíciles de seguir (p.e. *un rectángulo, elipse o círculo de tamaño medio o grande*).

También se han introducido algunas reglas de inducción constructivas en el sistema, con las que se pueden obtener generalizaciones más informativas, tales como la siguiente :

- **[número de v's = 3,4] [número de v's con textura liso = 2]**

  $\exists$ v1,v2 [cima (v1)] [sobre (v1,v2)] [tamaño (v1) = medio] [forma (v1) = polígono] [textura (v1) = liso] [tamaño (v2) = medio,grande] [forma (v2) = círculo,rectángulo]

  Hay 3 ó 4 objetos en cada ejemplo. De ellos exactamente dos no son sombreados. El objeto en posición más elevada es un polígono liso de tamaño medio, y está sobre un círculo o rectángulo de tamaño grande o medio.

Para cerrar esta sección, se puede ver qué resultados obtiene Michalski en un ejemplo que muestra en **[MICH80b]**. En uno de los ejemplos de este artículo intenta encontrar una descripción de una serie de trenes, formados por una serie de vagones de los cuales interesan las siguientes características:

- **Longitud**: hay vagones cortos y largos.
- **Forma**: forma que tiene el vagón (puede ser una elipse, un rectángulo abierto, un rectángulo cerrado, etc.).

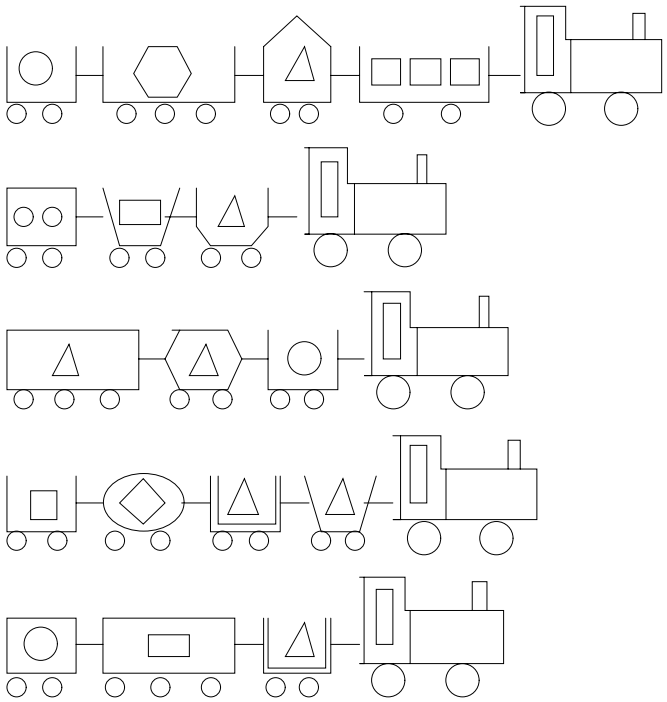


Figura 2.13: Ejemplo de los trenes de Michalski

- **Forma de la carga**: un vagón puede transportar círculos, triángulos, cuadrados, etc.
- **Número de partes**: número de unidades de carga que lleva cada vagón. Puede ser 1, 2 ó 3.
- **Número de ruedas**: cada vagón tiene 2 ó 3 ruedas.

En la figura 2.13 se pueden ver los trenes que usa Michalski en uno de sus ejemplos en **[MICH80b]**.

Las dos descripciones de este tipo de trenes que obtiene Michalski son las siguientes:

- $\exists$ $vagon_1$ [**longitud** ($vagon_1$) **= corto**] [**forma** ($vagon_1$) **= techo cerrado**]

  Hay un vagón que es corto y tiene el techo cerrado.

- $\exists$ $vagon_1$, $vagon_2$, $carga_1$, $carga_2$ [**delante** ($vagon_1$,$vagon_2$)] [**lleva** ($vagon_1$,$carga_1$)] [**lleva** ($vagon_2$,$carga_2$)] [**forma-carga** ($carga_1$) **= triángulo**] [**forma-carga** ($carga_2$) **= polígono**]

  Hay un vagón que lleva un triángulo, y el vagón que va detrás suyo lleva un polígono.

Aspecto	**Winston**	**Hayes-Roth**
Dominio	Mundo bloques	General
Lenguaje	Redes semánticas	PSR
Conceptos sintácticos	Nodos y uniones	Case frames, case labels, parámetros
Operadores	AND, excepción	AND
Reglas de generalización	Eliminar condición Constantes a variables Subir árbol de generalización	Eliminar condición Constantes a variables
Formas disjuntivas	No	No
Inmunidad al ruido	Muy baja	Baja
Conocimiento del dominio	Incorporado al programa	No
Inducción constructiva	Limitada	No

Tabla 2.1: Resumen de los métodos de Winston y Hayes-Roth

2.2.5 Comparación de los métodos

En general todos los métodos examinados dan resultados bastante parecidos. Todos ellos encuentran en algún punto la dificultad de tener que averiguar qué objetos de cada ejemplo están relacionados entre sí. Ese es, obviamente, el punto clave en cualquier algoritmo de adquisición de conceptos (**concept acquisition**), que ha de calcular la descripción de un concepto a partir de las semejanzas entre los ejemplos que se le presentan.

Winston asume que las redes semánticas que tiene que comparar serán muy similares y, por tanto, el algoritmo de cotejamiento no tendrá que enfrentarse con múltiples posibilidades. Esta idea procede del uso de **cuasiejemplos** (*near-misses*), que se diferencian en tan sólo pequeños detalles de los ejemplos positivos del concepto a aprender. Si Winston admitiese contraejemplos cualesquiera el algoritmo de comparación de redes semánticas sería mucho más costoso.

En el primer paso del algoritmo de Hayes-Roth se calculan todas las formas de correspondencia posibles a partir de los **case frames** que definen los ejemplos del concepto (que incluyen tanto relaciones entre objetos como propiedades de los mismos), pudiendo generarse por lo tanto múltiples combinaciones.

En el algoritmo de Vere se hace algo similar, ya que en el primer paso se construye un conjunto con todos los pares de **literales** que compartan un **término** en la misma posición. Aquí ya puede haber muchas posibilidades, pero en el siguiente paso normalmente se incrementa este número, ya que se estudian todos los posibles subconjuntos del conjunto de pares de literales. Los literales también engloban tanto relaciones entre objetos como propiedades.

En el método de Michalski el aspecto más interesante es que se busca la generalización en el espacio estructural, sin tener en cuenta los atributos en una primera fase. Eso hace que se reduzca el número de posibilidades respecto a los métodos anteriores.

Se puede ver en las tablas 2.1 y 2.2 un resumen de la comparación entre los métodos de Winston, Hayes-Roth, Vere y Michalski, teniendo en cuenta los siguientes aspectos:

- **Dominio** de aplicación del método.
- **Lenguaje de representación** utilizado.
- **Conceptos** que maneja el algoritmo.
- **Operadores** permitidos en el lenguaje de representación.
- **Reglas de generalización** conocidas por el algoritmo.
- Posibilidad de inclusión de **información disyuntiva**.
- **Robustez delante de ruido** en los datos de entrada.
- **Conocimiento del dominio** incluído en el programa.
- Posibilidad de realizar **inducción constructiva**.

Aspecto	**Vere**	**Michalski**
Dominio	General	General
Lenguaje	Predicados de primer orden sin cuantificadores	Predicados de primer orden ampliados
Conceptos sintácticos	Literales, constantes	Selectores, variables, descriptores
Operadores	AND	AND, OR, OR interno
Reglas de generalización	Eliminar condición Constantes a variables	Eliminar condición Constantes a variables Subir árbol de generalización Cerrar intervalos Generalización por OR interno
Formas disjuntivas	Sí	Sí
Inmunidad al ruido	Buena	Muy buena
Conocimiento del dominio	Sí	Sí
Inducción constructiva	No	Algunas reglas generales

Tabla 2.2: Resumen de los métodos de Vere y Michalski

2.2.6 Espacio de versiones

Mitchell **[MITC82]** propuso un marco unificado para el aprendizaje de conceptos llamado *espacio de versiones*. Este método supone que el proceso de aprendizaje de un concepto tiene lugar en un espacio H definido entre dos conjuntos de hipótesis llamados G y S (ver figura 2.14). El conjunto G contiene los elementos más generales de H. En S, por el contrario, se acumulan los elementos más específicos.

Una suposición elemental de este método es que dado un conjunto de instancias positivas y negativas es posible construir un *espacio de versiones* de fórmulas *consistentes* entre las cuales se encuentra el concepto que se ha de aprender. Aquí el conjunto de fórmulas consistentes puede ser definido como el conjunto de fórmulas *completas*, es decir, que reconocen todas las

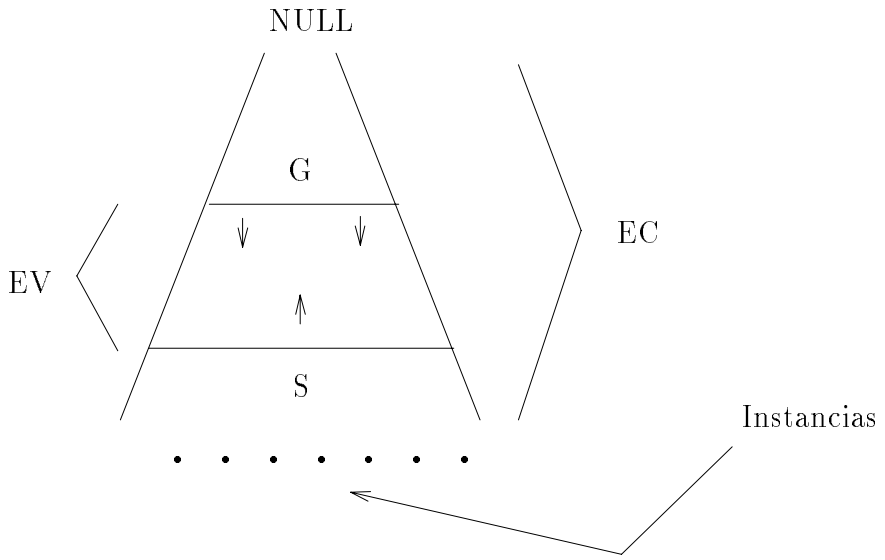


Figura 2.14: El espacio de versiones

instancias positivas, y *coherentes*, que no reconocen las negativas. La construcción de este espacio se lleva a cabo mediante la generalización y especialización del concepto C debida a la presentación de instancias positivas y negativas. Además cualquier descripción consistente con el concepto que puede aprenderse debe hacerse en términos consistentes con H. El resultado de este proceso es que hay un árbol de especializaciones y uno de generalizaciones, donde cada nodo está asociado a un modelo de concepto. Una interpretación de estos nodos es la siguiente:

1. Un nodo del árbol de generalización está conectado a un modelo que cubre todos los ejemplos positivos.
2. Un nodo del árbol de especializaciones est'a conectado a un modelo que no cubre ningún contraejemplo.

El espacio de versiones permite mantener toda la información *útil* extraída de un conjunto de entrenamiento sin tener que guardar ninguno de los ejemplos. Además, este método permite encontrar el estado exacto de la generalización en el cual un descriptor debe ser usado para optimizar la eficiencia de los operadores que le utilizan en la resolución de un problema. Una generalización, g, está contenida en el espacio de versiones definido entre G y S, si y sólo si:

- g es más específica o igual que algún miembro de G, y
- g es más general o igual que algún miembro de S

Se asume que el dominio está descrito por una teoría T [4], que contiene el conocimiento de

[4]Esta teoría es el sesgo semántico asociado al dominio.

respaldo (Δ). El conjunto de los ejemplos positivos se define como $P_i \in \mathcal{P}$. El conjunto de ejemplos negativos se define como $N_i \in \mathcal{N}$. I es el conjunto de descripciones de los ejemplos positivos y negativos del concepto objetivo. Este conjunto está naturalmente particionado así:

$$I = P \cup N$$

F y S son fórmulas lógicas, y $\leq$ es la relación "*más general que*", tal que:

$$F \leq R \iff T, R \vdash F$$

Para definir el *espacio de versiones* de un concepto es necesario tener :

1. Un criterio de consistencia.
2. Uno o más criterios para escoger la fórmula más específica (INF) y la más general (SUP) entre las consistentes.
3. Una definición de generalización.
4. Un conjunto de ejemplos positivos y negativos (*training set*).

Un posible criterio de consistencia es el siguiente: *F es consistente con la teoría T, el conjunto de ejemplos positivos $\mathcal{P}$ y el de los negativos $\mathcal{N}$ si*:

$$\forall i \ (T, P_i \vdash F)$$

$$\forall j \ (T, N_j \vdash \neg F)$$

La interpretación de este criterio es que dados $T, \mathcal{P}, \mathcal{N}$ se consideran *consistentes* todas aquellas fórmulas que puedan ser deducidas de cada uno de los P_i, pero ninguna de aquellas que se puedan deducir de los N_j, para cada P_i.

Un posible criterio de selección sería el siguiente:

- Conjunto de fórmulas más específicas

  Sea R cualquier fórmula y sea $S \in \{\mathcal{S}\}$, el conjunto de fórmulas más específicas. Entonces si S es consistente, para todo R se cumple que:

$$\forall R[\ R \text{ es consistente } \& \ T, R \vdash S] \Longrightarrow [T \vdash [R \iff S]]$$

  Intuitivamente esta fórmula indica que $T, R \vdash S$ significa que R es más específica que S: Como R es más particular que una fórmula de $\{\mathcal{S}\}$, entonces es una fórmula de $\{\mathcal{S}\}$.

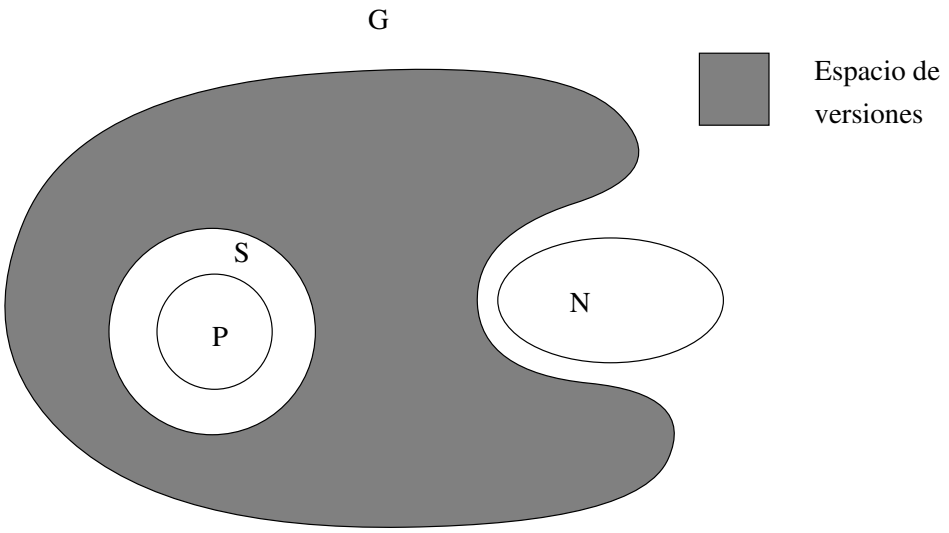


Figura 2.15: Otra visión del espacio de versiones

- Conjunto de fórmulas más generales

  Sea R cualquier fórmula y sea $G \in \{\mathcal{G}\}$, el conjunto de las fórmulas más generales. Si G es consistente, para todo R se cumple que:

$$\forall R \quad [R \, es \, consistente \; \& \; T, G \vdash R] \Longrightarrow [T \vdash [R \Longleftrightarrow G]]$$

El resultado de estos criterios de selección asegura la existencia de dos conjuntos $\mathcal{G}$ y $\mathcal{S}$ tales que:

$\{\mathcal{S}\}$ $s \in \{\mathcal{S}\}$ es una generalización que es consistente[5] con las instancias observadas y no hay ninguna que sea, al mismo tiempo, más específica que s y consistente con dichas instancias.

$\{\mathcal{G}\}$ $g \in \{\mathcal{G}\}$ es una generalización que es consistente con las instancias observadas y no hay otra que, al mismo tiempo, sea más general que g y consistente con dichas instancias.

Mientras s y g no sean iguales la existencia de $\{\mathcal{S}\}$ asegura que existe una descripción conjuntiva del concepto a aprender (*target concept*). Este sistema no permite la existencia de términos negados en la descripción de las instancias. En el caso de que existiesen habría que modificar los criterios de selección.

[5]Según el criterio propuesto anteriormente una fórmula es *consistente* si cubre todas las instancias de $\mathcal{P}$ y rechaza todas las de $\mathcal{N}$. Algunos autores utilizan el término *admisible*

El espacio de versiones de un concepto C es el conjunto de fórmulas consistentes entre INF y SUP. La noción de espacio de versiones depende de *cómo* se hace el *reconocimiento* de un ejemplo, es decir, de cómo se decide si una instancia es más específica, equivalente o más general que una fórmula C. De aquí se desprende que dado el conjunto de entrenamiento I y una teoría T, si se emplean diferentes criterios de consistencia, se pueden obtener (*aprender*) diferentes versiones del concepto C.

La idea que subyace al espacio de versiones es el mantenimiento de un conjunto de descripciones *posibles* del concepto C. Este conjunto está naturalmente acotado por $\mathcal{G}$ y $\mathcal{S}$ (ver figura 2.15), cuyos miembros más significativos son SUP e INF. Además, dependiendo de la secuencia de ejemplos positivos y negativos, este conjunto de descripciones posibles puede evolucionar hacia lo específico o hacia lo general. Por esto se puede considerar el aprendizaje en el espacio de versiones como una búsqueda guiada por los datos (*data-driven*).

Los ejemplos positivos recortan (*prune*) las descripciones generales, del conjunto $\mathcal{G}$, y los negativos hacen lo propio con las del conjunto $\mathcal{S}$. Una aportación de este método es el tratamiento simétrico de los ejemplos positivos y negativos.

El conjunto $\mathcal{S}$ se calcula, de lo específico a lo general, empleando la estrategia de búsqueda primero-en-anchura (*breadth-first*). Y el $\mathcal{G}$ con la misma estrategia pero esta vez de lo general a las generalizaciones más específicas. Así la estrategia seguida en la construcción del espacio de versiones puede ser considerada como una extensión de la estrategia de búsqueda primero-en-anchura hacia una búsqueda bidireccional. En la figura 2.14, EC representa el espacio de conceptos y EV el espacio de versiones del concepto que se está aprendiendo y que está limitado por G y S. En la figura parece como si $\mathcal{S}$ fuese más grande que $\mathcal{G}$, pero la interpretación correcta es que $\mathcal{S}$ es más específico y, por tanto, tiene más constantes instanciadas que $\mathcal{G}$. Al *aumentar* el número de constantes en $\mathcal{G}$, en el proceso de aprendizaje, éste se parece cada vez más a $\mathcal{S}$.

El algoritmo de Mitchell se conoce como el algoritmo de eliminación de candidatos, y se basa en la suposición de que los límites del espacio de representación de las hipótesis plausibles $\mathcal{H}$ son precisamente $SUP \in \mathcal{G}$ e $INF \in \mathcal{S}$. Mitchell define una *hipótesis plausible* del concepto, como cualquier hipótesis que no ha sido desestimada por los datos (instancias). El conjunto $\mathcal{H}$ contiene todas las hipótesis plausibles, es el espacio de versiones, y contiene todas las descripciones del concepto C que son consistentes con las instancias positivas procesadas hasta cierto momento.

A continuación se describe el algoritmo de eliminación de candidatos. Los parámetros de entrada son $\{P\}$, $\{N\}$, $\{VS\}$ y $\{S\} := \emptyset$. En $\{VS\}$ se acumulan las fórmulas válidas y $\{G\}$.

Eliminación de Candidatos

1. Si un nuevo ejemplo se añade a $\{P\}$, entonces $\{S\}$ y $\{G\}$ se actualizan de la siguiente manera:

   - Añadir a $\{S\}$ todos los elementos de $\{VS\}$ que cumplan las siguientes condiciones:
     (a) Son especializaciones de un elemento de $\{G\}$.
     (b) Son consistentes.

(c) Ningún elemento de $\{VS\}$ es más específico.

- Actualizar $\{G\}$
  Eliminar de $\{G\}$ todos los elementos de $\{G\}$ que **no** son consistentes teniendo en cuenta $\{P\}$.

2. Si un nuevo contraejemplo se añade a $\{N\}$, actualizar $\{S\}$ y $\{G\}$

   - Actualiza $\{S\}$
     Eliminar todas aquellas fórmulas que no son consistentes
   - Actualizar $\{G\}$
     Añadir en $\{G\}$ todos los elementos de $\{VS\}$ tales que:
     (a) Son generalizaciones de un elemento de $\{VS\}$.
     (b) Son consistentes.
     (c) Ninguno es más general en $\{VS\}$.

Uno de los efectos más importantes de este algoritmo es que, analizados algunos ejemplos positivos, permite el rápido y certero reconocimiento de los ejemplos negativos.

En el siguiente ejemplo de aplicación del algoritmo de eliminación de candidatos para aprender un concepto se podrá apreciar fácilmente su potencia. Suponga que se estudia el dominio de los `animales exóticos` y se tiene una colección de frames que representan los ejemplos positivos y negativos que serán empleados como conjunto de entrenamiento. Para facilitar la comprensión, el vocabulario de este dominio está restringido a los atributos: `Origen, Clase, Alimentación, Valor` y `Situación` y la extensión de cada uno de estos está definida como:

Vocabulario para el dominio de los Animales exóticos					
Origen	Á(frica)	AM(érica)	AS(ia)	E(uropa)	O(ceanía)
Clase	Mamífero	Aves	Pez	Reptil	
Alimentación	Carnívoro	Hervíboro	Omnívoro	Insectívoro	Piscívoro
Valor	Alto	Normal	Bajo		
Situación	Peligro	Normal	Extinguido	Desconocida	

Si el concepto que se pretende aprender es, por ejemplo, "*animal europeo valioso en peligro*", éste puede ser representado como:

$$\begin{aligned} \textit{Origen} &: \text{E} \\ \textit{Clase} &: x_2 \\ \text{Alimentación} &: x_3 \\ \text{Valor} &: \text{Alto} \\ \text{Situación} &: \text{Peligro} \end{aligned}$$

Ahora bien si se considera el conjunto de entrenamiento mostrado en la figura 2.16 y se aplica el algoritmo de eliminación de candidatos se obtiene la siguiente secuencia de nodos para S y G:

- Si se toma el primer ejemplo, los conjuntos S y G resultantes son:

  - $G = \{x_1,\ x_2,\ x_3,\ x_4,\ x_5\}$
  - $S =$ { A, Mamífero, Carnívoro, Alto, Peligro}

- El segundo ejemplo es negativo, así que el efecto es especializar G de tal forma que ningún ejemplo negativo sea cubierto por su definición. Dado el criterio de especialización escogido, en la práctica, la especialización consiste en un cambio de variables por constantes. Hay que recordar que el conjunto G debe ser especializado solamente con las descripciones disponibles en el espacio de versiones actual. El resultado es el siguiente:

  - $G = \{(x_1,\ M,\ x_3,\ x_4,\ x_5),\ (x_1,\ x_2,\ C,\ x_4,\ x_5),\ (x_1,\ x_2,\ x_3,\ Alto,\ x_5),\ (x_1,\ x_2,\ x_3,\ x_4,\ Peligro)\}$.
  - $S =$ { A, Mamífero, Carnívoro, Alto, Peligro }

    La interpretación de este conjunto G es que los `animales exóticos` que cumplen con la descripción son los de la `Clase Mamíferos`, o aquellos cuya `Alimentación` es de tipo `Carnívoro`, o su `Valor Alto`, o están en `Situación` de `Peligro`.

    El conjunto S no resulta afectado por el ejemplo negativo. Hay que notar que la variable x_1 se mantiene ya que en ambos ejemplos el valor es el mismo.

- Al considerar un nuevo ejemplo positivo el algoritmo realiza una generalización de S cuyo resultado es cambiar constantes por variables. Además, hay que eliminar de G el conjunto de descripciones que sean inconsistentes con el nuevo ejemplo positivo. Así se obtiene:

  - $G = \{(x_1,\ x_2,\ x_3,\ \text{Alto},\ x_5),\ (x_1,\ x_2,\ x_3,\ x_4,\ \text{Peligro})\ \}$
  - $S = \{\ (\text{A},\ x_2,\ x_3,\ \text{Alto},\ \text{Peligro})\ \}$

- En este momento el espacio de candidatos, formado por S y G, puede ser descrito como el conjunto de *los animales exóticos africanos de alto valor y en peligro*, (que se desprende de S) o *los animales exóticos de alto valor* o *los animales exóticos en peligro* (que es la lectura que se desprende de G). Con el siguiente ejemplo, que se podría identificar como negativo fácilmente ya que su `Origen` es `Europa` y no `Africa`, hay que especializar G para evitar que incluya *los animales exóticos europeos*. El conjunto S no se modifica. El resultado es:

  - $G = \{(\text{A},\ x_2,\ x_3,\ \text{Alto},\ x_5),\ (\text{A},\ x_2,\ x_3,\ x_4,\ \text{Peligro})\ \}$

Ejemplos positivos y negativos					
Origen	África	África	África	Europa	África
Clase	Mamífero	Reptil	Reptil	Mamífero	Mamífero
Alimentación	Carnívoro	Herbívoro	Herbívoro	Herbívoro	Carnívoro
Valor	Alto	Bajo	Alto	Bajo	Normal
Situación	Peligro	Normal	Peligro	Peligro	Peligro
Ejemplo	+	−	+	−	+

Figura 2.16: Conjunto de entrenamiento

- Los candidatos que cumplen con estas definiciones han de ser forzosamente *animales africanos*. El último ejemplo, que es positivo, afecta a los conjuntos S y G. En el primer caso hay que eliminar aquellas descripciones que no incluyan al ejemplo positivo. En el segundo hay que generalizar la definición. Así se tiene que:

  - $G = S = \{$ (A, x_2, x_3, x_4, Peligro) $\}$

El algoritmo finaliza cuando S y G convergen. Esto significa que no son necesarios más ejemplos para *aprender* el concepto objetivo. Una de las características del algoritmo de eliminación de candidatos es que es muy *conservador*: a cada paso, la poda del espacio de versiones es la más pequeña. Es decir, que si cambiamos el orden de los ejemplos, incluyendo los positivos primero, el sistema siempre dejará una puerta abierta a la posibilidad de incluir `animales exóticos` de otros orígenes hasta la aparición de ejemplos negativos (contraejemplos)[6]. Este conservadurismo, cómo no, tiene asociados algunos inconvenientes, tales como el hecho de que el algoritmo difícilmente converge ante conjuntos de entrenamiento esparsos. Es una tarea del *profesor* construir el conjunto de entrenamiento de manera que quede correctamente especificado. Otras características del algoritmo son:

- Se basa en una búsqueda del tipo primero-en-anchura (*breadth-first*) en el espacio de versiones.
- El conjunto S sólo contiene un elemento.
- La introducción de ruido puede ocasionar que el concepto objetivo sea podado del espacio de versiones[7].

El algoritmo del espacio de versiones aquí descrito está *sesgado* para aprender descripciones conjuntivas de conceptos. Así, en el ejemplo anterior es imposible aprender el concepto `animales exóticos europeos o africanos en peligro o extinguidos`. Este problema puede resolverse con otras versiones del algoritmo que permiten descripciones disyuntivas del concepto buscado.

2.3 Inducción de árboles de decisión

2.3.1 Árboles de decisión

Un árbol de decisión es una representación posible de los procesos de decisión involucrados en tareas inductivas de clasificación. Los atributos son utilizados para crear particiones de conjuntos de ejemplos; los nodos del árbol corresponden a los nombres o identificadores de los atributos, mientras que las ramas de un nodo representan los posibles valores del atributo asociado al nodo. Las hojas son conjuntos ya clasificados de ejemplos.

La estrategia de construcción del árbol consiste en seleccionar –en cada momento– aquel atributo *potencialmente más útil* para la clasificación, entendiendo como tal aquel que prometa generar el mejor árbol a partir de este momento. Dos son los criterios de evaluación de árboles de decisión:

[6] Esta es una aplicación evidente de la ley de Martin.
[7] Un ejemplo típico de ruido es la mala asignación de una etiqueta a un ejemplo.

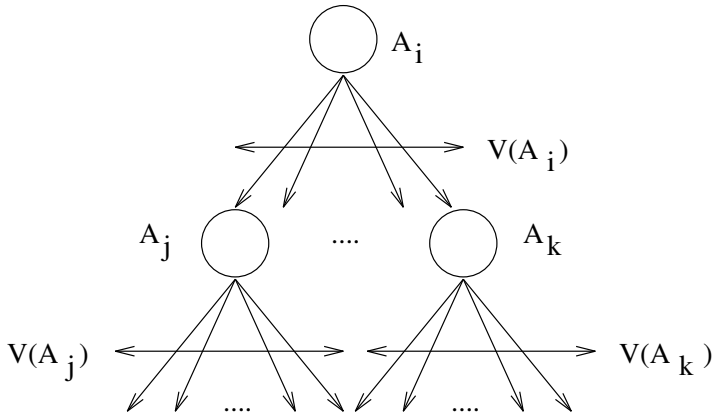


Figura 2.17: Aspecto de un árbol de decisión.

Coste: cuánto cuesta –en media– llegar de la raíz a una hoja. Depende de dos factores: longitud del camino (es decir, número de atributos consultados) y coste de cada consulta.

Bondad: capacidad de predicción del árbol para discriminar un conjunto independiente de ejemplos nuevos (se suele medir en porcentaje de acierto por clase)[8].

Otro aspecto que se suele tener en cuenta es la *legibilidad* del árbol, aunque este criterio se aplica tan sólo en caso de "empate" entre los dos anteriores.

Se puede definir formalmente un árbol de decisión como:

1. Un nodo hoja (o *respuesta*) etiquetado con un nombre de clase.

2. Un nodo interno (o de *decisión*), etiquetado con un nombre de atributo, del que –por cada valor del atributo– parte una rama –conectada a otro árbol de decisión– etiquetada con dicho valor.

Así, las ramas representan las preguntas o decisiones sobre los valores del atributo del nodo padre. De esta manera, un árbol de decisión puede ser visto como un árbol *y/o*, donde la profundidad marca la conjunción y la anchura la disyunción (figura 2.17).

2.3.2 El algoritmo ID3 básico

Este algoritmo fue desarrollado inicialmente por Quinlan [QUIN79], y pertenece a la familia TDIDT[9]. Cada elemento o instancia de la secuencia de entrada presentada al algoritmo toma la forma de una lista de pares atributo-valor, constituyendo una descripción conjuntiva. Cada instancia va asimismo acompañada de la clase a la que pertenece. El objetivo es construir un

[8] Distíngase entre *clasificar* (crear una partición de un conjunto) y *discriminar* (encontrar la clase de un elemento).

[9] *Top-down induction of decision trees.*

árbol de decisión que explique todas las instancias de la manera más compacta posible, según los criterios reseñados en el apartado anterior.

El algoritmo construye el árbol seleccionando en cada momento el mejor atributo según una cierta medida heurística, con lo que puede ser visto como una búsqueda *hill-climbing* sin vuelta atrás a través de todos los posibles árboles. Sea $\mathcal{X}$ el conjunto inicial de instancias, $\mathcal{A}$ el conjunto de atributos que las describen y $\mathcal{C}$ el de posibles clases, y denotemos por el operador # la cardinalidad de un conjunto. A lo largo del texto, y situados en un nodo cualquiera, denominaremos X al conjunto de instancias –subconjunto de $\mathcal{X}$– tal que sus valores coincidan con los del camino que va de la raíz a dicho nodo para los atributos involucrados. A ese camino le llamaremos *camino de discriminación*. Para el nodo raíz, se tiene $X = \mathcal{X}$. Dados un atributo $A \in \mathcal{A}$, un elemento $x \in \mathcal{X}$ y un valor v cualesquiera, definimos:

$$V(A) = \{\text{valores posibles de } A\}$$

$$A(x) = \text{valor de } x \text{ para } A$$

$$A^{-1}(X, v) = \{x \in X | A(x) = v\}$$

$$P_{\mathcal{C}}(X) = \text{partición de } X \text{ en las clases de } \mathcal{C}$$

$$Part(X, A) = \text{partición de } X \text{ con arreglo a } V(A)$$

El esquema básico de construcción del árbol es el siguiente:

{X: conjunto de instancias en curso $\wedge$ A conjunto de atributos que restan por usar}
función Id3 (X, A: conjunto) **devuelve** ad: `árbol-de-decisión`
var árbol1, árbol2: `árbol-de-decisión`;

```
    si (∃C∀x ∈ X : x ∈ C)[10]
        entonces árbol1:= crear-árbol (C)
    sino
        si A ≠ ∅ entonces
          a_M := máx {G(X, a)}_{a∈A};
          árbol1:= crear-árbol (a_M);
          para todo v ∈ V(a_M) hacer
              árbol2:= Id3 (A^{-1}(X, v), A \ {a_M});
              árbol1:= añadir-rama (árbol1, árbol2, v)
          fpara todo
        sino árbol1:= crear-árbol (clase-mayor(X))
        fsi
    fsi
devuelve árbol1
ffunción Id3
```

{ad es un árbol de decisión que describe X usando A}

donde G representa la función de selección, que tiene su valor máximo para aquel atributo considerado por ella como el mejor para continuar la clasificación. La llamada inicial será: Id3 ($\mathcal{X}$, $\mathcal{A}$). El algoritmo descrito utiliza las siguientes funciones auxiliares:

[10] En otras palabras, si todas las instancias de X son de una misma clase C.

clase-mayor(X) : devuelve la clase mayoritaria de los elementos de X.

crear-árbol(Y) : devuelve un árbol de decisión consistente en un solo nodo etiquetado con Y.

añadir-rama(X,Y,Z) : devuelve el árbol resultante de añadir a X una nueva rama etiquetada con Z, y que va a parar al árbol Y. La substituye si ya existía.

Variando la función de selección se obtienen diferentes árboles. La propuesta originalmente por Quinlan está basada en el concepto de entropía de Shannon:

Dado $X \subseteq \mathcal{X}$, se define

$$I(P_{\mathcal{C}}(X)) = -\sum_{C \in P_{\mathcal{C}}(X)} p(X,C)\, log_2 p(X,C)$$

donde

$$p(X,C) = \frac{\#(X \cap C)}{\#X}$$

Así, $I(P_{\mathcal{C}}(X))$ estima la aleatoriedad de la distribución de las instancias de X sobre las clases de $\mathcal{C}$, siendo $p(X,C)$ la probabilidad de que una cierta instancia de X pertenezca a C, definida como la proporción de elementos de X que también pertenecen a C. En otras palabras, $I(P_{\mathcal{C}}(X))$ mide la *cantidad de información* necesaria para obtener la clase, de entre las de $\mathcal{C}$, de un elemento de X.

Asimismo, al particionar un conjunto de elementos X atendiendo a los valores de un determinado atributo A, se puede obtener la información necesaria para discriminar un elemento de X por un árbol con raíz etiquetada con A, denotada por

$$E(X,A) = \sum_{x \in Part(X,A)} \frac{\#x}{\#X} I(P_{\mathcal{C}}(x))$$

donde, como ya se enunció,

$$Part(X,A) = \{A^{-1}(X,v)\}_{v \in V(A)}$$

representa la partición de X en clases mediante los valores de A; la función E, a su vez, estima la aleatoriedad con que las instancias estan distribuidas en las clases, consistiendo en la media ponderada de la cantidad de información requerida por las particiones generadas por los valores del atributo. Finalmente, la ganancia de información viene dada por

$$G(X,A) = I(P_{\mathcal{C}}(X)) - E(X,A)$$

Obsérvese que esta última fórmula equivale a seleccionar el atributo que minimice $E(X,A)$, ya que $I(P_{\mathcal{C}}(X))$ es igual para todos los atributos.

Veamos un ejemplo. En la tabla siguiente se describe un mini-dominio compuesto por los datos de 8 personas, correspondientes a su altura, color del cabello y color de los ojos, distribuidas en dos clases, C^+ y C^-, y se busca el mejor árbol de decisión que lo caracteriza.

Clase	Elemento	Altura	Cabello	Ojos
C^+	1	bajo	rubio	azules
	2	alto	pelirrojo	azules
	3	alto	rubio	azules
C^-	4	alto	rubio	marrones
	5	bajo	castaño	azules
	6	alto	castaño	azules
	7	alto	castaño	marrones
	8	bajo	rubio	marrones

Así pues, $\mathcal{C} = \{C^+, C^-\}$, $\mathcal{X} = \{1,2,3,4,5,7,8\}$ y su partición en las dos clases existentes sería $P_{\mathcal{C}}(\mathcal{X}) = \{\{1,2,3\},\{4,5,7,8\}\}$. Por consiguiente,

$$I(P_{\mathcal{C}}(\mathcal{X})) = -3/8 log_2\ 3/8 - 5/8 log_2\ 5/8 = 0.954$$

Analicemos ahora los atributos:

$$E(\mathcal{X}, \texttt{Altura}) = 3/8 I(P_{\mathcal{C}}(\{1,5,8\}) + 5/8 I(P_{\mathcal{C}}(\{2,3,4,6,7\}) = 0.951$$

Con

$$I(P_{\mathcal{C}}(\{1,5,8\})) = -1/3 log_2\ 1/3 - 2/3\ log_2 2/3 = 0.918$$

$$I(P_{\mathcal{C}}(\{2,3,4,6,7\})) = -2/5 log_2\ 2/5 - 3/5 log_2\ 3/5 = 0.971$$

Finalmente, la ganancia generada por `Altura` será:

$$G(\mathcal{X}, \texttt{Altura}) = 0.954 - 0.951 = 0.003$$

Similarmente,

$$E(\mathcal{X}, \texttt{Cabello}) = 0.454$$

$$E(\mathcal{X}, \texttt{Ojos}) = 0.347$$

Por tanto, se elegirá como atributo raíz `Cabello`. El proceso continuaría ahora para generar los 3 subárboles correspondientes a los 3 valores de `Cabello`, utilizando para ello los conjuntos de instancias $A^{-1}(\mathcal{X}, \texttt{castaño})$, $A^{-1}(\mathcal{X}, \texttt{pelirrojo})$ y $A^{-1}(\mathcal{X}, \texttt{rubio})$, respectivamente. El proceso completo se puede observar en las figuras 2.18 y 2.19.

2.3.3 El algoritmo ID3 normalizado

El método anterior tiene el inconveniente de favorecer indirectamente aquellos atributos con muchos valores, que no son necesariamente los más útiles[11].

Se ha propuesto (en [CEST86]) la *binarización*[12] de los atributos. De este modo se obtienen árboles de decisión binarios y se independiza el proceso del número de valores de un atributo

[11] El atributo `altura` referido a una persona puede tomar muchos valores diferentes pero sería inútil para determinar, pongamos, el tipo de ocupación que desempeña.

[12] Por ejemplo, si el atributo `color del cabello` toma como valores *pelirrojo, rubio, castaño, moreno*, se crearían 4 atributos binarios (únicos valores posibles *sí* y *no*), denominados `cabello pelirrojo`, `cabello rubio`, `cabello castaño` y `cabello moreno`.

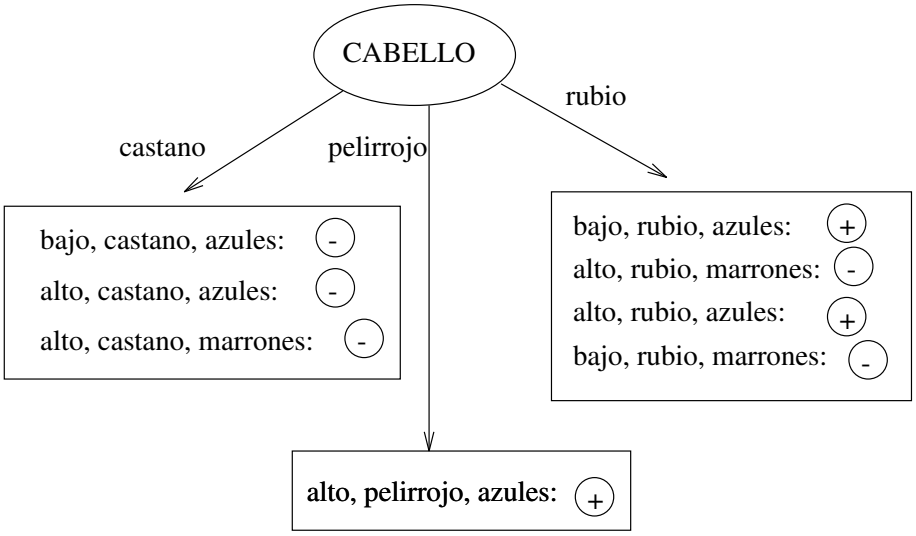


Figura 2.18: Paso según los cálculos del texto.

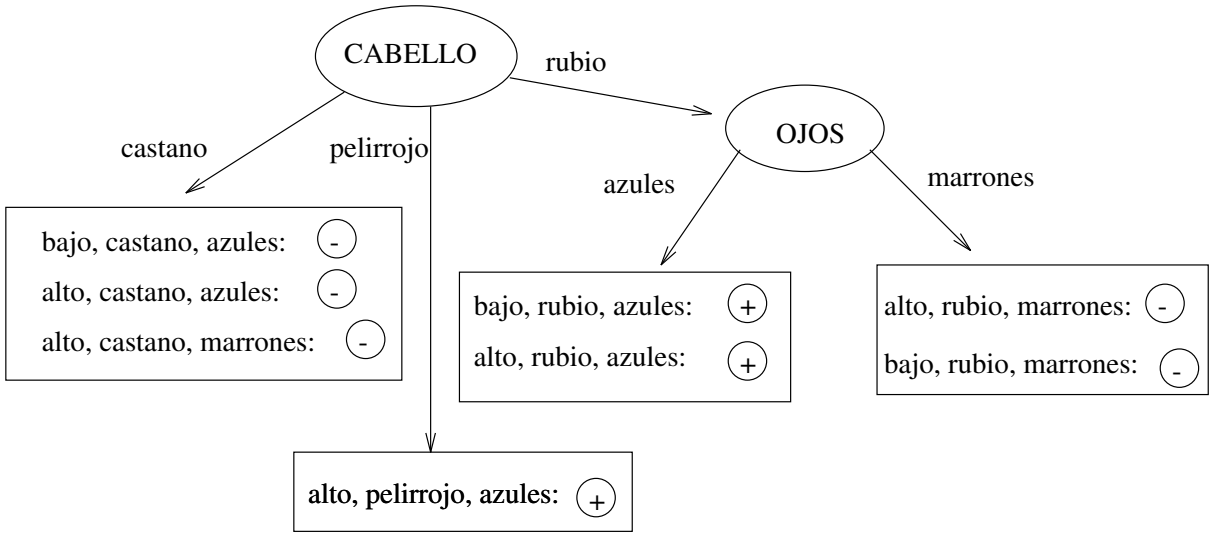


Figura 2.19: Árbol de decisión final generado por ID3.

(que es siempre dos). Lo malo es que los árboles resultantes son poco eficientes, pues preguntan varias veces por un mismo atributo y, además, son menos legibles.

Una alternativa la propuso el propio Quinlan [QUIN86], y consiste en normalizar la ganancia G de su método por un factor que representa la cantidad de información –para un elemento cualquiera– necesaria para conocer el valor de un cierto atributo. Se define

$$IV(X,A) = -\sum_{v \in V(A)} p(X,v)\ log_2\ p(X,v)$$

donde $p(X,v)$ = probabilidad de que, dado un $x \in X$, $A(x) = v$. La nueva ganancia G_N se define como

$$G_N(X,A) = \frac{G(X,A)}{IV(X,A)}.$$

La desventaja de este nuevo método recae en aquellos casos en los que el denominador es anormalmente bajo con lo que –aun sin tener gran ganancia G– el cociente se hace muy grande[13]. Una solución es aplicar G_N sólo a los atributos con una G por encima de la media.

2.3.4 El algoritmo RLM

Un acercamiento diferente lo constituye el propuesto por R. López de Mántaras [LOPE91], al que denominaremos algoritmo RLM. Consiste en escoger aquel atributo que provoque una partición de X más próxima a la correcta, entendiendo por *partición correcta* aquélla en la cual todos los elementos de cada subconjunto de la partición son de la misma clase de $\mathcal{C}$, es decir, la partición $P_{\mathcal{C}}(X)$. Es necesaria, por consiguiente, una definición de distancia entre particiones. El siguiente proceso nos conducirá a ella:

Sean $P_A(X)$ y $P_B(X)$ dos particiones de X. Su información media –que medirá la aleatoriedad de la distribución de los elementos de X por entre las clases de $P_A(X)$ y $P_B(X)$– vendrá dada por:

$$I(P_A(X)) = -\sum_{a \in P_A(X)} p(X,a)\ log_2\ p(X,a)$$

$$I(P_B(X)) = -\sum_{b \in P_B(X)} p(X,b)\ log_2\ p(X,b)$$

Considerando ahora la intersección de ambas particiones obtenemos la expresión:

$$I(P_A(X) \cap P_B(X)) = -\sum_{a \in P_A(X)} \sum_{b \in P_B(X)} p(X, a \cap b)\ log_2\ p(X, a \cap b).$$

Finalmente, la información condicionada de $P_B(X)$ dada $P_A(X)$ es

$$I(P_B(X)/P_A(X)) = I(P_B(X) \cap P_A(X)) - I(P_A(X))$$

[13] Esto ocurre –al ser $IV(X,A)$ una medida de la concentración media de los elementos de X en los valores de A– cuanto más distribuido está el atributo en sus valores.

$$= - \sum_{a \in P_A(X)} \sum_{b \in P_B(X)} p(X, a \cap b)\ log_2\ \frac{p(X, a \cap b)}{p(X, a)}.$$

Es fácilmente demostrable entonces que la medida $d(P_A(X), P_B(X)) = I(P_B(X)/P_A(X)) + I(P_A(X)/P_B(X))$ es una distancia. Si ahora dividimos la expresión anterior por $I(P_A(X) \cap P_B(X))$ conseguiremos su normalización:

$$d_N(P_A(X), P_B(X)) = \frac{d(P_A(X), P_B(X))}{I(P_A(X) \cap P_B(X))} \quad \in [0, 1]$$

que, obviamente, sigue siendo una distancia. Así, este criterio elegirá aquel atributo tal que *minimice* la distancia entre la partición correcta y la generada por el atributo, pudiéndose definir la nueva ganancia –a la que denominaremos G_{RLM}– como:

$$G_{RLM}(X, A) = d_N(P_{\mathcal{C}}(X), Part(X, A)).$$

Es instructivo comparar esta medida con el criterio de Quinlan. La ganancia de información al dividir respecto de un atributo A era $G(X, A) = I(P_{\mathcal{C}}(X)) - E(X, A)$, siendo $I(P_{\mathcal{C}}(X))$ precisamente la información media de la partición correcta de X.

Por otro lado,

$$E(X, A) = \sum_{x \in Part(X, A)} \frac{\#x}{\#X} I(P_{\mathcal{C}}(x))$$

y se demuestra[14] que $E(X, A) = I(P_{\mathcal{C}}(X)/Part(X, A))$.

Por tanto, $G(X, A) = I(P_{\mathcal{C}}(X)) - I(P_{\mathcal{C}}(X)/Part(X, A))$ de donde, sumando y restando $I(Part(X, A)/P_{\mathcal{C}}(X))$, obtenemos:

$$G(X, A) = I(Part(X, A)/P_{\mathcal{C}}(X)) + I(P_{\mathcal{C}}(X))$$

$$-I(P_{\mathcal{C}}(X)/Part(X, A)) - I(Part(X, A)/P_{\mathcal{C}}(X))$$

$$= I(Part(X, A) \cap P_{\mathcal{C}}(X)) - [I(P_{\mathcal{C}}(X)/Part(X, A)) + I(Part(X, A)/P_{\mathcal{C}}(X))]$$

Dividiendo ahora por $I(Part(X, A) \cap P_{\mathcal{C}}(X))$:

$$\frac{G(X, A)}{I(Part(X, A) \cap P_{\mathcal{C}}(X))} = 1 - \frac{I(P_{\mathcal{C}}(X)/Part(X, A)) + I(Part(X, A)/P_{\mathcal{C}}(X))}{I(Part(X, A) \cap P_{\mathcal{C}}(X))}$$

$$= 1 - d_N(P_{\mathcal{C}}(X), Part(X, A))$$

Se demuestra también que $IV(X, A) = I(Part(X, A))$, con lo que $1 - d_N(P_{\mathcal{C}}(X), Part(X, A))$ equivale a normalizar la ganancia de Quinlan $G(X, A)$ por $I(Part(X, A) \cap P_{\mathcal{C}}(X))$ en vez de por $I(Part(X, A))$, que sí es una normalización en $[0, 1]$ y está siempre definida, pudiéndose demostrar además que desaparece el sesgo favorable a los atributos con un número elevado de valores.

[14] Se deja como ejercicio al lector.

2.3.5 Algoritmos incrementales

Una de las limitaciones más evidentes de los métodos expuestos hasta ahora es que todos ellos operan *de una sola vez*, es decir, si dado un árbol ya construido se desea actualizarlo de manera que se acomode a nuevos ejemplos, se debe construir éste de nuevo. Esta manera de trabajar es apropiada para tareas de aprendizaje en las que se tiene un conjunto fijo de instancias, todas ellas conocidas *a priori*. Cuando esto no ocurre, sería de desear que se pudiera revisar el árbol y alterarlo –de la manera más eficiente posible– para dar cabida al nuevo ejemplo. Además, podría ser interesante observar cómo *evoluciona* el árbol a medida que se van proporcionando instancias.

Los algoritmos ID4 e ID4R

Primera tentativa importante de construir un árbol de decisión de manera incremental, este algoritmo fue desarrollado por Schlimmer y Fisher [SCHL86] como una derivación de ID3 aunque –como se verá– existen conceptos aprendibles por ID3 que no lo son por ID4.

Así, este método toma como parámetros de entrada un árbol de decisión y una instancia, y devuelve un nuevo árbol adaptado a ella. Cada nodo mantiene información sobre el número de instancias en las clases para cada valor de cada atributo que pueda servir como decisión en ese nodo –es decir, que no haya aparecido antes en el camino a la raíz– de entre las instancias de X, de cara a recalcular su función E correspondiente. Estos contadores se habrán eventualmente modificado con la introducción de la nueva instancia. De esta manera, si el atributo de la raíz del subárbol que está siendo considerado no es el que mantiene la E más baja, se substituye por el que la tenga, no habiendo necesidad de alterar los contadores asociados al nodo.

Ahora bien, cuando se produce esta substitución, ¿ qué hacemos con los subárboles descendientes del nodo? Lo único claro es que estos árboles ya no son de utilidad. Existen dos estrategias:

- Conformarse con expandir el nuevo atributo en sus valores (es decir, crear sólo un nivel más). Éste es el algoritmo ID4 original.
- Continuar reconstruyendo hacia abajo hasta el final, utilizando ID3. Esta versión se denomina ID4R.

Obsérvese que un subárbol al que le ha sido cambiado (o creado, si antes era una hoja) el atributo raíz no tiene porqué rehacerse por completo. En este caso, el nodo de decisión resultante queda *al aire*, sin refinar. Este refinamiento tendrá lugar eventualmente con la llegada de nuevas instancias que sigan el mismo camino. Esta divergencia entre ID4 y ID4R provoca que el primero no sea equivalente a ID3 y el segundo sí, donde por equivalencia entendemos que los árboles resultantes sean idénticos.

El problema de estos algoritmos es que ciertos conceptos –es decir, sus conjuntos de instancias– pueden dar lugar a continuos descartes de subárboles, sin llegar a una estabilización final de la representación del concepto, y no sean por ello aprendibles, además de degradar en demasía el árbol. Esta situación se suele dar cuando –en el momento de elegir nuevo atributo en un nodo– el ganador no sea *claramente* el mejor. Si había varios con medidas E similares, es muy

probable que haya un nuevo cambio de atributo pronto, con la llegada de una nueva instancia. De todas maneras, son obviamente una mejora si la alternativa es construir cada vez el árbol partiendo de cero, como hacía ID3.

Los algoritmos ID5 e ID5R

Estos dos algoritmos son debidos a Utgoff [UTGO90], y difieren de los dos anteriores en que, en el momento de expandir un subárbol –debido a un cambio de atributo en su raíz– no descartan los subárboles ya existentes, sino que los *reestructuran* convenientemente, aprovechando así el trabajo realizado cuando se crearon. La ventaja de esta técnica –denominada *pull-up*– es que recalcula la información (los contadores) de cada nodo aprovechando los ya existentes a medida que reestructura cada subárbol. La tarea de *pull-up* es, pues, traer el atributo deseado (el que tenga ahora una medida E más baja) a la raíz del subárbol, manteniendo la consistencia con las instancias ya presentadas. Definamos primero –por claridad– la información que debe contener un árbol de decisión para aplicar estos algoritmos:

1. Si es un nodo hoja (de respuesta), un nombre de clase y el conjunto de instancias que discrimina.
2. Si es un nodo de decisión,
   - Un nombre de atributo, con una rama hacia otro árbol de decisión para cada valor del atributo.
   - Todos los posibles atributos de test (incluyendo el actual) en ese nodo, y contadores del número de instancias pertenecientes a cada clase para cada valor de dichos atributos.

Como se ve, se necesita la misma que para los ID4(R), con el añadido de que las hojas deben guardar las instancias que discriminan. La diferencia entre ID5 y ID5R es que éste último –después de reestructurar un subárbol para traer el atributo deseado a la raíz– continua la reestructuración recursivamente por sus subárboles, cosa que aquel no hace, lo que provoca de nuevo que no sea equivalente a ID3, mientras que ID5R sí lo es.

Veamos un esquema a modo de síntesis[15] de los cuatro algoritmos incrementales vistos hasta ahora. Para una mejor comprensión, piénsese que lo que tienen en común todos los métodos es su objetivo: repasar el camino de discriminación de la nueva instancia forzando que cada nodo siga estando etiquetado con el atributo elegido por la función heurística.

Denotemos:

x	Nueva instancia a aprender
C_x	El camino completo de discriminación de x
N	Un nombre de nodo cualquiera del árbol

[15] No se pretende aquí explicarlos en extremo detalle –pues escapa a la concepción del libro– sino que se ha preferido remarcar sus principios básicos y, sobretodo, sus *diferencias*. Para el lector animoso se dan referencias a todos los métodos descritos.

A_N El atributo asociado al nodo N
A_H El nuevo atributo elegido en N al actualizar sus contadores

Siendo el esquema principal:

para todo nodo $N \in C_x$ (de la raíz a la hoja) **hacer**
  Actualizar-contadores (N);
  si $A_N \neq A_H$ **entonces**
    Expandir N un nivel mediante A_H (`ID4`)
    Aplicar TDIDT[16] a N (`ID4R`)
    Aplicar *pull-up* un nivel para substituir
      A_N por A_H (`ID5`)
    Aplicar *pull-up* recursivamente a todo el
      subárbol cuya raíz es N (`ID5R`)
  fsi
fpara todo

El algoritmo de *pull-up* es sencillo, y por ello lo formalizaremos un poco más que el esquema anterior. Utilizaremos las funciones auxiliares:

raíz(X) : devuelve el nodo raíz del árbol X.

es-hoja(Y) : devuelve cierto si el nodo Y es una hoja y falso en caso contrario.

subárbol(Z,V) : devuelve el árbol correspondiente a la rama de Z etiquetada con V.

expandir(Z,A) : devuelve el árbol resultante de expandir Z un nivel usando A.

Veamos el algoritmo:

{S: Subárbol en curso $\wedge$ A: atributo a subir}
función Pull-up (S: `árbol-de-decisión`; A: atributo)
  devuelve T: `árbol-de-decisión`
var T: `árbol-de-decisión`;

si es-hoja (raíz(S)) **entonces** T:= Expandir (S,A)
sino (* es un nodo de decisión *)
  si $A = A_{raiz(S)}$ **entonces** T:=S
  sino
    para todo $v \in V(A_{raiz(S)})$ **hacer**
      S:= añadir-rama (S, Pull-up (subárbol(S,v),A), v)
    fpara todo;
    T:= Transponer (S)
  fsi
fsi
devuelve T

[16] Cualquier método es bueno: ID3, ID3 normalizado, RLM, etc. Por supuesto, si se utiliza RLM el árbol resultante no será equivalente a ID3.

ffunción Pull-up
{T es un subárbol consistente con S y con A como raíz}

La función **Transponer (S)**, dado que todos los subárboles de S están etiquetados con el mismo atributo (el que ha sido subido a todos ellos), lo intercambia con el de la raíz de S.

Existen métodos más sofisticados, como IDL, debido a W. van de Velde [VELD90], que usa hábilmente los tres operadores básicos en árboles de decisión: expansión, poda[17] y transposición. El proceso en IDL se realiza en dos fases:

1. Primero, se utiliza la expansión al estilo ID3 para clasificar la nueva instancia. Hay que notar que la medida de selección está basada en la *distancia entre particiones* de RLM.

2. Una vez se tiene una hipótesis discriminatoria consistente, se inicia un proceso de *revisión* del camino de discriminación de la instancia pero, esta vez, en sentido contrario al paso anterior, es decir de la hoja a la raíz. Aquí se utiliza el concepto de *relevancia topológica*: el proceso usa transposición para rehacer el camino de discriminación podando siempre que sea posible para así obtener uno substancialmente más corto.

Sumario

Los algoritmos presentados representan una clara evolución sobre el modelo original presentado por Quinlan. Parte de estas mejoras han sido apuntadas o realizadas por él mismo. Se pueden encontrar muchas comparaciones tanto teóricas (por lo menos, en lo que a coste se refiere) como empíricas, en varios dominios ya clásicos en el área. Ciertos problemas son comunes a todos ellos –como los presentes en los propios datos, que aquí se han supuesto ideales– y son tratados en otros apartados del libro, ya que conciernen a la formación de conceptos en general. Veamos una tabla a modo de resumen de los aquí expuestos:

Método	Medida	Objetivo	Incremental	Equiv. ID3	Operadores
ID3	Entropía	Coste Bondad	No	Sí	Expansión
ID4	Entropía	Equivalencia con ID3	Sí	No	Expansión Poda
ID4R	Entropía	Equivalencia con ID3	Sí	Sí	Expansión Poda
ID5	Entropía	Equivalencia con ID3	Sí	No	Expansión Poda Transposición
ID5R	Entropía	Equivalencia con ID3	Sí	Sí	Expansión Poda Transposición
IDL	Entropía Topología	Minimalidad topológica	Sí	No	Expansión Poda Transposición

[17] La poda es la inversa de la expansión.

2.4 La relevancia de los atributos

La naturaleza de la clasificación se ha estudiado ampliamente en el campo del aprendizaje, especialmente aquellos procesos que pueden ser realizados automáticamente por una máquina. Muchos sistemas expertos con éxito de la primera generación eran en la práctica clasificadores. Este tipo de sistemas usa un conjunto de reglas, representadas como árboles de decisión, para determinar la clase de una entrada dada en base a un conjunto de sus características. En el acercamiento clásico, el experto humano es el responsable de decidir cuáles son los *atributos relevantes* para la clasificación y posterior formación de reglas. Las limitaciones de este acercamiento, apuntadas por varios autores, (*e.g.* [HAYE84], [BAIM88], etc.), han estimulado el desarrollo de sistemas que puedan tomar la responsabilidad de decidir si un atributo es potencialmente útil o no, usando métodos de inferencia inductiva para descubrir los patrones o relaciones entre datos que puedan ser útiles en la formación de reglas para clasificaciones. En aprendizaje, varios son los métodos *heurísticos* que se han desarrollado para la evaluación de atributos en términos de su utilidad potencial (que se ha identificado con su relevancia) de cara a decidir la clase a la que pertenece una entrada. La gran mayoría estan basados en la teoría de la información clásica (*e.g.* [QUIN79], [QUIN86], [SCHL86], [BAIM88], [LOPE91], etc.).

2.4.1 El problema de la relevancia

En este apartado se hace una caracterización del problema de la relevancia de un atributo, y una definición formal de relevancia y atributos no relevantes o atributos *nought*.

Sea $\mathcal{U}$ un universo y C = $\{C_1, C_2, \ldots, C_m\}$ una clasificación (partición) de $\mathcal{U}$. Asúmase que los elementos de $\mathcal{U}$ se pueden representar por la n-tupla $(A_1(u), \ldots, A_n(u))$ ordenada de atributos (*i.e.* características mesurables) de los objectos en $\mathcal{U}$, con rangos $X_1, X_2, \ldots, X_n$, respectivamente. Formalmente hablando, cada A_i es una función con dominio $\mathcal{U}$ y rango X_i. Nótese que, en general,

$$\{(A_1(u), \ldots, A_n(u)) \mid u \in \mathcal{U}\}$$

es un subconjunto de

$$X_1 \times \ldots \times X_n$$

pero no necesariamente igual.

Expresado intuitivamente, contra más *información* proporciona un atributo a una clasificación, más relevante es para la clasificación. La manera natural de *mesurar* la información que un atributo proporciona es por medio de su capacidad de discriminar los elementos de $\mathcal{U}$. La relevancia de un atributo no es una propiedad absoluta, sino que es relativa a una clasificación o interpretación. Un atributo puede ser muy importante para una clasificación pero *irrelevante* en otra. Por ejemplo, el color de la piel de las personas no es importante para determinar (discriminar) el tipo de enfermedad cardiovascular que puedan tener, pero sí lo es para determinar su clasificación étnica. En la práctica, el conjunto inicial de atributos potencialmente útiles debe ser escogido por el experto.

Un atributo es relevante si tiene la capacidad de discriminar entre los elementos de $\mathcal{U}$. Como primera idea, podríamos pensar que ésta viene dada por el número de elementos que el atributo discrimina. Esto es, si la proporción de elementos de $\mathcal{U}$ discriminada por un atributo A_i es más

pequeña que la proporción correspondiente a otro A_j, entonces la relevancia de este último ha de ser superior a la del primero. Sin embargo, las cosas no son tan fáciles, como muestra el siguiente ejemplo.

Ejemplo

Sea $\mathcal{U}$ un universo que contenga los objetos $o_1, o_2, \ldots, o_{11}$, clasificados en C_1, C_2, C_3, C_4 por los atributos A_1, A_2, A_3, con un rango de valores de entre el conjunto $\{1, 2, 3, 4\}$.

Clase	*Objeto*	A_1	A_2	A_3
	o_1	1	2	1
C_1	o_2	1	3	1
	o_3	2	1	2
C_2	o_4	2	2	2
	o_5	2	2	3
	o_6	2	1	3
C_3	o_7	3	4	2
	o_8	3	1	1
	o_9	3	2	4
C_4	o_{10}	3	3	2
	o_{11}	4	3	2

Cada entrada de la tabla de clasificación especifica el valor para el objeto dado por la fila y el atributo dado por la columna. También se indica la clase a la que pertenece cada objeto.

Por sí solo, el atributo A_1 discrimina tres elementos (o_1, o_2 y o_{11}), dos más que A_2 y A_3 (los cuales solamente discriminan o_7 y o_9, respectivamente) [18]. No obstante, si observamos con un poco más de atención veremos que A_2 y A_3 son *más relevantes* para la clasificación, ya que el conjunto $\{A_2, A_3\}$ es totalmente discriminante[19], mientras que $\{A_1, A_3\}$ y $\{A_1, A_2\}$ no lo son. El otro conjunto totalmente discriminante es, evidentemente, $\{A_1, A_2, A_3\}$[20].

Así, no sólo no hay suficiente con el número de objetos sino que esta estimación es errónea. Por consiguiente, la relevancia de un atributo depende no solamente en la proporción de objetos que discrimina, sino en cómo *interactúa* con el resto de atributos. Cualquier definición formal de relevancia debe, por tanto, tener en cuenta ambos conceptos. Las definiciones siguientes estan extraídas de [BELA91] y [NÚÑE91b], y conforman un acercamiento teórico al problema de la relevancia.

Definición 2.1. El atributo A_i es más relevante que el A_j si, y sólo si, el mínimo número de atributos que se han de añadir a A_i para obtener un conjunto totalmente discriminante,

[18] Para encontrar qué objetos discrimina un atributo, búsquense aquellos para los cuales no existen otros objetos con idéntico valor para ese atributo en clases diferentes.

[19] Diremos que un conjunto de atributos es *totalmente discriminante* si los atributos que lo conforman son suficientes para discriminar *todos* los elementos de $\mathcal{U}$.

[20] Ya que se toma como hipótesis que el conjunto inicial de atributos es suficiente para clasificar todo el universo.

es menor que el número de atributos requeridos para A_j. Si este número es el mismo para ambos, concluiremos que son igualmente relevantes.

Esta definición se puede expresar como:

Sea $N = \{1, 2, \ldots, n\}$, y, por cada $S \subset N$, sea

$$\begin{array}{rcl} \mathrm{A}_S & = & \{A_s \mid s \in S\}, \\ \mathrm{S}_i & = & \{S \subset N \setminus \{i\} \mid \{A_i\} \cup \mathrm{A}_S \text{ es totalmente discriminante}\} \\ \mathrm{S}_j & = & \{S \subset N \setminus \{j\} \mid \{A_j\} \cup \mathrm{A}_S \text{ es totalmente discriminante}\} \end{array}$$

Entonces A_i es más relevante que A_j si, y sólo si,

$$min\{\#S|S \in \mathrm{S}_i\} < min\{\#S|S \in \mathrm{S}_j\}.$$

A_i es igualmente relevante que A_j si, y sólo si,

$$min\{\#S|S \in S_i\} = min\{\#S|S \in S_j\}.$$

Esta definición es generalizable de manera natural a conjuntos de atributos:

Definición 2.2. Sean S_1 y S_2 dos subconjuntos de N, con

$$\begin{array}{rcl} \mathrm{A}_{S_1} & = & \{A_s \mid s \in S_1\}, \\ \mathrm{A}_{S_2} & = & \{A_s \mid s \in S_2\}, \end{array}$$

entonces:

$$\begin{array}{rcl} \mathrm{S}_{S_1} & = & \{S \subset (N \setminus S_1) \mid \mathrm{A}_{S_1} \cup \mathrm{A}_S \text{ es totalmente discriminante}\} \\ \mathrm{S}_{S_2} & = & \{S \subset (N \setminus S_2) \mid \mathrm{A}_{S_2} \cup \mathrm{A}_S \text{ es totalmente discriminante}\} \end{array}$$

A_{S_1} es más relevante que A_{S_2} si, y sólo si,

$$min\{\#S|S \in S_{S_1}\} < min\{\#S|S \in S_{S_2}\}.$$

Los conjuntos de atributos A_{S_1} y A_{S_2} tienen la misma relevancia si, y sólo si,

$$min\{\#S|S \in S_{S_1}\} = min\{\#S|S \in S_{S_2}\}.$$

La idea intuitiva que subyace a la definición es que dos conjuntos de atributos, independientemente de su cardinalidad, son igualmente relevantes si tienen la misma capacidad de discriminación o, equivalentemente, si *la cantidad de información que les falta para ser totalmente discriminantes es la misma.*

Por tanto, para determinar si un conjunto dado de atributos es más relevante que otro tendríamos que generar, como mínimo, todos los conjuntos totalmente discriminantes (de relevancia máxima) que contengan alguno de los dos conjuntos dados. Pero este es un método impráctico e intratable desde el punto de vista de la complejidad. Así, la solución propuesta ha sido establecer heurísticas para evaluar la utilidad potencial de un atributo, de tal manera que escogen, sucesivamente, aquellos que llevarán a obtener árboles de decisión cercanos al óptimo. Las soluciones propuestas hasta ahora tienen varios elementos comunes, y su discusión cae fuera de este texto introductorio. Para un completo estudio de estas medidas y sus características, véase [BELA91].

La definición de relevancia para un conjunto dado de atributos $\mathcal{A}$ introduce una relación de equivalencia $\mathcal{R}$ en el conjunto potencia $\mathcal{P}(\mathcal{A})$, donde las clases de equivalencia estan formadas por conjuntos igualmente relevantes. Junto con la anterior definición, esto nos permite introducir un *orden total,* $\leq$, en el conjunto cociente de clases de equivalencia $\mathcal{P}(\mathcal{A})/\mathcal{R}$.

Definición 2.3. Sean A_{S_1}, A_{S_2} dos conjuntos de atributos y $[A_{S_1}]$, $[A_{S_2}]$ sus clases de equivalencia. Entonces:

$$[A_{S_1}] \leq [A_{S_2}]$$

si, y sólo si,

$$min\{\#S|S \in S_{S_1}\} \leq min\{\#S|S \in S_{S_2}\}.$$

Nótese que las clases $[\mathcal{A}]$ y $[\emptyset]$ son, respectivamente, el máximo y el mínimo con respecto a ese orden, *i.e.* $[\emptyset] \leq [A_S] \leq [\mathcal{A}], \quad \forall\, A_S \subset \mathcal{A}$. Otro punto importante es que si A_{S_1} es más relevante que A_{S_2}, entonces cualquier subconjunto de A_{S_1} es más relevante que cualquier otro de A_{S_2}. El problema, en la práctica, es obtener el *óptimo representante de* $[\mathcal{A}]$*, que clasifica el conjunto original* $\mathcal{U}$ *con el mínimo esfuerzo.* Este es el objetivo de las heurísticas antes mencionadas.

2.4.2 Los atributos nought

Se ha dicho ya que la relevancia de un conjunto de atributos no es inherente a ellos, sino que depende de la clasificación. Frecuentemente nos enfrentamos con situaciones en que un atributo o, en general, un conjunto de atributos, no son inportantes para un proceso de clasificación dado (aunque podrían serlo para otro), es decir, *no tienen relevancia para una determinada clasificación.* Esta relevancia nula sera referida como relevancia *nought* [SÁNC89]. A partir de ahora, nos referiremos también a los atributos no relevantes como atributos *nought.* En estos casos, estos atributos están ya dados y no pueden ser ignorados. Lo que se pretende es que no distorsionen la clasificación considerando sólo los no *nought.*

De acuerdo con la definición de relevancia, los atributos *nought* son aquellos sin capacidad de discriminación. Por tanto, cualquier conjunto *nought* tiene relevancia nula y es un elemento de $[\emptyset]$, el conjunto vacío.

Definición 2.4. Un conjunto de atributos A_n es *nought* si, y sólo si, $[A_n] = [\emptyset]$.

Nótese que cualquier conjunto de atributos *nought* A_n, añadido a un conjunto de atributos

A_S dado, no le altera la relevancia, es decir, la capacidad de discriminación de la clase $[A_S]$ es la misma que la de la clase $[A_n \cup A_S]$, $\forall\, A_S \subset A$.

Sea A_n un conjunto de atributos *nought*. Entonces $[A_n \cup A_S] = [A_S]$, $\forall\, A_S \in P(A)$

Ahora bien, el hecho de que, al añadir un conjunto de atributos cualquiera a otro, resulte un nuevo conjunto con la misma relevancia que el original *no* implica necesariamente que el conjunto añadido sea *nought*. Esto se puede observar en el siguiente ejemplo:

Ejemplo

Sea $\mathcal{U}$ un universo que contenga los objetos $o_1, o_2, \ldots, o_6$, clasificados en C_1 y C_2 por los atributos A_1, A_2, A_3 y A_4, con un rango de valores del conjunto $\{a, b, c\}$.

Clase	*Objeto*	A_1	A_2	A_3	A_4
C_1	o_1	a	a	a	a
	o_2	a	a	b	a
	o_3	c	a	c	a
C_2	o_4	a	b	b	c
	o_5	b	b	b	b
	o_6	c	a	c	b

Supongamos ahora que añadimos al conjunto $\{A_1, A_2\}$ el conjunto $\{A_3\}$. Si nos fijamos en la tabla, veremos que –entre otros– tenemos los siguientes conjuntos totalmente discriminantes:

- $\{A_1, A_4\}$
- $\{A_2, A_4\}$
- $\{A_3, A_4\}$
- $\{A_1, A_2, A_3\}$
- ...

Entre los no totalmente discriminantes encontramos:

- $\{A_1, A_2\}$
- $\{A_2, A_3\}$
- $\{A_1, A_3\}$
- ...

Por tanto, parece claro que el atributo A_3 *no es nought*, pues le falta menos —concretamente, por ejemplo, el atributo A_4— para ser totalmente discriminante que al conjunto vacío (que, como mínimo, necesita que se le añadan dos atributos). No obstante, podemos ver que los

conjuntos $\{A_1, A_2\}$ y $\{A_1, A_2, A_3\}$ tienen la misma relevancia (y, por tanto, están en la misma clase de equivalencia, como hemos visto).

Por consiguiente, en cada caso, se trata de encontrar el conjunto de atributos que nos clasifiquen el dominio con el mínimo esfuerzo. Para acabar de formalizar este concepto y, con él, el estudio de la relevancia, estableceremos, finalmente, el siguiente criterio:

Definición 2.5. El elemento de $[\mathcal{A}]$ de cardinalidad mínima es el optimal de $[\mathcal{A}]$.

Siendo precisamente éste el elemento a buscar por las medidas heurísticas.

2.5 Aprendizaje por observación y formación de conceptos

La metodología de la que se ocupa esta sección constituye la estrategia de aprendizaje inductivo más difícil y ambiciosa de las vistas hasta ahora. En su planteamiento original no presupone ningún conocimiento previo sobre lo que se quiere aprender. A diferencia de los algoritmos de aprendizaje de la sección anterior, en los que se inducía la descripción de un concepto a partir de la presentación de diferentes instancias de éste, y en algunos casos también de contraejemplos seleccionados, en este grupo de estrategias se parte de un conjunto de ejemplos de los que se puede inducir un número no preestablecido de conceptos. Tampoco existe un maestro que conozca los conceptos a aprender a *priori*, por esta razón se denomina a este tipo de aprendizaje **no supervisado** (*Unsupervised Learning*).

La creación de una clasificación de un conjunto de observaciones se puede tomar como la primera aproximación para desarrollar una teoría sobre éstas, por lo que es importante desarrollar técnicas que ayuden a realizar de manera automática estas labores. El objetivo de estas técnicas será descubrir *patrones comunes* entre los datos, que permitan separar los ejemplos en clases o jerarquías de clases. De éstas se podrán extraer caracterizaciones, o permitirán predecir características, o deducir relaciones útiles, es lo que se denomina *agrupación* (*clustering*).

Los métodos que vamos a describir, junto con el resto de mecanismos de aprendizaje inductivo, permiten reducir el *cuello de botella* que supone la adquisición y el refinamiento de bases de conocimiento para los sistemas basados en el conocimiento, transformándolos en herramientas más atractivas.

2.5.1 La componente psicológica

Todas estas técnicas parten de las ideas y teorías que ha desarrollado la psicología cognitiva sobre cómo los humanos establecemos las definiciones de las cosas y cómo caracterizamos grupos de objetos que consideramos que pertenecen a un mismo concepto [MEDI89], [LAKO87], [SMIT81]. Vamos a resumir brevemente la evolución de las teorías que han ido apareciendo en psicología para explicar cómo construímos categorías los humanos.

La visión clásica

La *visión clásica* en psicología sobre la categorización humana, se basa en que todas las instancias de una categoría tienen una característica fundamental en común que determina la pertenencia a esa categoría. Por lo tanto, una categoría quedaría representada por una lista de propiedades o características que individualmente son necesarias para la pertenencia de un objeto a ésta, y colectivamente son suficientes para determinar su pertenencia. Por ejemplo, la categoría *número primo* quedaría determinada por las propiedades "*ser un número natural*" y "*sólo ser divisible por sí mismo y por la unidad*". Si falla cualquiera de las dos propiedades no se es número primo, y el cumplir las dos determina serlo.

Muchos estudios han evidenciado la falta de solidez de esta visión de la categorización. Estos son sus principales problemas:

1. *Incapacidad para determinar las características que definen una categoría.* Tras varios estudios se ha podido comprobar que muchos conceptos, a pesar de que la gente piense que se pueden definir a partir de condiciones necesarias y suficientes, se escapan a una observación detallada. En estos ensayos, se intentó en varias áreas de la ciencia que sus expertos dieran conjuntos de propiedades para varios conceptos de sus áreas de conocimiento, que cumplieran todas las instancias que quedaban englobadas en ellos, sin conseguirlo.

2. *Gradación entre los ejemplos.* Dado cómo se define la categorización, al haber un conjunto de propiedades que representan una categoría, cualquier elemento de ésta es tan bueno como otro para tomarlo como ejemplo, ya que todos comparten las mismas propiedades. No obstante, las investigaciones evidencian que existen ejemplos mejores que otros dentro de las categorías[21]. Esto define efectos de *tipicalidad* entre los ejemplos de una categoría.

3. *Existencia de asignaciones ambiguas.* Esta visión clásica de la categorización presupone una forma no ambigua de determinar a qué concepto pertenece cualquier ejemplo que se nos presente, sólo hace falta comprobar las características que los definen. Sin embargo, hay ejemplos claros de situaciones en las que es difícil decidir[22].

La visión probabilística

Todos estos problemas han hecho evolucionar a las teorías psicológicas hacia un punto de vista *probabilístico* de la estructura de las categorías. Esta visión considera la estructura de las categorías como algo *difuso* y supone que éstas se organizan a partir de un conjunto de atributos correlacionados que son sólo rasgos característicos, pero no propiedades que definen la categoría.

Esta visión de las categorías resuelve algunos de los problemas de la visión clásica. Ahora existe una gradación entre los ejemplos de una categoría, ya que los miembros no tienen porqué cumplir todas las características, hay miembros más típicos y menos típicos. También se pueden explicar los ejemplos a los que es difícil de asignar a una clase, pues pueden poseer

[21] Todo el mundo estaría de acuerdo en que una vaca ejemplifica mejor a un mamífero que una ballena.
[22] ¿Debería considerarse a un ordenador como un electrodoméstico?

características que pertenezcan a la clase, pero no las suficientes para permitir una asignación clara.

Este punto de vista supone que las categorías se organizan respecto a lo que se llama *parecido familiar* (*family resemblance*). Éste se podría definir como un elemento ideal, que resume las características de todos los objetos de la clase, al que usualmente se denomina *prototipo*. La asignación a una clase se decide en base a la similaridad de un ejemplo con el prototipo de la clase. La base de esta teoría se encuentra en la idea de que con el tiempo la gente abstrae de los ejemplos que se van encontrando su tendencia central y se usa ésta como representación del concepto.

Extendiendo el punto de vista anterior, otras teorías apuestan por una representación de las categorías mediante un grupo de ejemplos en lugar de un único elemento que resuma las propiedades de la clase. Las investigaciones realizadas sobre la comparación de los dos puntos de vista han dado como conclusión que la representación como prototipos es adecuada para representar una forma de asignación más inexperta, y que la basada en ejemplos sería utilizada por sujetos con mayor experiencia. No obstante, las dos teorías se basan en el mismo principio, la pertenencia a una clase se determina a través de la similaridad de un ejemplo con el prototipo o con el grupo de ejemplos.

La concepción de similaridad en la que se basan estas dos teorías se fundamenta en cuatro principios básicos:

1. La similaridad entre dos elementos es una función creciente de los atributos que comparten y decreciente de los que difieren.
2. Todos los atributos pueden ser tratados como independientes.
3. Todos los atributos que se usan pertenecen al mismo nivel de abstracción.
4. Un concepto es más o menos equivalente a su lista de propiedades.

Las investigaciones han hecho ver lo erróneo de estas suposiciones en la mayoría de los casos reales en lo que respecta a la similaridad y a la forma de tratar la información que caracteriza a las categorías.

El primer problema de las teorías basadas en prototipos es que tratan a los conceptos de manera independiente del contexto. Al extraer información únicamente de la tendencia central de la clase, tampoco tienen en cuenta información que, como se ha evidenciado experimentalmente, sí usan las personas para categorizar, como el tamaño de la clase, la variabilidad de los ejemplos o la correlación entre los atributos. Tampoco son capaces de distinguir entre categorías más difíciles y mas fáciles de aprender, ya que tal como se modelizan las categorías, las que son linealmente separables[23] deberían ser más aprendibles que las que no, habiéndose demostrado experimentalmente que no es así.

Las teorías basadas en ejemplos salen mejor del paso, ya que guardan mas información que las basadas en prototipos, manteniendo información sobre más detalles y siendo más sensibles al contexto. Además son capaces de inferir información basándose en información parcial, ya

[23] Se dice que dos clases son linealmente separables si existe una función lineal capaz de establecer la frontera entre ellas.

que los modelos basados en ejemplos intentan no descartar información que permita hacer predicciones.

No obstante, el mayor problema de las teorías basadas en la visión probabilística de la categorización se encuentra en su concepción de similaridad. A pesar de ser el concepto de similaridad bastante intuitivo, implica muchas más cosas que una simple coincidencia de atributos. La importancia de los atributos que describen a los ejemplos puede variar dependiendo del contexto en que se encuentren, haciendo pesar a unos atributos más que otros y estableciendo relaciones entre ellos. Se rompe de esta manera la idea de que los atributos que forman las categorías son independientes entre sí. También se ha de establecer de alguna manera cuáles son los atributos que se deben usar para categorizar un conjunto de ejemplos, ya que el número de ellos que se puede dar, si no se pone ninguna restricción, puede ser virtualmente inacabable. Por lo tanto, la categorización dependerá de la definición de qué atributos son necesarios y cuál es la relación que establece la importancia de cada uno y será esto lo que determine como se mide la similaridad entre los ejemplos.

Categorización basada en teorías

En las teorías más recientes se ha desarrollado la idea, apoyada por la evidencia experimental, de que las categorías se desarrollan en torno a teorías que se forma la gente sobre el mundo, que apoyan la existencia de las diferentes categorías que usan. A partir de estas ideas, se busca hallar una explicación de cómo los humanos creamos categorías, objetivo que no se alcanzaba con las anteriores visiones de la categorización.

La categorización dirigida por teorías es capaz de dar explicación a la formación de categorías que son difícilmente asumibles bajo el punto de vista de la similaridad. Por ejemplo, una categoría formada por un cepillo de dientes, ropa interior y un pijama sólo toma sentido si decimos que hablamos de "*cosas que llevar para pasar una noche en casa de un amigo*".

Experimentos han demostrado que la noción de similaridad no es algo absoluto y es muy dependiente de los ejemplos y de las ideas que los relacionen. Por ejemplo, Medin y Shoben [SHOB88] descubrieron que los términos *cabello blanco* y *cabello gris* se tomaban como más similares que *cabello gris* y *cabello oscuro*, pero, en cambio, *nubes blancas* y *nubes grises* se consideraban menos similares que *nubes grises* y *nubes negras*. Todo ello se explica porque *cabello blanco* y *cabello gris* están relacionados mediante la idea del *envejecimiento*, mientras que las *nubes blancas* y las *nubes grises* no.

No obstante las teorías no son suficientes para explicar la categorización, aún es necesario mantener la similaridad como herramienta, pero con una concepción de ella radicalmente distinta a la usada en la teoría de prototipos. Para que la similaridad sea coherente con esta nueva noción de categorización ha de cumplir cuatro reglas:

1. Es necesario incluir en las descripciones atributos, relaciones, ...
2. Las propiedades no suelen ser independientes, sino que están interrelacionadas.
3. Las propiedades suelen hallarse en diferentes niveles de abstracción.
4. Los conceptos son algo más que una lista de propiedades.

Añadiendo esta nueva visión de la similaridad, se consigue adaptar la teoría basada en prototipos a una teoría en la que la categorización está guiada por concepciones y teorías y que determina la pertenencia de los ejemplos mediante una exploración mas profunda de sus características.

2.5.2 Aproximaciones computacionales

Paralelamente a los estudios de los psicólogos se han desarrollado algoritmos que en parte se pueden encuadrar en algunas de las teorías que modelizan la caracterización humana de las que se ha hablado. Estos algoritmos pertenecen a áreas diferentes, pero mantienen el objetivo común de extraer agrupaciones a partir de ejemplos, de las que extraer información sobre la estructura que subyace bajo los ejemplos y las relaciones que existen entre los atributos que los describen.

La mayoría de ellas parten de una representación común del conocimiento del que se pretende extraer categorías útiles. Esta se suele basar en un conjunto de ejemplos descritos mediante grupos de pares *atributo–valor*[24]. Sobre estas descripciones se definen los criterios que guían el proceso de aprendizaje. Los tipos de atributos que se pueden utilizar son variados. Tres destacan en la literatura [MICH84a]:

Descriptores categóricos o nominales: El valor de este tipo de descriptores consiste en símbolos entre los que no existe ningún tipo de ordenación o jerarquía, (e.g.: el color del pelo de una persona: rubio, castaño, pelirrojo).

Descriptores lineales o cuantitativos: Los valores corresponden a un conjunto totalmente ordenado, incluyendo tanto valores discretos (e.g.: meses del año), como contínuos (e.g.: peso).

Descriptores estructurados: Los valores de estos atributos forman una jerarquía que representa la relación de generalidad entre los valores, por ejemplo ver figura 2.20.

Los valores de los atributos de cada ejemplo pueden mostrar diferentes estados, dependiendo de la calidad de la información o de la relación entre los diferentes atributos que los describen. Son los siguientes:

Valores normales: Se refieren a los valores habituales de los atributos, son valores conocidos.

Valores perdidos (*missing values*): Se refieren a valores que se desconocen, por pérdida o error.

Valores irrelevantes (*nought values*): Valores que no son importantes para describir un ejemplo en particular. Esto incluye información adicional de la relación entre un atributo y el resto de los del ejemplo.

[24] Muchas críticas se han hecho sobre la limitación de este tipo de representaciones. Actualmente se está incluyendo la posibilidad de trabajar con objetos compuestos descritos mediante la combinación de relaciones y atributos [THOM91].

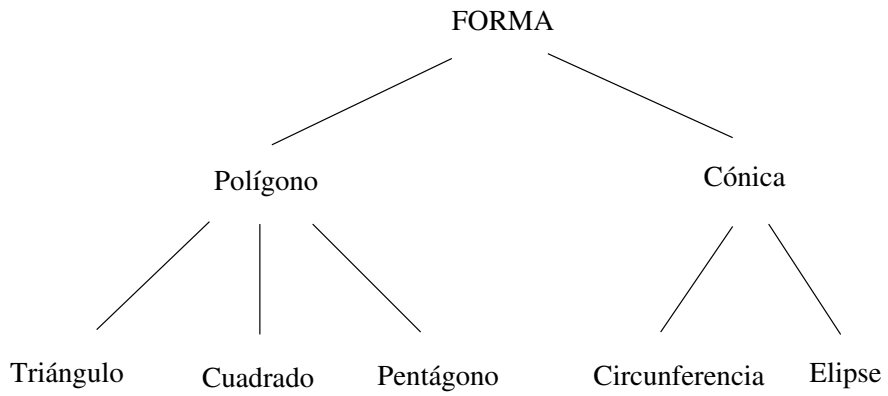


Figura 2.20: Ejemplo de atributo estructurado

Valores ilegales o prohibidos: Aparecen en atributos que están relacionados con otros y que dependiendo de los valores de estos últimos pueden poseer un valor o no. Por ejemplo, hablando de personas, la propiedad *número de partos* estaría relacionado con el atributo *sexo*, que en el caso de tener el valor *varón* dejaría sin sentido al primer atributo.

Estos estados especiales de los valores de los atributos tienen diferentes tratamientos que consiguen, en parte, incluir la información o falta de información que presenta el conjunto de datos.

Dos son las áreas en las que se han desarrollado estos algoritmos: la taxonomía numérica y el aprendizaje automático.

Las técnicas de aprendizaje automático pretenden ser una evolución y una mejora de las deficiencias que aparecen en la taxonomía numérica. En las siguientes secciones se describen las técnicas aparecidas en las dos áreas, centrando nuestra atención en el aprendizaje automático.

2.5.3 Taxonomía numérica

El primer área en el que se ha estudiado el análisis y extracción de información a partir de ejemplos ha sido la *taxonomía numérica*, en la que se han desarrollado múltiples algoritmos para la agrupación de objetos en clases. De estas técnicas han partido alguna de las ideas y criterios que usan los algoritmos de aprendizaje no supervisado.

Los algoritmos que ha desarrollado la taxonomía se basan en la agrupación de objetos similares en una misma categoría. Para la medición de la similaridad entre objetos utilizan funciones basadas en las descripciones de los objetos. Existen muchos tipos de distancias que se pueden utilizar para estimar la similaridad o disimilaridad entre las descripciones de los objetos [DUBE88], con variaciones según el tipo de los atributos. De entre ellas se puede destacar:

Métrica de Minkowski: Son métricas definidas sobre el espacio $\mathbb{R}^n$, usadas preferentemente

para atributos lineales, cuya expresión general es:

$$d(i,k) = (\sum_{j=1}^{d} |x_{ij} - x_{kj}|^r)^{1/r} \quad para\ r \geq 1$$

Donde x_{ij} y x_{kj} son los valores del atributo j de las observaciones i y k. De estas medidas las más utilizadas son la *euclídea* (n=2) y la de *hamming* (n=1).

Distancia de Mahalanobis: Esta distancia también se define sobre atributos lineales y tiene la siguiente expresión:

$$d(i,k) = (x_i - x_k)^T \cdot \varphi^{-1} \cdot (x_i - x_j)$$

Donde φ es la matriz de covariancias entre los atributos, incorporando así en la medida las correlaciones existentes entre los datos.

Distancia de χ^2: Está pensada para variables categóricas. Para poder computarla hace falta transformar cada columna de datos correspondiente a una variable categórica, en tantas columnas como modalidades tenga. Para cada elemento se pone un 1 en la columna correspondiente a la modalidad que tenga y 0 en el resto. Por lo tanto, si la propiedad k posee c modalidades, ésta se transforma en c propiedades binarias.

Siendo C el número de modalidades de una propiedad, x_j el número de objetos que poseen la modalidad j, y n el número total de objetos, la distancia de χ^2 para dos individuos y una propiedad dada sería:

$$d(i,k) = \frac{1}{n} \cdot \sqrt{\sum_{m=1}^{C} \frac{(x_{im} - x_{km})^2}{x_j}}$$

Distancia del coseno: Se basa en las propiedades de vectores en un espacio euclídeo. Mide el coseno del ángulo de dos vectores en un espacio N-dimensional. Su expresión es la siguiente, siendo N el número de propiedades:

$$d(i,k) = \frac{\sum_{j=1}^{N} x_{ij} \cdot x_{kj}}{\sqrt{\sum_{j=1}^{N} x_{ij}^2 \cdot \sum_{j=1}^{N} x_{kj}^2}}$$

Cuando todas las propiedades son binarias (toman el valor 0 o el 1) se le puede dar una interpretación no geométrica. Tomando el sumatorio del denominador como el número de atributos comunes entre las dos instancias y el denominador como la media geométrica del número de atributos que posee x_i y x_k, entonces la medida se podría interpretar como la relación de atributos comunes que poseen ambas instancias.

Los algoritmos

A partir de estas medidas se definen algoritmos que construyen jerarquías a partir del conjunto de ejemplos, *métodos jerarquicos*. Pueden usar estrategias *Bottom–up*, *métodos aglomerativos*, o *Top–down*, *métodos divisivos*.

El algoritmo aglomerativo más típico consiste en ir creando una jerarquía calculando la similaridad entre todos los objetos y agrupando a cada paso la pareja más similar, creando con ésta un nuevo objeto y substituyendo los dos primeros objetos por el nuevo.

Los algoritmos divisivos realizan el paso inverso, parten del conjunto de objetos como una sola clase y a cada paso deciden la partición de las clases que se tiene hasta que todas las clases se componen de un único elemento.

El resultado de ambos tipos de estrategias es el mismo, un árbol que indica el proceso de unión del conjunto de objetos desde una clase por objeto hasta una clase con todos los objetos. No se obtienen descripciones, y es labor del analista de datos escoger el nivel del árbol que dé agrupaciones útiles.

Alternativos a estos métodos se han desarrollado algoritmos que únicamente particionan el conjunto de datos, son los llamados métodos de *optimización* o *partición*. Éstos utilizan una medida de calidad sobre las agrupaciones para guiar la búsqueda de la partición que mejor se adapte a los datos[25]. Las medidas más frecuentes tratan de maximizar la similaridad entre los objetos de cada agrupación a la vez que minimizan la similaridad entre los grupos. En muchos casos este tipo de medidas no se han mostrado suficientemente efectivas, por lo que algunos algoritmos necesitan que el usuario explicite el número de clases que se han de formar.

A pesar de la amplia utilización de estos algoritmos en estadística para análisis de datos, se han vertido muchas críticas sobre la efectividad y la corrección de estos métodos, sobre todo cuando la labor consiste en extraer información que permita caracterizar los datos, o predecir propiedades.

Su principal defecto son las medidas de similaridad utilizadas, pues la mayoría están demasiado orientadas a datos numéricos, presentando muchos problemas a la hora de analizar datos no numéricos. Además, esta función sólo tiene significado en la medida en que los atributos que se han escogido son relevantes para la caracterización del conjunto de datos y de las diferentes clases que se pretenden descubrir, teniendo todos ellos el mismo peso en el proceso de determinar las clases. A esto hay que añadir que estas medidas no suelen incluir información sobre el contexto que pudiera ser útil para el proceso de clasificación.

La mayoría de los métodos sólo dan información sobre la similaridad entre los objetos y no ofrecen una caracterización o explicación de las observaciones y las agrupaciones. Unido a ésto, estos métodos dejan la mayor parte del trabajo de búsqueda de la mejor partición y de análisis de los resultados al usuario.

Como se verá a continuación los algoritmos y metodologías desarrolladas en el área del *aprendizaje automático* intentan facilitar el tratamiento de valores no numéricos (bastante frecuentes en los dominios de aplicación de la Inteligencia Artificial) y tratan de incluir parte de la labor de búsqueda en el espacio de posibles particiones que realiza el usuario.

[25] Hay 2^n maneras posibles de particionar un conjunto de datos.

2.5.4 Técnicas de aprendizaje automático

En la Inteligencia Artificial el aprendizaje no supervisado se ha intentado ver desde un punto de vista menos numérico, adoptando las ideas surgidas de la psicología cognitiva y construyendo modelos computacionales de cómo los humanos categorizamos y construímos conceptos a partir de grupos de objetos.

El origen de estos métodos parte también del deseo de intentar resolver los problemas que aparecen del uso de los algoritmos de taxonomía numérica, por lo que algunos de los algoritmos que se han desarrollado utilizan las ideas de estos métodos, pero introduciendo mejoras respecto a la información que se utiliza para la agrupación de los objetos, los criterios que permiten decidir la formación y la coherencia de una clase y la caracterización y la explicación de los resultados. También se ha intentado incorporar en los algoritmos parte de la labor de búsqueda y análisis que los métodos de taxonomía numérica dejaban en manos del usuario.

El punto de partida de todos estos métodos es también un conjunto de datos caracterizados mediante pares atributo-valor al que se le puede añadir información relevante sobre el dominio de clasificación como restricciones, propiedades de los atributos (relaciones causa-efecto, correlaciones, ...) y criterios para evaluar la calidad de las agrupaciones resultantes. Se ha tenido presente también que los dominios sobre los que se habrá de trabajar no tienen que estar compuestos únicamente por datos numéricos.

El resultado puede ser un conjunto o una jerarquía de clases caracterizadas mediante los atributos más relevantes[26] de entre los usados para describirlas y los valores que toman. El tipo de descripción varía dependiendo de los métodos. Los hay que dan como resultado una conjunción de atributos necesarios y suficientes para la pertenencia a cada clase adoptando la visión clásica de la caracterización. Otros adoptan la visión probabilística, dando como resultado un conjunto de características suficientes que con cierta probabilidad se deben poseer para pertenecer a una clase.

En algunos métodos se busca la posibilidad de poder predecir con la máxima exactitud los atributos de los objetos de una clase conociendo la clase a la que pertenecen, en otros se busca que la caracterización permita clasificar futuras instancias en vistas a utilizar esta caracterización como base de conocimiento.

Los métodos de aprendizaje no supervisado se han dividido en dos grupos teniendo en cuenta si la adquisición se realiza de forma incremental o no. Ambas metodologías tienen sus ventajas e inconvenientes. A la variante no incremental se la ha denominado **agrupación conceptual** (*conceptual clustering*), a la incremental **formación de conceptos** (*concept formation*). A continuación se describirá en detalle cada una de las dos, junto a los principales sistemas a los que han dado lugar.

2.5.5 Agrupación conceptual

El término agrupación conceptual se debe a Michalski [MICH80a]. Él lo define como:

> "Agrupar objetos en clases conceptualmente simples basadas en los valores de los atributos tomando en consideración todo conocimiento acerca de las relaciones

[26] Otro problema no menos importante es el de decidir qué información es la más relevante.

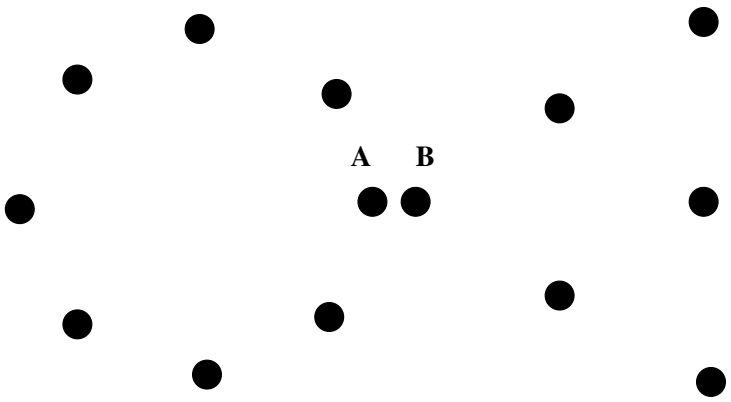


Figura 2.21: Grupo de objetos

> semánticas entre los atributos de los objetos o cualquier concepto global que pueda ser usado para caracterizar las clases que se forman." [MICH84b][MICH86]

Por lo tanto, el rasgo distintivo de la agrupación conceptual es intentar introducir la mayor cantidad de conocimiento sobre el contexto en el se quiere realizar el aprendizaje que pueda ser útil.

El origen de estos métodos parte de la constatación de la falta de contexto de las típicas medidas de similaridad. Éstas sólo tienen en cuenta a la hora de contrastar dos objetos los valores de sus atributos, no teniendo en consideración los conceptos que pueden ayudar a describirlos.

Las características que permiten describir a un grupo de objetos como pertenecientes a una categoría no se encuentran únicamente en el contraste de las propiedades que poseen cada par de objetos. Por lo tanto, hace falta más información para llegar a construir categorías[27].

La agrupación conceptual pretende asignar los objetos a clases no en base a una distancia entre ellos, sino a su pertenencia a cierto concepto que les da sentido, es lo que se denomina pertenecia conceptual (*concept membership*). Debido a esto, las tareas de división y de clasificación de los objetos no son independientes entre sí. Una división en clases de un grupo de objetos sólo será buena si y sólo si existe una buena interpretación de las clases.

Las funciones a que dan lugar todas estas ideas pasan de ser funciones que toman como parámetro únicamente los dos objetos a comparar ($f(A,B)$) a ser funciones de estos dos objetos, de los objetos con los que están relacionados, los que denominaremos entorno (E) y de un conjunto de conceptos disponibles para describirlos C ($f(A,B,E,C)$)[28]. La generación de este tipo de funciones se ha llevado a cabo desde muchos puntos de vista diferentes que van desde las funciones de la Teoría de la Información pasando por las aproximaciones probabilísticas hasta las propias funciones de similaridad utilizadas en los métodos estadísticos incluyendo información sobre el dominio dentro de su cálculo.

[27] Michalski afirma que las medidas de similaridad son incapaces por sí solas de captar las propiedades de forma (*Gestalt*) de los grupos de objetos.

[28] Michalski denomina a esta función cohesión conceptual (*Conceptual cohesiveness*).

En el ejemplo de la figura 2.21 se puede observar que una medida de similaridad típica, que no tuviera en cuenta ninguna información adicional, agruparía a los objetos A y B en la misma clase, y que una medida como la descrita en el punto anterior que incluyera los conceptos de las figuras geométricas no lo haría.

Para ilustrar estas ideas se estudiarán tres modelos muy diferentes entre sí en lo que respecta a las suposiciones básicas de las que parten, las restricciones que plantean y a los resultados que desean obtener. El primero de ellos (**CLUSTER**) se basa en la creación de categorías descritas en base a propiedades suficientes y necesarias utilizando una función a optimizar sobre las descripciones que se van creando. El segundo (**WITT**) basa su algoritmo en funciones tomadas de la Teoría de la Información favoreciendo clases con descripciones menos rígidas, más acorde con las tendencias de la *psicología cognitiva* incluyendo operadores que permiten modificar dinámicamente las clases obtenidas. La tercera aproximación (**AUTOCLASS**) se basa en la aplicación del teorema de Bayes y las funciones de distribución que presentan los atributos que describen los datos. Las categorias que se obtienen no son disjuntas y las observaciones tienen un grado de pertenencia a cada una de ellas.

CLUSTER

Esta metodología es realmente toda una generación de herramientas que han ido dejando atrás restricciones y suposiciones, ampliando el ámbito de trabajo hasta adoptar todas las ideas de la agrupación conceptual.

La primera herramienta de esta familia que se puede incluir dentro de la agrupación conceptual es **CLUSTER/2**[29], cuyo resultado es jerarquías de clases formadas por conceptos disjuntos, descritos a partir de conjunciones de atributos. Esto supone una restricción al tipo de conceptos que se pueden adquirir. Este tipo de agrupación conceptual se denomina **agrupación conceptual conjuntiva** (*Conjunctive Conceptual Clustering*). Las caracterizaciones están formadas por condiciones suficientes y necesarias para la pertenencia a las diferentes clases. Como ya se ha visto (ver 2.5.1), esta restricción limitará mucho el tipo de dominios en los que se puede trabajar.

La descripción de las observaciones se realiza mediante pares atributo-valor como los comentados en 2.5.2 y la función sintáctica de distancia utilizada para evaluar la similaridad de las clases se basa en la suma de las distancias individuales de los atributos que las describen. Esta distancia se diferencia respecto a los diferentes tipos de atributos que se utilizan:

- Si los atributos son categóricos su diferencia es 0 si corresponden al mismo valor y 1 en caso contrario.
- Si los atributos son lineales su distancia es el valor absoluto de la diferencia entre sus valores normalizada por el rango de valores del atributo.
- Si los atributos son estructurados dependerá de los niveles de la jerarquía. La distancia entre los valores de las hojas de ésta dependerá de si son categóricos o lineales.

La descripción de las clases se basa en conjunciones de condiciones lógicas. Cada condición

[29] Esta versión es la sucesora de **CLUSTER/PAF**.

es denominada sentencia relacional (*relational statement*) o selector, y está compuesta por un atributo, un operador relacional y uno o varios valores.

En el caso de los atributos categóricos los operadores relacionales se reducen a la igualdad ($=$) y desigualdad ($\neq$) y se pueden expresar disyunciones de valores, como por ejemplo:

$$\begin{aligned} color &= verde \vee amarillo \\ forma &\neq cuadrada \vee rectangular \end{aligned}$$

En el caso de los atributos cuantitativos los operadores se amplian con $\geq$, $>$, $<$, $\leq$ y el operador de rango .., como por ejemplo:

$$\begin{aligned} altura &> 1.5 \\ temperatura &= 15..25 \end{aligned}$$

La conjunción de selectores es denominada **complejo lógico** (*logical complex (ℓ-complex)*). Una observación o satisface un complejo lógico cuando cumple todos sus selectores, y a todas las observaciones que cumplen un ℓ-complejo se las denomina un **conjunto de complejo** (*set complex* (s-complex)). Al conjunto de complejos que cubren a una serie de observaciones se los denomina **estrella** (*star*). Se define también una medida de calidad sobre los complejos denominada **dispersión** (*sparseness*) que mide la relación entre la generalidad de un complejo (los objetos diferentes que puede describir) y el número de observaciones que realmente la cumplen.

El algoritmo utilizado por CLUSTER/2 se basa en la optimización de los *s-complejos* y *estrellas* respecto al número de observaciones que los cumplen. Esta optimización es llevada a cabo mediante una serie de operadores que permiten la *creación* de s-complejos a partir de observaciones, la *unión* de complejos para generar complejos que cubran la unión de observaciones, y la *generalización* y *especialización* de complejos.

Los criterios que se siguen para esta optimización se basan en una función de evaluación que mide la calidad de las agrupaciones obtenidas. El mayor problema consiste en decidir qué parámetros se utilizan para medirla.

Está claro que las descripciones que se obtengan tienen que ser simples para poder darles una interpretación clara, pero esto puede entrar en contradicción con la necesidad de que los datos queden cubiertos adecuadamente por las descripciones ya que éstas pueden necesitar cierta complejidad para hacelo con precisión.

Nos encontramos con el mismo problema que en taxonomía numérica al intentar minimizar la similaridad entre los objetos de una clase y maximizar la disimilaridad entre las diferentes clases. La optimalidad de las clases obtenidas, por lo tanto, debe ser un equilibrio entre la simplicidad de las caracterizaciones que se extraen y la adecuación de estas a las observaciones.

En CLUSTER/2 se han ampliado las medidas que se utilizan para medir la optimalidad de la clasificación que se va calculando, y su combinación es utilizada como criterio a maximizar. Estas medidas son:

- La adecuación de las agrupaciones a las observaciones.
- La simplicidad de las descripciones de las agrupaciones.
- La distancia entre las agrupaciones.
- La capacidad de discriminación.
- La reducción de dimensiones.

Para medir la *adecuación* de las observaciones a las descripciones se utiliza la medida de dispersión entre los complejos comentada anteriormente. La *simplicidad* de las descripciones se calcula como el número total de *selectores* que hay en todas las descripciones. Para hallar la *distancia* entre las agrupaciones suma el número de complejos que hay en las diferentes descripciones que no intersectan entre sí, favoreciendo de esta manera las agrupaciones con mayor número de propiedades diferentes. La capacidad de *discriminación* se calcula como el número de atributos que toman valores diferentes en todas las clases. La *reducción* de dimensiones se mide como el número mínimo de atributos necesarios para distinguir entre las diferentes clases.

El algoritmo se basa en la optimización de una función que combina todos estos criterios. Cada uno tiene un peso que indica su importancia a la hora de evaluar una clasificación. Estos pesos vienen indicados en la que se denomina **función de evaluación lexicográfica con tolerancias** (*Lexicographical Evaluation Functional with tolerances* (LEF)). Esta se define como una lista de pares *criterio-tolerancia*, en la que en cada par se indica el grado de cumplimiento que deben observar las clasificaciones respecto cada criterio. Dado un conjunto de clasificaciones que compiten, se escogerá la clasificación que cumpla todos los criterios a la vez.

El algoritmo utilizado para construir una clasificación a partir de un conjunto de datos consta de dos partes. Un primer paso que construye las clases bases que particiona de manera óptima, según la función de evaluación (LEF), a partir del conjunto de observaciones y un segundo paso que construye una jerarquía a partir de estas clases base.

El algoritmo básico para particionar las observaciones iniciales es el siguiente:

Partimos de:

- Un conjunto de observaciones (O).
- El número de clases que se quiere obtener (k).
- El criterio de evaluación (LEF).

Algoritmo:

1. Se determina un conjunto inicial de k semillas iniciales del conjunto O que se pueden escoger al azar o según algún criterio predeterminado.
2. Se genera una *estrella* para cada una de las semillas. Es decir, un conjunto de ℓ-*complejos* que incluyen a cada semilla sin incluir al resto.

3. Se modifican las *estrellas* para que sean disjuntas entre sí, haciendo más específicos los *complejos* que las forman de manera que no haya intersecciones entre ellos.

4. En este momento se tiene una posible partición de los objetos representada por k *estrellas* disjuntas. Si es el primer paso del algoritmo esta partición se guarda, si no lo es se aplica la función de evaluación (LEF) para ver si la mejora respecto a alguna de las mejores particiones previas. En el caso de que varias particiones satisfagan la función de evaluación se retienen todas. El algoritmo terminaría en este punto si después de un número predeterminado de pasos no aparece ninguna partición que mejore a la que es óptima en este momento.

5. Si no se ha llegado al óptimo se seleccionan nuevas semillas, una para cada clase. Se utilizan dos criterios, o o se escoge el objeto dentro de una clase más cercano a su centro geométrico, o el más alejado. La primera estrategia se elige siempre que esto incremente la calidad de la partición, cuando esto deja de suceder se eligen los objetos más alejados. A partir de aquí se repite el algoritmo desde el paso 2.

El resultado del algoritmo[30] son k ℓ-complejos que cubren todas las observaciones de manera disjunta y que maximizan los criterios de la función LEF.

La generación de la jerarquía se realiza aplicando recursivamente el algoritmo de generación de clases a cada una de las clases obtenidas inicialmente hasta que cierto criterio determina que el número de niveles es suficiente. La jeraquía es creada por lo tanto de las clases más generales a las más específicas (*top-down*).

El sucesor de CLUSTER/2 es CLUSTER/S [MICH86]. En éste se amplia la potencia expresiva de la representación de las observaciones pasando de una representación atributo-valor a una basada en el cálculo de predicados de primer orden (CP_1) que es denominada **Cálculo de Predicados con Anotaciones** (*Annotated predicate calculus (APC)*). Esto le permite hacer descripciones de objetos estructurados (compuestos por partes).

El APC incluye al cálculo de predicados anotaciones en cada predicado, variable y función indicando, entre otras informaciones, su tipo y los atributos que están relacionados con él. Ésto amplía la capacidad expresiva de los ℓ-complejos que se utilizaban en CLUSTER/2, dando además la capacidad de realizar deducciones entre atributos.

En esta mejora se hace énfasis en el *conocimiento de respaldo* (*Background Knowledge*) que es necesario incluir para conseguir una mejor clasificación. Además de la función de evaluación (LEF) ya utilizada, se incluye nuevo conocimiento que consiste en una red de objetivos de clasificación, reglas de inferencia y heurísticas para deducir nuevos descriptores más generales a partir de los que se utilizan en la descripción de los objetos, definiciones de los dominios de valores de los atributos y sus tipos. A esta red se la denomina **red de dependencia de objetivos** (*Goal Dependency Network* (GDN)).

A la hora de realizar una clasificación se incluye como dato de entrada un objetivo que junto con la información de las anotaciones y la GDN es utilizado para guiar al algoritmo de clasificación y que permite deducir a partir de los atributos existentes nuevos atributos que permiten mejorar la clasificación y ayuda a decidir cuáles son los atributos más útiles.

[30] La parte del algoritmo que supone mayor coste es la generación de las *estrellas* disjuntas a partir de las semillas, tiene coste exponencial. Se puede consultar [MICH84b] para conocer las heurísticas aplicadas para reducir este coste.

WITT

La propuesta de la metodología anterior caería dentro de la visión clásica de la categorización desde el punto de vista de la psicología cognitiva, pero ya se ha visto que suponer que las categorías deben estar representadas por condiciones necesarias y suficientes es algo demasiado restrictivo y que realmente la categorización humana está muy lejos de ser así.

Este método se fundamenta en cuatro resultados de la psicología cognitiva para apoyar sus hipótesis:

- Las categorías tienden a poseer miembros que no se describen por características suficientes y necesarias. Es lo que se denota como *polimorfia*.
- Las categorías tienen una distribución entre sus miembros (ver 2).
- Las categorías pueden ser representadas mediante las *intercorrelaciones* y relaciones entre los atributos que las describen. El descubrir y utilizar estas relaciones puede ser importante para comprender la estructura de cada categoría y comprender su naturaleza.
- Las categorías surgen de su *contraste* con las demás. Cada categoría tiene sentido por lo que la diferencia de las que conviven con ella en el mismo contexto.

La representación de las observaciones en WITT[31] [HANS86] [HANS90] se realiza en forma de pares atributo-valor con la variante de que una observación puede tener más de un valor para un atributo, y se limita únicamente a atributos categóricos.

Adicional a la descripción habitual de las observaciones, el sistema añade a la representación las correlaciones entre pares de atributos en la forma de tablas de contingencia. Éstas guardan la coincidencia de aparición entre cada pareja de valores de todos los atributos utilizados. Puede verse un ejemplo en la figura 2.22.

Debido a esta representación, la correlación entre los atributos de los objetos y de las clases tendrá una gran importancia a la hora de la categorización.

Como metodología base para la contrucción de categorías, WITT utiliza una función de *teoría de la información* para contrastar las clases, tratando de maximizar la similaridad dentro de cada clase y minimizar la similaridad entre clases. A esta medida se la denomina *cohesión*. La expresión de esta función para una categoría es:

$$C_c = \frac{W_c}{O_c}$$

Donde W_c es la cohesión intra-clase (de los objetos de la clase) y O_c representa la cohesión media de la clase c con el resto de clases existentes. Se puede interpretar esta medida como el contraste entre la media de la distancia de los objetos en el interior de una clase respecto a la media de la distancia de esa clase con el resto. Esta distancia tiene en cuenta la correlación entre los atributos de los objetos, en contraste con la típica medida euclídea, que asume la independencia entre ellos.

[31] Su nombre se debe al filósofo Wittgenstein que estudió ampliamente la esencia de la categorización.

Textura	**Color**	**Sabor**
liso	verde	dulce
rugoso	naranja	dulce
rugoso	verde	amargo
liso	verde	amargo

Textura x Color

	verde	naranja
liso	2	0
rugoso	1	1

Textura x Sabor

	dulce	amargo
liso	1	1
rugoso	1	1

Color x Sabor

	dulce	amargo
verde	2	1
naranja	0	1

Figura 2.22: Representación de las observaciones en WITT

La cohesión intra-clase (W_c) es calculada como la media de las variancias de las coocurrencias de todos los posibles pares atributo-valor para una categoría:

$$W_c = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} D_{ij}}{N \cdot (N-1)/2}$$

donde N es el número de atributos y D_{ij} es la distribución de coocurrencias asociada a la tabla de contingencia de los atributos i y j, que queda definida como:

$$D_{ij} = \frac{\sum_{m=1}^{i_v}\sum_{n=1}^{j_v} f_{mn}\log(f_{mn})}{(\sum_{m=1}^{i_v}\sum_{n=1}^{j_v} f_{mn})(\log(\sum_{m=1}^{i_v}\sum_{n=1}^{j_v} f_{mn}))}$$

donde f_{mn} es la frecuencia con la que el valor m del atributo i y el valor n del atributo j coocurren, y i_v y j_v son el número de modalidades de los atributos i y j. El realizar este cálculo supone la suma de los valores de todas las tablas de contingencia almacenadas.

El cálculo del numerador de la función de *cohesión* (o_c) requiere medir la cohesión de una clase con el resto de clases. Para ello definimos la cohesión entre dos clases c y k como:

$$B_{ck} = \frac{1}{W_c + W_k - 2W_{c\cup k}}$$

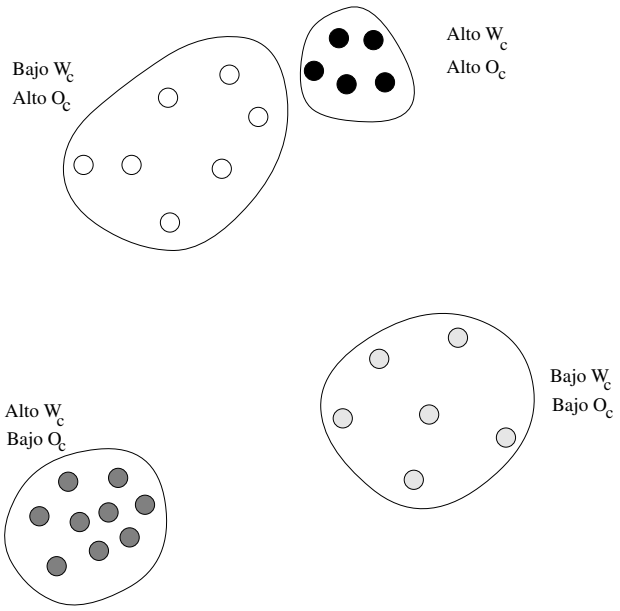


Figura 2.23: Relación entre las medidas W_c y O_c

Esta expresión mide la variancia de las coocurrencias entre la unión de las dos categorías respecto a la de las dos por separado. De esta manera podemos calcular la variancia total respecto a todas las categorías como:

$$O_c = \frac{\sum_{k=1(k\neq c)}^{L} B_{ck}}{L-1}$$

donde L es el número de categorías. En la figura 2.23 se ilustra gráficamente el efecto de estas dos medidas W_c y O_c respecto a la dispersión de los objetos y la distancia entre las clases. El algoritmo de categorización de WITT utilizará esta medida para guiar la formación de sus clases.

El algoritmo en sí consta de dos fases, una primera en la que genera un conjunto inicial de clases, y una fase de refinamiento en la que se aplican tres operadores diferentes sobre el conjunto inicial y las observaciones, añadiéndolas a las existentes, creando nuevas clases o fusionando clases entre sí.

La primera fase se basa únicamente en las distancias entre las observaciones, y no utiliza la función de evaluación de categorías que se ha explicado. Los pasos que sigue son los siguientes:

1. Se calcula la distancia entre todos los objetos y se guarda la menor de todas (D), calculando el valor P_1 como $F \times D$, donde F es un parámetro escogido por el usuario que indicará la generalidad de los grupos que se han de formar. Cuanto mayor sea el valor de F las agrupaciones iniciales que conseguiremos tendrán más clases y menos objetos.

2. Se selecciona el par de objetos más cercanos del conjunto de observaciones.
   (a) Si su distancia es mayor que el parámetro P_1 el proceso para.
   (b) Sino, se combinan los objetos para formar una clase y se reemplazan los objetos utilizados por ésta.
3. Se calcula la distancia entre este nuevo objeto y el resto, y se continúa en el paso 2.

El parámetro del usuario F es crucial en lo que respecta a las categorías iniciales que se obtendrán.

Una vez obtenido un conjunto de clases se pasa a una fase de refinamiento en la que entra en juego la medida de calidad de las categorías que hemos explicado. Para la ejecución de este algoritmo son necesarios dos parámetros de usuario adicionales que llamaremos P_2 y P_3 que se utilizarán para controlar la inclusión de un objeto a una clase existente, el primero, y para la creación de nuevas clases y la fusión de clases ya existentes, el segundo.

El algoritmo de refinamiento es el siguiente:

1. Se calcula el valor de la función de cohesión C entre cada combinación de pares de objetos no clasificados y de clases existentes.
2. Se selecciona el par instancia-categoría que tiene el mejor valor para esta función (C).
3. Si el valor de C es superior a P_2 se añade el objeto a la categoría y se vuelve al paso 1.
4. Si no existe tal par, se vuelve a invocar al algoritmo de generación de clases con los objetos que quedan por clasificar para crear nuevas clases.
   (a) Para cada nueva categoría creada c se calcula el valor de la función $W_{i \cup c}$ para cada categoría i ya existente, y si éste es siempre menor que el parámetro P_3 se la añade al conjunto de categorías.
   (b) Si al menos se ha añadido una categoría se pasa al paso 1.
5. Si no se ha consegido añadir nuevas categorías se calcula el valor de $W_{i \cup j}$ para todos los pares de categorías existentes y se selecciona el par con mejor valor.
6. Si este valor es mayor que P_3 se unen las dos categorías y se continúa en el paso 1, sino se para el algoritmo.

El que durante la agregación de objetos a categorías ningún par supere el parámetro P_2 se interpreta como que las categorías creadas hasta ese momento son inadecuadas y que se necesitan nuevas clases para complementar las existentes.

Esto se arregla utilizando de nuevo el algoritmo de creación de clases que se usó al principio. Para que una nueva categoría sea añadida al conjunto debe asegurarse que no ocupe el área de otra ya existente. Para ello se calcula el valor de cohesión intraclase de las posibles nuevas categorías con su unión con cada una de las categorías existentes ($W_{c \cup j}$). Si no supera el parámetro de usuario P_3, significa que no ocupa ninguno de los espacios cubiertos por las categorías ya existentes.

Si no se consigue ninguna categoría nueva significa que las existentes ya cubren todo el espacio de observaciones, por lo que se considera la unión entre ellas. Para ello calcula también la cohesión intraclase de todas las posibles uniones ($W_{i \cup j}$). Si alguno de estos valores supera el parámetro P_3 significa que son dos categorías que intersectan y son candidatas a unirse. Si no pasa es que ninguna de las categorías se superponen por lo que no tiene sentido formar más categorías.

Los parámetros P_2 y P_3 son los que determinan la forma de las clases que se obtendrán al final de la ejecución del algoritmo. Si el cociente entre P_2 y P_3 es grande, las categorías que se formen tenderán a presentar caracterizaciones con condiciones suficientes y necesarias. A medida que disminuye este cociente, esta restricción se relaja, obteniéndose categorías que muestran diferentes grados de *polimorfia*.

Estos parámetros también controlan la aplicación de los diferentes operadores utilizados para la construcción de las categorías (inserción, creación y unión). WITT se basa en el principio de que la reorganización (creación y unión de categorías) es un fenómeno poco frecuente en la categorización humana, por lo tanto hay que favorecer a los operadores más sencillos. Este principio está en la línea de las ideas expresadas por la psicología cognitiva, en constraste con otros métodos que se estudiarán más adelante que dan igual peso al uso de todos los operadores.

Es importante hacer notar que al contrario que en la metodología presentada por CLUSTER aquí no se determina *a priori* el número de clases que se han de crear, sino que surgen de manera natural de los datos que se clasifican, aunque dependen en gran medida de los parámetros P_2 y P_3.

AUTOCLASS

Los fundamentos teóricos de los que parte este sistema se basan ([DUDA73]) en la teoría bayesiana, aprovechándose del extenso trabajo que se ha realizado en teoría de la probabilidad. El uso de esta técnica tiene ventajas sobre los métodos anteriores:

- **El número de clases se determina automáticamente**. El número de clases surge a partir del uso del conocimiento a priori de la distribución de los atributos. La probabilidad *a priori* da preferencia a más clases más sencillas y la probabilidad *a posteriori* que se va obteniendo prefiere menos clases más complejas. El balance entre éstas da el número de clases más adecuado a los datos.

- **Los objetos no se asignan a clases de manera única**. La clasificación que se obtiene no es de clases disjuntas, hay una probabilidad de pertenencia asociada a cada objeto para cada clase. Esto está mas acorde con la existencia de asignaciones ambiguas de objetos a clases y la gradación de pertenencia a una clase constatada por la psicología cognitiva.

- **Se pueden mezclar datos descritos a la vez por atributos cuantitativos y cualitativos**. Muchos métodos sólo admiten uno de los dos tipos de atributos o hacen transformaciones de uno a otro, con la pérdida de información que eso supone.

Los autores de AUTOCLASS[CHEE88] marcan como ventaja adicional que **todos los atributos son valorados por igual**. Esta ventaja es bastante irreal desde el punto de vista de la clasificación humana, dado que las personas tienden a fijar su atención en los atributos más relevantes ya que no podemos manejar las relaciones entre una gran cantidad de atributos. Las evidencias de la psicología cognitiva son abrumadoras respecto a la selectividad en los atributos a la hora de decidir la asignación de una observación a una categoría. No obstante se marca como una ventaja sobre la clasificación humana el poder tomar en cuenta toda la información disponible.

El algoritmo de clasificación se basa en el teorema de Bayes para la combinación de probabilidades. Dado un conjunto de observaciones O y una hipótesis H, la probabilidad de que la hipótesis explique los datos $p(O|H)$ (probabilidad *a posteriori* de la hipótesis dados los datos) es proporcional a la probabilidad de observar los datos si la hipótesis fuera cierta $p(O|H)$ (la *verosimilitud* (*likelihood*) de los datos) por la probabilidad de la hipótesis independientemente de los datos $p(H)$ (su probabilidad *a priori*). La probabilidad *a priori* de los datos $p(O)$ es una constante que puede ser obviada, ya que para los cálculos que necesitamos hacer sólo nos interesa el valor relativo entre las diferentes probabilidades. Su fórmula habitual es:

$$p(H|O) = \frac{p(H) \cdot p(O|H)}{p(O)}$$

Para el propósito de la clasificación se toma como hipótesis H el número de clases y los descriptores que existen en las observaciones. Por lo tanto, el objetivo es encontrar una partición que maximize la probabilidad $p(H|O)$.

Las restricciones que se imponen para la aplicación del método son las siguientes:

- Los datos han de ser independientes entre sí, es decir, no deben proceder de una serie temporal.
- Las distribuciones de los atributos se han de poder aproximar por distribuciones normales de probabilidad.
- Los valores de los atributos para una misma observación han de ser independientes entre sí.

Estas suposiciones pueden ser bastante fuertes en algunos dominios, lo que restringe el ámbito de aplicabilidad del método.

La base teórica del algoritmo de clasificación, de manera sucinta, se describe a continuación.

Cada observación del conjunto debe pertenecer a alguna de las J posibles clases existentes por lo que posee una distribución de probabilidad para cada clase $p(x_i|x_i \in C_j, \overrightarrow{\theta_j})$ que da la distribución de probabilidad de los atributos de cada dato si pertenecieran a la clase j. La distribución de los atributos de una clase $\overrightarrow{\theta_j}$ se describe a partir de su media μ_j y variancia σ_j^2, bajo la suposición de que sigue una distribución normal.

A la probabilidad de que un objeto cualquiera pertenezca a la clase j se la denomina *probabilidad de clase* (π_j). La probabilidad de que un objeto pertenezca a un conjunto de clases es la suma de las probabilidades de que pertenezca a cada una de ellas por separado.

$$p(x_i|\overrightarrow{\theta}, \overrightarrow{\pi}, J) = \sum_{j=1}^{J} \pi_j \cdot p(x_i|x_i \in C_j, \overrightarrow{\theta_j})$$

Bajo la suposición de que las observaciones son independientes entre sí, la *verosimilitud* (*likelihood*) del conjunto total de datos será el producto de las probabilidades de cada objeto.

$$p(\overrightarrow{x}|\overrightarrow{\theta}, \overrightarrow{\pi}, J) = \prod_{i=1}^{I} p(x_i|\overrightarrow{\theta}, \overrightarrow{\pi}, J)$$

Para unos parámetros de clasificación dados se puede calcular la probabilidad de que un objeto i pertenezca a una clase j aplicando el teorema de Bayes como:

$$p(x_i \in C_j|x_i, \overrightarrow{\theta}, \overrightarrow{\pi}, J) = \frac{\pi_j \cdot p(x_i|x_i \in C_j, \overrightarrow{\theta_j})}{p(x_i|\overrightarrow{\theta}, \overrightarrow{\pi}, J)}$$

La clasificación que se obtiene asigna una probabilidad de pertenencia para cada objeto a cada una de las clases existentes.

El problema de clasificación se divide en dos partes, estimar los parámetros $\overrightarrow{\theta}$ y $\overrightarrow{\pi}$ y determinar el número de clases J óptimo.

Para ambos casos se sigue un proceso de búsqueda y optimización de las funciones de probabilidad a *posteriori* que maximizan la probabilidad de pertenencia de los objetos a las clases, utilizando heurísticas que evitan que en el proceso de búsqueda se caiga en máximos locales. En el caso particular del número de clases se puede comenzar la búsqueda con un número de clases superior al esperado. Si las probabilidades de pertenencia de objetos a las clases π_j no son significativas se puede reducir el número, sino se intenta con un número de clases mayor. Para conocer más detalles sobre cómo se estiman los diferentes parámetros se puede consultar [CHEE88].

Lo que diferencia a este método de los métodos estadísticos es la elección de las probabilidades a priori de los párametros que hay que estimar sobre las clases. Se supone más lógico asignar distribuciones sencillas a falta de mayor información. Además ésto permite poder determinar el número de clases necesarias.

2.5.6 Formación de conceptos

La formación de conceptos (*concept formation*) busca el mismo objetivo que la agrupación conceptual, obtener una clasificación de un conjunto de observaciones y una caracterización de las clases obtenidas que permita identificar los diferentes grupos. Estas técnicas además ponen especial énfasis en la construcción de una jerarquía que permita relacionar los conceptos.

La diferencia fundamental con los métodos anteriores está en el planteamiento del aprendizaje como una tarea incremental. Se pretende simular el comportamiento de un agente que va adquiriendo su conocimiento a partir de ir acumulando la experiencia de las observaciones que le van llegando.

Ésta es la causa por la que todos los métodos de formación de conceptos plantean su estrategia de manera que cada nueva observación se sitúa en la jerarquía de conceptos que se va construyendo de forma que complemente el conocimiento que se tenía hasta ese momento. Las modificaciones que son necesarias en la estructura jerárquica se guían a través de funciones que optimizan ciertos criterios sobre lo que se va aprendiendo.

Los algoritmos de formación de conceptos, por lo tanto, realizan una búsqueda en un espacio de jerarquías de conceptos con un método de *ascenso* (*hill-climbing*). La diferencia fundamental con los métodos clásicos de búsqueda por ascenso[32] es que, al contrario que en éstos, el objetivo al que se debe llegar mediante la función heurística de evaluación no se mantiene constante. La adquisición de nuevo conocimiento va cambiando el entorno global y por lo tanto lo que se debe aprender. Otra diferencia es que estos métodos limitan su memoria a una única solución en curso, no tratan diferentes alternativas desde las que llegar a la solución.

El precio que se ha de pagar por la incrementalidad es la sensibilidad de todos estos métodos al orden de entrada de las observaciones [FISH92] y al *ruido* que pueden presentar sus descripciones. Para reducir estos efectos se ha propuesto proveer de operadores capaces de modificar la jerarquía de conceptos en el momento en que las nuevas observaciones permitan detectar errores en lo aprendido. Esto permitiría poder hacer una búsqueda bidireccional produciendo el efecto de una *vuelta atrás* (*Backtracking*) pero sin el coste en espacio que esto requeriría. De todas formas el efecto de estos operadores aún es limitado.

La evolución de los métodos de formación de conceptos ha ido dejando un conjunto de métodos que intentan plasmar todas estas ideas. En los siguientes apartados se describirán los sistemas que más han destacado.

EPAM

EPAM [FEIG61][FEIG84] se puede considerar como uno de los primeros modelos de formación de conceptos. Pretendía servir como modelo del aprendizaje humano en las tareas de memorización verbal, utilizándose para explicar una serie de fenómenos en el aprendizaje observados por los psicólogos.

La representación de conocimiento en EPAM se realiza a partir de pares atributo-valor donde cada componente puede a su vez tener una lista de descriptores, por lo que se permite la descripción de objetos compuestos por partes.

El algoritmo de aprendizaje es capaz de construir a partir de las observaciones un árbol de decisión al estilo de ID3[QUIN86] donde cada nodo no terminal de la jerarquía corresponde a un test sobre un atributo, y de él parten ramas que corresponden a diferentes valores del atributo. A diferencia de los arboles de decisión no se asume que se conozcan todos los valores de un atributo, por ello en todos los nodos no terminales hay una rama especial etiquetada como *otros* para los nuevos valores que puedan aparecer. En los nodos terminales se guarda

[32] Ver "Inteligencia Artificial" (POLITEXT nº 17) Capítulo 4.

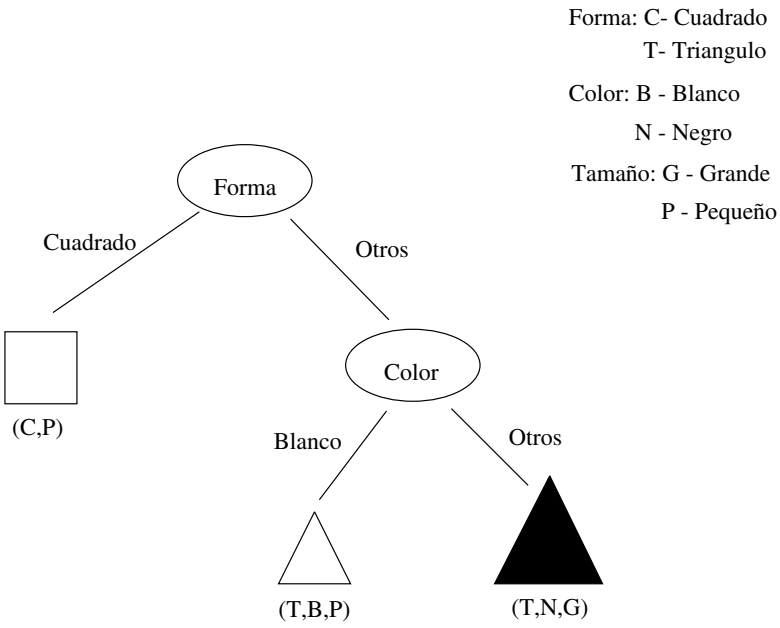


Figura 2.24: Jerarquía construída por EPAM

una lista con los valores de los atributos que se espera que tomen las observaciones clasificadas en ese punto. En la figura 2.24 se puede ver un ejemplo de una jerarquía de conceptos.

El algoritmo de aprendizaje funciona comprobando los diferentes tests que se encuentran en los nodos de la jerarquía comenzando por la raíz hasta llegar a un nodo terminal. El proceso es el siguiente:

- Si el nodo sobre el que se está es no terminal, se comprueba el valor de la observación sobre el atributo que marca el nodo.
  - Si existe una rama en el nodo con el valor del objeto, se aplica el proceso recursivamente desde el nodo al que apunta esa rama.
  - Si no hay tal rama se sigue por la etiquetada como *otros* y se aplica recursivamente el proceso desde ese nodo.
- Si el nodo sobre el que se está es terminal:
  - Si la observación coincide con el concepto almacenado en el nodo se realiza un proceso de *familiarización*. Este proceso consiste en añadir al nodo uno de los atributos que posee la observación y no posee éste.
  - Si alguno de los atributos de la observación no coincide con el nodo se realiza un proceso de *discriminación*. La diferencia puede deberse a dos causas:
    1. Puede que la observación haya sido clasificada pasando por alguna rama etiquetada como *otros*. Si es así se detecta la rama y se añade una nueva que

contemple el valor de la observación, aumentando de esta manera la anchura del árbol.

2. Si la diferencia está en el nodo terminal y no en los atributos del camino que llevaron hasta él, se crea un nuevo nodo decisión que contemple los dos valores diferentes para el atributo en cuestión. Se construye una rama para el valor del nodo y otra para la observación y se crea un nodo terminal para cada uno. De esta manera se aumenta la profundidad del árbol.

En la figura 2.25 se puede ver un ejemplo de este proceso. Se introducen tres instancias sobre la jerarquía de la figura 2.24, contemplando los tres diferentes casos que pueden aparecer en el algoritmo. Las instancias que se introducen son: Un cuadrado negro y pequeño, que da lugar a una familiarización, un cuadrado grande y blanco que da lugar a una discriminación y un círculo blanco y pequeño que da lugar al segundo tipo de discriminación que puede darse.

La importancia de este método radica en la influencia que tuvo sobre los métodos que se desarrollaron a continuación. Fue el primero en introducir el uso de una jerarquía de conceptos y en integrar en un algoritmo incremental las tareas de clasificación y aprendizaje. Además, introduce los operadores de *familiarización* y *discriminación* para el proceso de la construcción de la jerarquía de conceptos.

UNIMEM y CYRUS

UNIMEM [LEBO87] y CYRUS [KOLO83] son la siguiente generación de sistemas de formación de conceptos. Adoptan las ideas de EPAM en cuanto a construcción de una jerarquía y en la introducción incremental de las instancias. La principal diferencia es que en ambos casos cada nivel de la jerarquía se distingue del siguiente a través de un conjunto de atributos a diferencia de EPAM, en el que cada nivel se diferenciaba sólo por un atributo.

Estos sistemas estaban ideados para crear índices para la recuperación eficiente de información y se encuadran dentro del área de *memorias basadas en generalización* (*generalization-based memory*). La representación del conocimiento que utilizan es la habitual de pares atributo-valor, pero a diferencia con el método anterior se admiten valores tanto categóricos como lineales.

La jerarquía que se construye tiene asociada una descripción en cada nodo, en lugar de sólo en los nodos terminales como en EPAM. Esta descripción consiste en una lista de pares atributo-valor donde cada par tiene asociado un número[33] que representa la *confianza* (*confidence*) en el atributo. Además de este valor, se incluye para cada atributo el número de veces que aparece en los nodos de la jerarquía, utilizando este valor como una medida de la *predecibilidad* (*predictiveness*) del atributo. En la figura 2.26 se puede ver un ejemplo de jerarquía construída por estos métodos.

El algoritmo de UNIMEM clasifica las instancias partiendo de la raíz de la jerarquía comparando con cada uno de los nodos no terminales hasta conseguir integrarlas. El método es el siguiente:

[33] En UNIMEM éste es solamente un número entero calculado de forma *ad hoc*, en CYRUS representa una probabilidad.

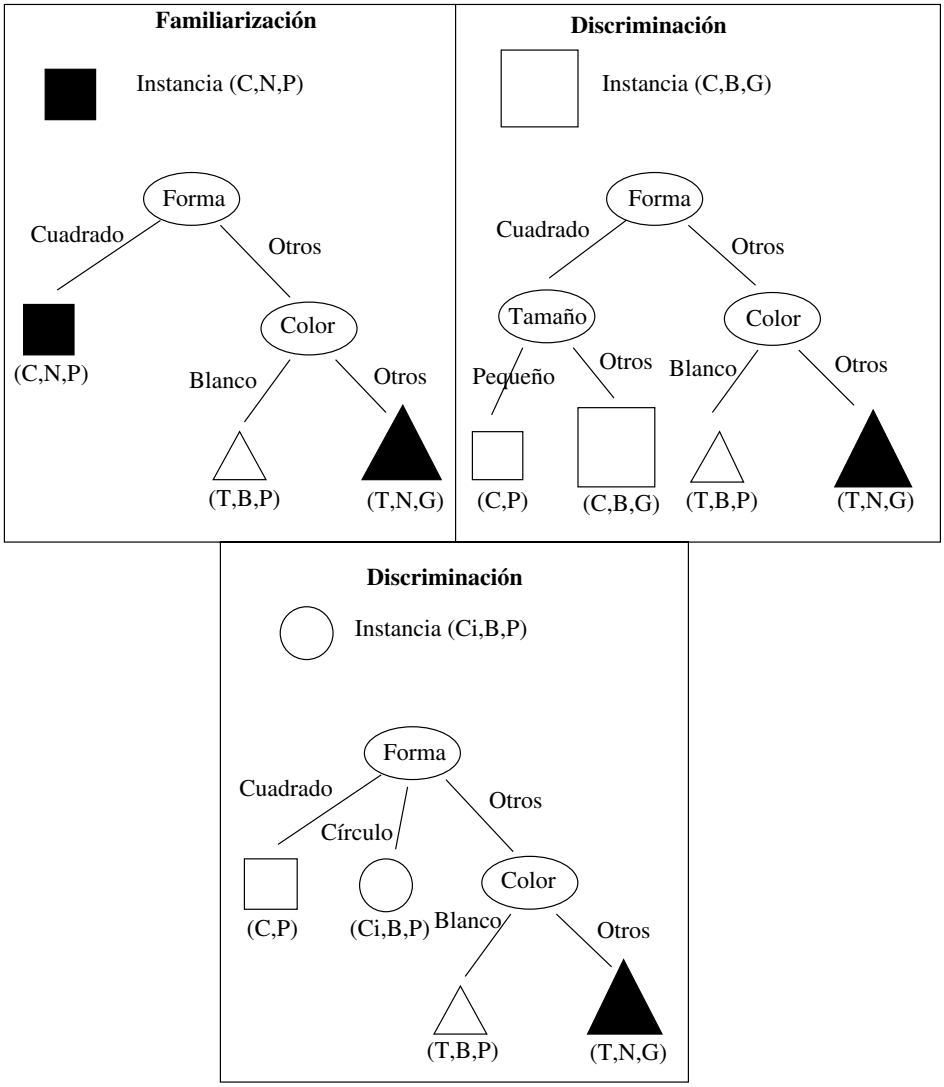


Figura 2.25: Clasificación de instancias con el algoritmo de EPAM

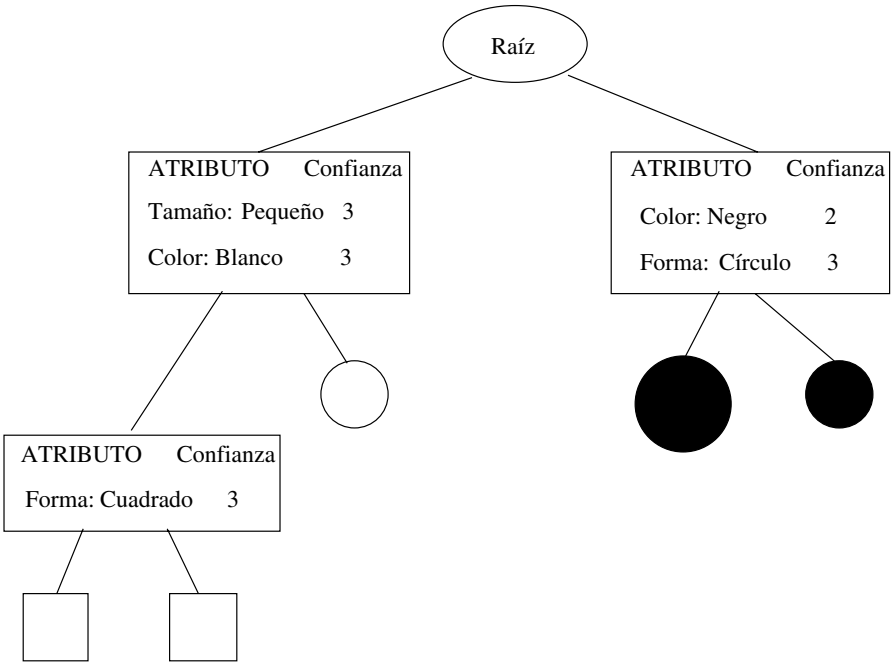


Figura 2.26: Jerarquía construída por UNIMEM

- Dado un nodo se comparan los atributos de éste con los de la instancia. Para calcular el parecido entre la instancia y los nodos se utiliza una función de distancia, y se usa un parámetro determinado por el usuario para poner el límite en el número de atributos en que éstos deben ser parecidos. Con este método es posible que se clasifique la instancia en varias ramas diferentes.
- Tanto si se coincide con los descendientes como si no, los valores de *confianza* y *predecibilidad* son modificados teniendo en cuenta la instancia.
  - Si existen descendientes que coinciden con la instancia se sigue por el camino de los nodos que más se parezcan y que coincidan con los valores de la instancia.
  - Si ningún descendiente llega al límite de similaridad se examinan las instancias almacenadas bajo ese nodo.
    - Si alguna de éstas comparte suficientes valores con la nueva instancia, dependiendo de otro parámetro de usuario, se crea un nuevo nodo generalizando las instancias parecidas y se almacenan estas instancias bajo el nuevo nodo. Cuando esto pasa, el algoritmo incrementa la *predecibilidad* de los atributos que aparecen en este nuevo nodo.
    - Si no hay ninguna instancia suficientemente similar, se almacena la nueva instancia bajo el nodo en curso.

Con este algoritmo, una instancia puede ser clasificada en varias categorías, por lo que se produce un *solapamiento* (*overlapping*) entre las distintas clases. Esto puede ser una ventaja

para algunos dominios, permite mantener a la vez varias categorías que pueden ser útiles en cierto momento y que con la incorporación de nuevas instancias pueden ir desapareciendo dejando sólo las mejores, ayudando así en la búsqueda dentro del espacio de posibles jerarquías.

Añadidos al algoritmo, hay ciertos parámetros especificados por el usuario que ayudan a dirigir la búsqueda modificando los atributos que hay en los nodos. El valor de *confianza* de los atributos es aumentado o disminuído dependiendo de si las instancias coinciden o no con cada uno de ellos. Si éste supera cierto parámetro, el atributo correspondiente es fijado, y nuevas instancias no lo modificarán, dando a su valor como seguro. Si éste es menor que otro parámetro, se le hace desaparecer de la descripción del nodo. Si la *predecibilidad* de un atributo de un nodo se hace muy alta se elimina el atributo de los descendientes de ese nodo, haciendo así que se reduzca la frecuencia con que un concepto es usado para guiar la clasificación. Por último, si el número de atributos que hay en la descripción de un nodo es menor que otro parámetro, éste es eliminado, ya que será un nodo demasiado general y poco útil para clasificar instancias.

Las principales ventajas que presentan estos métodos respecto a su predecesor (EPAM) son varias. La primera es el incorporar una descripción a cada uno de los nodos de la jerarquía en lugar de sólo a los nodos terminales. Otra más es el introducir más de un atributo como elemento de decisión entre cada uno de los niveles de ésta. Se introducen también medidas de cómo cada uno de los atributos es útil para clasificar las instancias, modificando dinámicamente el peso que tiene cada uno de ellos. A pesar de estos avances, aún se presentan muchos problemas, como la gran cantidad de parámetros que debe usar el usuario para guiar la aplicación de los diferentes operadores que se usan para modificar la jerarquía y guiar el aprendizaje.

COBWEB

COBWEB [FISH87] es fruto de las ideas de UNIMEM y CYRUS y de las ideas de la psicología cognitiva acerca del *nivel básico* (*basic level*) [SMIT81] de categorización.

Ciertos estudios psicológicos sobre categorización humana han llegado a descubrir que a la hora de formar conceptos se prefiere un nivel de generalización respecto a otros. En este nivel es más fácil la predicción de las características generales de los miembros y la labor de identificar instancias de diferentes categorías. Se han hecho experimentos en los que, por ejemplo, dada una instancia particular de una paloma, la gente tarda menos en verificar que es un *pájaro* de lo que tarda en verificar que es un *animal* o una *paloma*.

De entre las múltiples medidas que se han desarrollado para descubrir el *nivel básico* [MURP82] [JONE83], COBWEB ha adoptado la medida desarrollada por [GLUC85] denominada *utilidad de categoría* (*category utility* (CU)) para desarrollar su algoritmo. Esta medida es utilizada para guiar el proceso de aprendizaje ya que debería ser maximizada por las categorías que se encuentran en el nivel básico. Esta función da mayor valor a las clases que presentan una alta similaridad entre sus miembros y una baja similaridad con el resto de clases. Es una función que mantiene un balance entre *predecibilidad* (*predictiveness*) y *previsibilidad* (*predictability*) (lo que se denominaba en UNIMEM y CYRUS *confianza*).

La *predecibilidad* es la probabilidad de que una observación pertenezca a una clase dado el valor de un atributo, la *previsibilidad* es la probabilidad de que una observación tenga un valor en un atributo dado que pertenece a cierta clase. Los atributos *predecibles* son aquellos que

son prácticamente exclusivos de una clase y por lo tanto diferenciadores. Es interesante que el número de estos atributos en una clase sea alto ya que maximiza la diferencia entre clases. Los atributos *previsibles* son los que comparten muchos miembros de una clase. El favorecer la existencia de muchos atributos previsibles maximiza la similaridad entre los miembros de una clase.

Dado que los atributos no siempre pueden ser *predecibles* y *previsibles* a la vez la *utilidad de categoría* intenta maximizar el equilibrio entre las dos propiedades. Para medir la *predecibilidad* de un conjunto de clases se usa la fórmula:

$$\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k)^2$$

donde K es el número de clases, I el de atributos y J el de valores. $P(C_k)$ es la probabilidad de que exista la clase C_k y $P(A_i = V_{ij}|C_k)$ es la probabilidad condicional de el valor V_{ij} dada la pertenencia a la clase. Para medir la *previsibilidad* se usa la fórmula:

$$\sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2$$

donde $P(A_i = V_{ij})$ es la probabilidad de un valor particular en el conjunto de datos. La *utilidad de categoría* (CU) para un conjunto de clases $\{C_1, C_2, \ldots C_K\}$ se calcula como la diferencia de estos dos valores normalizada por el número de clases:

$$CU(\{C_1, C_2, \ldots C_K\}) = \frac{\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k)^2 - \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2}{K}$$

La representación del conocimiento utilizada es la típica de atributo-valor, sólo que no se admiten más que atributos categóricos. El resultado del algoritmo es una jerarquía en la que en cada nodo se guardan los valores de las probabilidades de cada uno de los valores de los atributos que se clasifican bajo ese nodo y la probabilidad de la clase que representa el nodo. En la figura 2.27 se puede ver un ejemplo de esta jerarquía.

El algoritmo de COBWEB va incluyendo instancias en la jerarquía descendiendo a través del árbol guiándose por la medida de *utilidad de categoría* para decidir el descendiente por el que ha de continuar o el operador que debe aplicar al árbol para incorporar el nuevo conocimiento. Se pueden aplicar cuatro operadores diferentes: Incorporar a una clase (*Incorporate*), crear una nueva clase (*New class*), unir dos clases existentes (*Merge*) o dividir una clase en sus descendientes (*Split*). El algoritmo es el siguiente:

- Actualizar las probabilidades del nodo en curso según los valores de la observación
- Si el nodo es terminal, el resultado es incorporar el nodo modificado, finalizando el algoritmo.

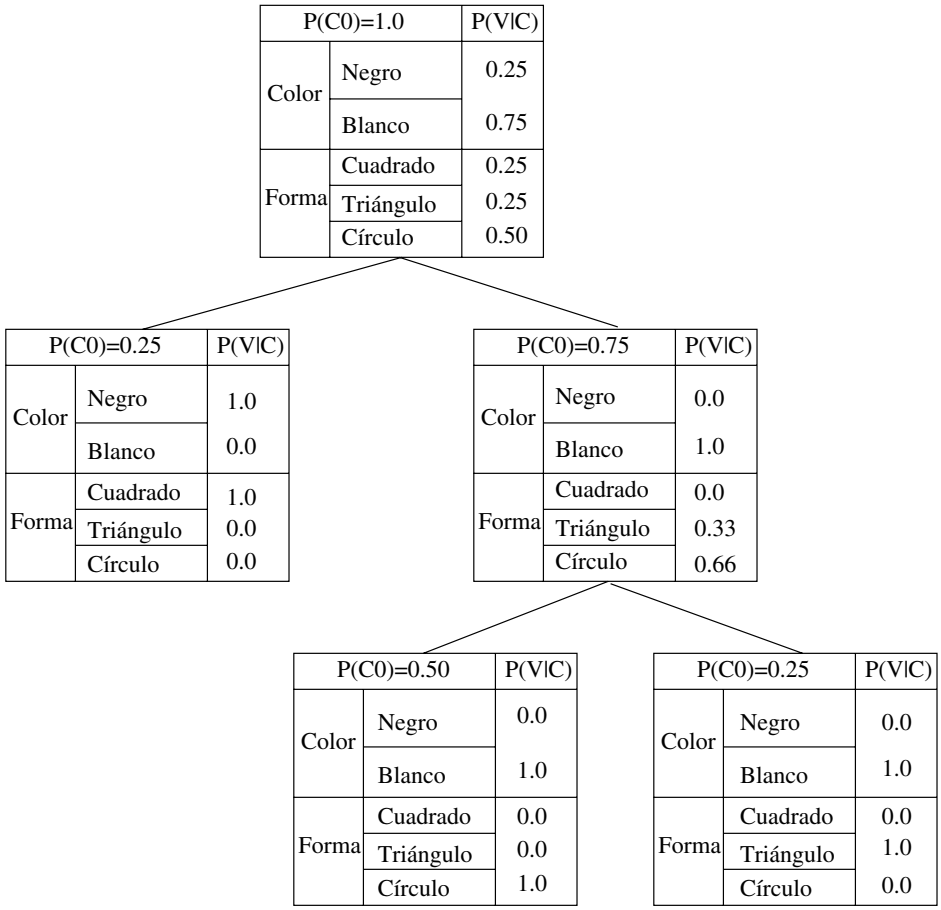


Figura 2.27: Jerarquía construída por COBWEB

- Si el nodo no es terminal se evalúan las siguientes posibilidades según la función CU, se escoge la mejor y se llama recursivamente a la función con el nodo en el que se haya decidido incorporar la observación.

  1. Se clasifica la observación en cada descendiente del nodo en curso y se identifica el que maximize la función CU. Ése sería el nodo en el que se incorporaría la observación. (*Incorporate*)
  2. Se calcula la función CU añadiendo una nueva clase que contenga únicamente la observación. (*New class*)
  3. Se une el mejor par de clases y se incorpora la observación a esta clase. Se escogería esta opción si se mejora la función CU del nodo en curso. (*Merge*)
  4. Se particiona la mejor clase y se añaden sus descendientes, calculando el resultado de la función CU al añadir la observación a cada una de las clases incorporadas, dejándola en la mejor. (*Split*)

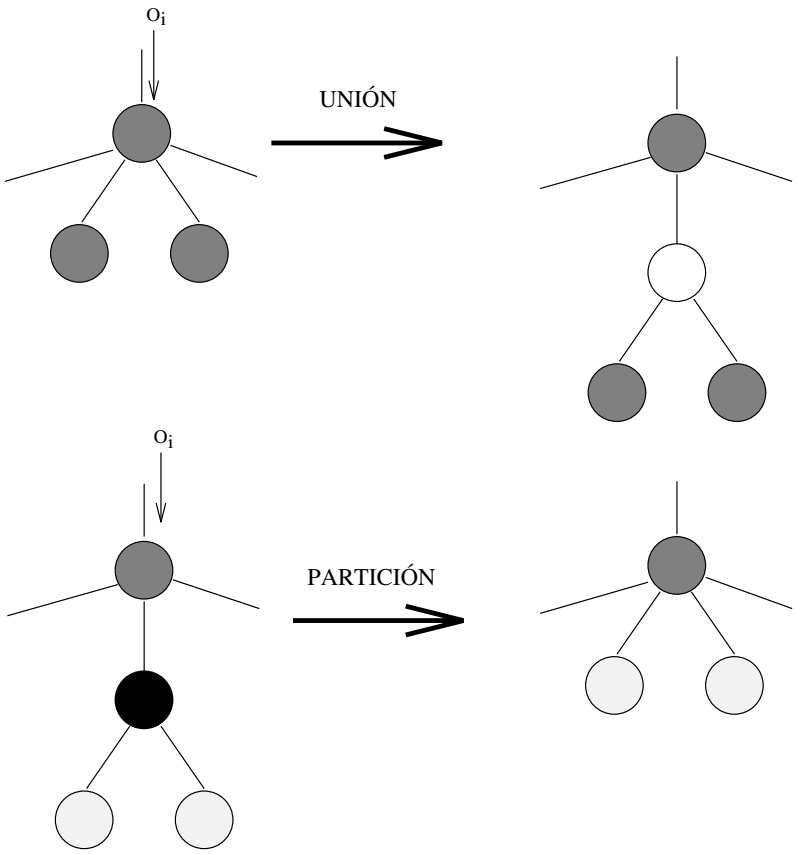


Figura 2.28: Efecto de los operadores de *unión* y *partición*

Los operadores de *unión* y *partición* se incorporan para evitar la sensibilidad al orden de entrada de los objetos, permitiendo la reorganización del arbol en el momento en que se detecten errores en la jerarquía formada. Éstos permiten un movimiento bidireccional en la exploración en el posible espacio de jerarquías. En la figura 2.28 se puede observar el efecto de estos dos operadores.

El principales avance que incorpora COBWEB en su metodología es la utilización de una descripción probabilística de las clases, además de fundamentar la creación de su jerarquía en una medida que está orientada hacia la búsqueda del nivel básico de categorización. Además, define los cuatro operadores necesarios para la construcción de la jerarquía y fundamenta la elección de cada uno de ellos en la medida que utiliza.

A partir de COBWEB han surgido otros algoritmos que complementan y amplían su capacidad. CLASSIT [GENA89] cambia la representación del conocimiento dando la posibilidad de utilizar atributos cuantitativos únicamente[34]. Para ello se tuvo que modificar la función

[34] Esto es debido a que CLASSIT fue diseñado para ser usado en el dominio del reconocimiento de imágenes.

de *utilidad de categoría*, transformando la probabilidad del valor de un atributo. Se toma como suposición que la distribución de los valores sigue una *distribución normal* por lo que la transformación es la siguiente:

$$\sum_{j=1}^{J} P(A_i = V_{ij})^2 \Leftrightarrow \int_{-\infty}^{\infty} \frac{1}{\sigma^2 2\pi} e^{-(\frac{x-\mu}{\sigma})^2} dx = \frac{1}{\sigma} \frac{1}{2\sqrt{\pi}}$$

El término $\frac{1}{2\sqrt{\pi}}$ puede ser obviado al ser una constante, por lo que la función quedaría como:

$$CU(\{C_1, C_2, \ldots C_K\}) = \frac{\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \frac{1}{\sigma_{ik}} - \sum_{i=1}^{I} \frac{1}{\sigma_{ip}}}{K}$$

donde I es el número de atributos, K es el número de clases, σ_{ik} es la desviación estándar de un atributo en una clase y σ_{ip} es la desviación estándar de un atributo en el nodo raíz de las clases.

El algoritmo usado por CLASSIT es idéntico al de COBWEB, pero ahora los nodos que se generan en el árbol guardan la media y la desviación estándar de los atributos. Además, a la estrategia normal se le añaden dos parámetros de usuario que controlan la búsqueda y mejoran el rendimiento respecto a COBWEB, uno denominado *corte* (*cutoff*) y otro denominado *agudeza* (*acuity*).

El parámetro *corte* evita que una observación sea clasificada en un nivel del árbol demasiado profundo. En un punto del algoritmo, y dependiendo del valor del parámetro, se puede decidir que ya no merece la pena seguir adelante para asignar la observación, dejándola en el nodo actual. De esta manera se controla también la profundidad del árbol que se genera. Esto hace que se descarte la información específica de la instancia. Con este parámetro se consiguen dos efectos. El primero es disminuir el problema de *sobredescripción* (*overfitting*) de los datos que se puede presentar en dominios donde las instancias presentan *ruido* con la consiguiente ganancia de rendimiento. Por otra parte, el guardar toda la información de las instancias en el árbol puede llevar a tener que tratar estructuras de datos demasiado grandes para hacer aplicable la metodología a dominios reales.

El parámetro *agudeza* se utiliza para evitar el problema de los nodos en los que hay una única instancia, y que hacen que el inverso de la desviación estándar de los atributos sea infinito. Con este valor se indica a partir de cuando se debe tomar en cuenta el valor de la desviación estándar para ser tenida en cuenta. Este factor se puede tomar como el factor umbral de reacción que se utiliza en psicología. Con el valor de este parámetro se consigue controlar el factor de ramificación del árbol al afectar a la función de evaluación y por lo tanto se controla la anchura del árbol.

Otros ejemplos de algoritmos inspirados en COBWEB aplicados a otros dominios son LABYRINTH [THOM91] aplicado al aprendizaje de objetos estructurados (formados por partes) al estilo de los algoritmos supervisados que se han visto en este capítulo y BRIDGER, una variante específica para problemas de diseño.

LA promera herramienta que combina los dos tipos de atributos es COBWEB/3 [THOM93].

2.6 Resumen

El aprendizaje inductivo se ocupa de todos aquellos procesos en los que se adquiere nuevo conocimiento después de realizar inferencia inductiva sobre los datos de entrada. La visión clásica de este tipo de aprendizaje es la búsqueda en un espacio de estados ([**MITC82**]), que se va recorriendo con la ayuda de operadores de especialización y generalización.

Los dos tipos básicos de aprendizaje inductivo son la **adquisición de conceptos** (o aprendizaje a partir de ejemplos) y la **generalización descriptiva** (o aprendizaje a partir de la observación).

Dentro de la adquisición de conceptos, en este capítulo se trata el aprendizaje de descripciones a partir de ejemplos. Se han detallado los métodos más clásicos, como son los de Winston ([**WINS70**]), Hayes-Roth y McDermott ([**HAYE77**]), Vere ([**VERE75**]) y Michalski y Dietterich ([**DIET81**]). Todos estos métodos inducen la descripción de un concepto a partir de la presentación de ejemplos positivos (y a veces también negativos) de los mismos. Se cierra este tema con la presentación de un algoritmo que hace una búsqueda bidireccional guiada por los ejemplos positivos y negativos, el *espacio de versiones* de Mitchell ([**MITC82**]).

A continuación se explican los **árboles de decisión**, estructuras que modelizan los procesos de decisión involucrados en tareas de decisión. Se muestran varios algoritmos clásicos para construir estos árboles, como son el ID3, el ID3 normalizado (ambos propuestos por Quinlan) y el RLM (de Ramón López de Mántaras). dentro de los algoritmos incrementales se encuentran el ID4(R) (de Schlimmer y Fisher), el ID5(R) (de Utgoff). Este tema está muy relacionado con el estudio de la **relevancia de los atributos**, también tratado en este capítulo.

En el campo de la generalización descriptiva, se explica extensamente el tema de la **formación de conceptos**. Se muestran las aproximaciones computacionales al aprendizaje por observación, y se explica cómo se aplican dentro del aprendizaje automático. Se describen sistemas como CLUSTER, WITT, AUTOCLASS, EPAM, UNIMEM, CYRUS y COBWEB.

2.7 Ejercicios

1. ¿Cúales son, según lo explicado en 2.2.1, las dos fases más importantes en el aprendizaje? Identifique y explique en que consisten estas fases en el algoritmo de Winston "**Learning from structural descriptions**".

2. Suponga que dispone del algoritmo **W** de aprendizaje de Winston.

   (a) Diseñar un frame para representar (con no menos de 5 atributos) objetos que pertenecen al mundo de los bloques. Definir los prototipos de algunas de las clases más usuales.

   (b) Diseñar un conjunto de entrenamiento para aprender el concepto $Torre$[35]. El número mínimo de objetos por ejemplo es tres. Los elementos deben pertenecer al mundo de los bloques. Asuma que todos los ejemplos están colocados sobre una mesa M. Describa las relaciones entre objetos.

[35] El último objeto es una pirámide

(c) Aprende el concepto realizando las generalizaciones y especializaciones oportunas. Explicar cada paso.

3. Ilustre con ejemplos significativos el concepto de cuasiejemplo *near-miss*.

   (a) Cuando se utiliza para especializar un concepto, y

   (b) Cuando se emplea para generalizar.

4. Ilustre a partir de ejemplos distintos a los empleados en el ejercicio anterior las heurísticas *forbid-link*, *require-link* y, finalmente, *climb-tree*.

5. ¿Explique la fase de generalización en el algoritmo de eliminación de candidatos en el espacio de versiones.

6. Dada la base de ejemplos Δ:

objeto	gris	mamífero	grande	vegetariano	salvaje	tipo
elefante	si	si	si	si	si	+
elefante	si	si	si	no	si	+
ratón	si	si	no	no	si	−
jirafa	no	si	si	si	si	−
dinosaurio	si	no	si	no	si	−
elefante	si	si	si	si	no	+

   (a) Generar una representación tipo frame.

   (b) Generar una representación en fórmulas lógicas de primer orden.

   (c) Obtenga los conjuntos **E** y **G** de fórmulas más específicas que se pueden obtener de Δ usando el espacio de versiones. Dibuje la parte útil del espacio de versiones.

   (d) Intente, si es posible, generalizar las fórmulas conjuntivas obtenidas en (b).

7. Dada la base de ejemplos Δ:

objeto	país	marca	color	década	tipo	ejemplo
000	Alemania	Audi	Azul	1980	Deportivo	+
001	Japón	Honda	Azul	1980	Económico	−
002	Japón	Toyota	Rojo	1970	Deportivo	+
003	Japón	Azul	Toyota	1990	Económico	−
004	USA	Chrysler	Rojo	1980	Económico	−
005	Japón	Honda	Blanco	1980	Económico	−
006	GB	Rover	Rojo	1990	Deportivo	+
007	GB	Rover	Rojo	1990	Familiar	−
008	USA	Ford	Rojo	1980	Deportivo	+

   (a) Use los ejemplos para aprender el conjunto más general y el más específico posible. Dibuje el espacio útil de versiones.

(b) Se puede aprender el concepto `coche económico`, qué es lo que hay que cambiar.

8. Si se usa el ejemplo definido en la figura 2.16 y se cambia el orden de los ejemplos colocando los tres positivos primero y luego los dos negativos.

   (a) ¿Cómo cambia el espacio de versiones?

   (b) Discuta el resultado final de la aplicación

9. Discuta las diferencias metodológicas entre el espacio de versiones y el aprendizaje *a la Winston*.

10. Implementar en LISP las medidas heurísticas G, G_N y RLM, y probarlas con el ejemplo desarrollado en el texto.

11. Utilizando el trabajo del apartado anterior, desarrollar en LISP un algoritmo de creación de árboles de decisión estilo ID3.

12. Demostrar, para el método RLM, las igualdades

    (a) $IV(X, A) = I(Part(X, A))$

    (b) $E(X, A) = I(P_{\mathcal{C}}(X)/Part(X, A))$

13. Un problema clásico es el de detección de paridad par de tres variables booleanas. Se necesita un árbol de tres niveles, siendo cualquiera de las tres variables igualmente buena (o mala) como raíz. Encontrar un orden fijo y repetido de presentación de las ocho instancias posibles tal que haga que el concepto no sea aprendible utilizando ID4.

Capítulo 3 Analogía

"Si el Cielo y la Tierra duran desde siempre
es porque no viven para sí mismos.
Eso es lo que los hace durar eternamente.
Por eso el Sabio excluye su persona
y siempre se halla en el primer lugar.
Se despoja de sí mismo
y por eso permanece.
Porque no busca su provecho
es que logra su provecho."

"Tao Te King", Lao Tse

3.1 Introducción

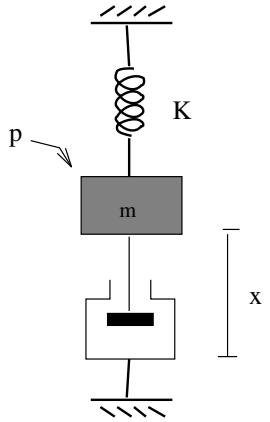
La analogía es una estrategia de pensamiento que permite explotar la experiencia acumulada (conocimiento de respaldo) al resolver problemas para intentar tratar problemas actuales. El proceso de analogía está basado en la siguiente suposición:

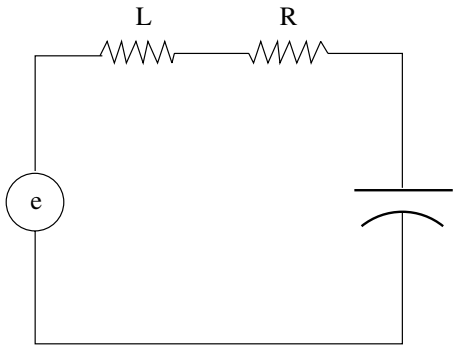
Si dos situaciones son similares *en algún aspecto entonces pueden serlo en otro.*

La utilidad de tal estrategia es evidente y de hecho se utiliza corrientemente en la vida diaria. Un ejemplo puede extraerse de los libros de física en los cuales aparece el esquema mostrado en la figura 3.1.

En este caso la base de la analogía es incuestionable, ya que si alguien conoce bien el funcionamiento de un circuito RLC, le será muy fácil de entender el de las leyes globales del comportamiento del sistema de fuerzas pues, como muestran las ecuaciones diferenciales que los describen, ambos sistemas son análogos. En este caso la analogía ha servido para trasladar conocimiento de un dominio bien conocido a otro.

Sin embargo, existen muchos ejemplos de *falsas* analogías; quizá una de los más conocidas es la que se empleó, en la antigüedad, para tratar de construir objetos que permitiesen volar al intentar imitar el vuelo de los pájaros. Los aviones tienen una forma semejante a la de las





$$m\frac{\partial^2 x}{\partial t^2} + f\frac{\partial x}{\partial t} + Kx = p \qquad ; \qquad e = L\frac{\partial^2 q}{\partial t^2} + R\frac{\partial q}{\partial t} + \frac{1}{C}q$$

$$\begin{aligned} Fuerza\ p &= e\ Tensión \\ Masa\ m &= L\ Inductancia \\ Coef.\ Elasticidad\ f &= R\ Resistencia \\ Desplazamiento\ x &= q\ Carga \end{aligned}$$

Figura 3.1: Analogía mecánica de un circuito *RLC*

aves pero no mueven las alas, como se intentó[1] durante mucho tiempo.

En el presente capítulo se presentarán algunos de los modelos que han sido definidos en el campo de la Inteligencia Artificial para desarrollar sistemas capaces de llevar a cabo un razonamiento (y, posteriormente, un aprendizaje) basado en analogías. En sección 3.2 se definen los conceptos básicos del razonamiento analógico; en la sección 3.3 se presenta el **modelo unificado de analogía**, definido por Kedar-Cabelli, que pretende servir de marco común a todos los sistemas analógicos; en la sección 3.4 se presenta el modelo de **inferencia analógica útil** (implementado en el sistema `NLAG`), haciendo especial énfasis en sus aspectos más teóricos; las sección 3.5 introduce el esquema de **analogía transformacional**, definido por Carbonell, el cual da paso, en la sección 3.6, a su extensión natural conocida con el nombre de **analogía derivacional**.

3.2 Definiciones previas

En esta sección se definirán algunos conceptos básicos para comprender cómo funcionan los procesos analógicos tanto de **razonamiento** como de **aprendizaje**.

En el vocabulario del razonamiento analógico, los dos conceptos más importantes que existen son:

[1] La historia nos revela que sólo Dédalo y su hijo Ícaro lograron volar con alas móviles.

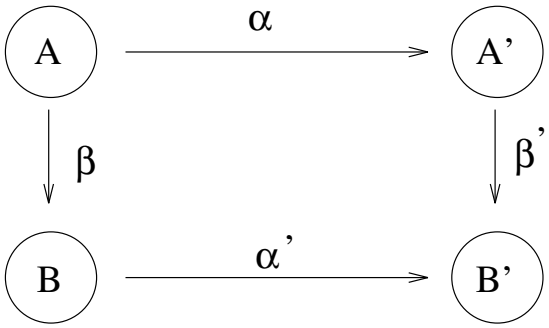


Figura 3.2: Un esquema general de razonamiento analógico

Problema base (o base): Es el referente de la analogía, es decir, el problema, ya resuelto en el pasado, tal que su solución servirá de base para resolver el nuevo problema. En caso de hacer analogía entre dominio diferentes, se llama **dominio base** a aquél en el cual está definido el referente. También se denomina **conocimiento base**, a toda la información disponible sobre el problema base y su dominio.

Problema objetivo (u objetivo): Es el nuevo problema a resolver utilizando información referente a uno o más problemas base. De forma análoga al problema base, también se habla de **dominio objetivo** como aquél een el que está planteado el problema objetivo, y de **conocimiento objetivo** como el necesario para resolver el problema objetivo y que se obtiene por transformación de parte del conocimiento base.

En la figura 3.2 se muestra un esquema general de razonamiento por analogía. En ella β representa una inferencia conocida en el dominio de base entre los hechos `A` y `B` (usualmente denominada relación de **causalidad**). La relación entre `A` y `A'`, denotada por α, representa la **similitud** entre ambos hechos (pertenecientes a los dominios de base y objetivo, respectivamente). El **razonamiento analógico** consiste en la inferencia, denotada por β', que se obtiene por transformación de β. La conclusión `B'` que se obtiene también guarda una relación de similitud (α') con la conclusión `B` del dominio de base.

Veamos un ejemplo que, aunque muy simple, servirá para aclarar los elementos que aparecen en el esquema general. Concretamente se trata de considerar la posible analogía entre el movimiento del agua de un río y la corriente eléctrica: por un lado sabemos que el agua se mueve desde los lugares altos hacia los bajos (β); además, el flujo de agua es similar a la corriente eléctrica, de forma que podemos establecer una relación (α) entre la altura y el potencial eléctrico; por tanto, como conclusión podemos derivar (β') que la corriente eléctrica fluye desde el potencial alto hacia el bajo.

3.3 Modelo unificado de analogía

El problema de analogía ha sido abordado de formas muy diversas. Para intentar dar una visión unificada de los componentes básicos de un sistema que realize analogías, se describirá el modelo propuesto por Kedar-Cabelli (keda88), que intenta ser lo más general posible.

Antes de empezar a exponer los componentes del modelo, se ha de establecer el tipo de problema que pretende resolverse. El problema tipo será de la forma:

> Dada como entrada una situación objetivo, dar como resultado una representación aumentada de la misma en la que consten las inferencias analógicas obtenidas de una situación base.

Los componentes o fases del sistema que deba resolver el problema propuesto y su descripción y propósito son:

Recuperación Dada la situación objetivo, el sistema ha de ser capaz de recuperar un caso base potencialmente análogo y poner en correspondencia las partes correspondientes de ambos.

Elaboración Dadas la base y el conocimiento que hay disponible sobre ella, derivar atributos, relaciones o cadenas causales adicionales que puedan ser utilizados posteriormente sobre la situación objetivo.

Mapeo Dada la descripción aumentada del caso base, mapear los atributos seleccionados sobre el objetivo, evidentemente con posibles modificaciones.

Justificación Dados los atributos mapeados sobre la situación objetivo, justificar que son en efecto válidos.

Aprendizaje El aprendizaje llevado a cabo como resultado del razonamiento analógico de las fases anteriores consiste en guardar la representación aumentada de la situación objetivo, en la creación de reglas generales motivadas por la analogía o en el refinamiento de las mismas a partir de más razonamientos sobre la misma o diferentes situaciones base.

El orden de las diferentes fases puede variar de un sistema a otro (dependiendo de la tarea a resolver), pero un sistema capaz de resolver el problema propuesto deberá realizar todas ellas.

Para ejemplificar las fases descritas anteriormente, en la figura 3.3 se representa, desde el punto de vista de este modelo, una inferencia analógica bien conocida: *"el átomo de hidrógeno es semejante al sistema solar"*.

3.4 El modelo de Greiner

Greiner definió un modelo de analogía basado en la abstracción denominado `NLAG`, cuyo objetivo es doble: por un lado definir un modelo formal y claro de proceso analógico; y por otro, describir un algoritmo que use dicho modelo. De ningún modo se pretendía formalizar el uso que la gente realiza de las analogías.

Para ello se define el operador de **inferencia analógica útil**, denotado por $\overset{\sim}{\vdash}$, el cual toma tres operandos: Th es una teoría, $A \sim B$ es una posible analogía (suposición) , y PT es el problema a resolver. El resultado de dicha inferencia es la proposición $\varphi(A)$; es decir:

$$Th,\ A \sim B \quad \overset{\sim}{\underset{PT}{\vdash}} \quad \varphi(A)$$

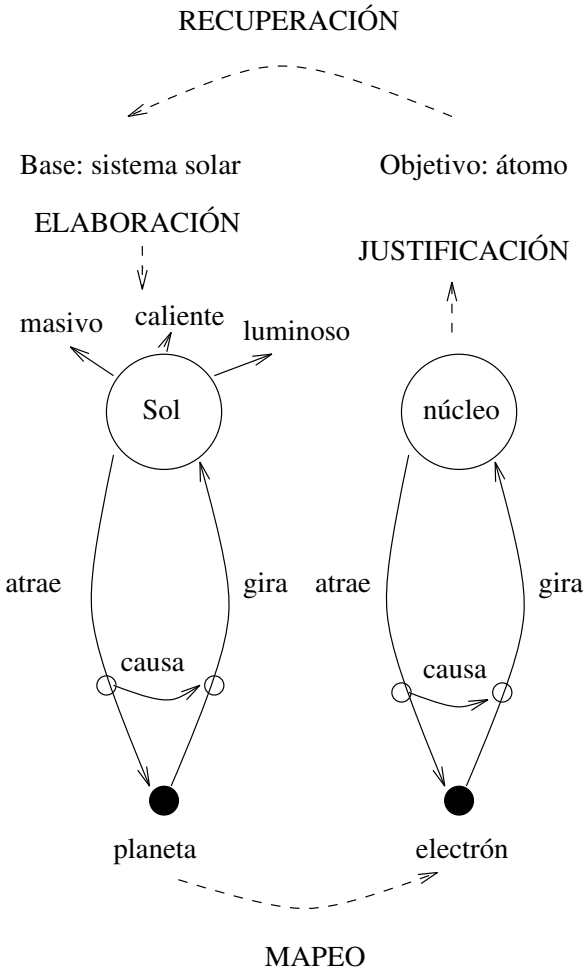


Figura 3.3: Componentes del modelo unificado

donde se satisfacen las siguientes propiedades:

Novedad:	$Th \not\models \varphi(A)$
Consistencia:	$Th \not\models \neg\varphi(A)$
Fundamentación:	$Th \models \varphi(B)$
Útil:	$Th \cup \varphi(A) \models PT$

Para justificar dichas propiedades definiremos, primeramente, qué se entiende por **aprendizaje**; particularizaremos dicha definición al caso del **aprendizaje por analogía**; y, finalmente, definiremos formalmente el concepto de **analogía útil**.

Aprendizaje: Durante toda la presentación, se asumirá que el conocimiento de que disponemos, la teoría Th, es un conjunto de proposiciones (que incluye hechos, reglas, restricciones,

etc.) finito y consistente, referente a los dos dominios que constituyen la analogía. Con este conocimiento es posible resolver un conjunto bien definido de problemas, a saber, aquéllos que pertenezcan a la clausura deductiva de Th. O, a la inversa, ningún sistema deductivo será capaz de resolver un problema σ tal que $Th \not\models \sigma$.

Consideraremos un **paso de aprendizaje** como una expansión de las proposiciones de la clausura deductiva de Th; es decir, aumentaremos Th con una proposición ρ, para formar la nueva teoría Th' consistente en $Th \cup \rho$. Para conseguir que la clausura deductiva de Th' sea mayor que la de Th es necesario que $Th \not\models \rho$; y, para no caer en teorías inconsistentes, es necesario también que $Th \not\models \neg\rho$.

Estas dos restricciones son las propiedades de **novedad** y **consistencia** requeridas para $\varphi(A)$.

Aprendizaje por analogía: Para definir el aprendizaje por analogía en relación con el concepto de aprendizaje definido en el apartado anterior, simplemente se ha de considerar cómo se obtiene la proposición ρ.

Para ello se utiliza el segundo parámetro del operador $\mid\!\sim$, es de decir, la "pista" de que $A \sim B$. Un paso de aprendizaje se califica como analógico si la nueva proposición ρ es sobre el dominio objetivo y, por ello, la denotamos como $\varphi(A)$. A φ se la conoce con el nombre de **fórmula analógica**.

La inferencia analógica utiliza la base de la analogía para dotar de cierta credibilidad a la proposición $\varphi(A)$, insistiendo en que la fórmula analógica sea cierta en el dominio de base; es decir, se requiere que $Th \models \varphi(B)$.

Analogía útil: Por último, se ha de insistir que no vale cualquier fórmula analógica, sino solamente aquéllas que nos permitan resolver el problema PT. Es decir, se requiere que la analogía, además de ser correcta, sea útil. Por ello se ha impuesto la restricción $Th \cup \varphi(A) \models PT$.

Es necesario destacar que esta última condición solamente puede ser determinada a *posteriori*: la única forma de saber si una fórmula analógica es o no útil, es añadirla a la teoría Th y probarla.

Para ilustrar el funcionamiento del modelo de Greiner, desarrollaremos un ejemplo de razonamiento analógico que, como el mostrado anteriormente, hace referencia al campo de la Física.

El problema consiste en resolver un problema sencillo de hidráulica: dada la *conexión en Y* de la figura 3.4, determinar el flujo de agua en Q_c (por ello denotaremos nuestro problema como Q_c =?).

El conocimiento, Th_{CF}, del que disponemos consiste en nociones básicas sobre el flujo del agua y sobre la corriente eléctrica. Dicho conocimiento no permite solucionar el problema, es decir, se cumple que $Th_{CF} \not\models Q_c$ =?.

Además se dispone de la suposición analógica de que:

El flujo de agua (en hidráulica) es similar al de la corriente (en electricidad).

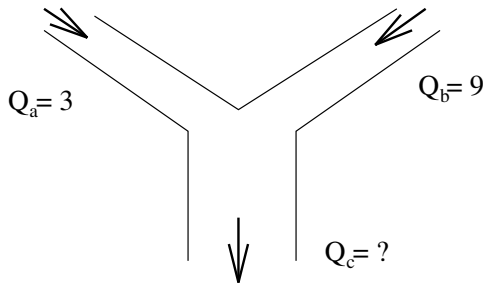


Figura 3.4: Un sencillo problema de hidráulica.

que, en nuestra notación, corresponde a $Flujo \sim Corriente$.

El hecho relevante de la teoría disponible sobre el dominio de base (electricidad), consiste en que ésta cumple la *Primera Ley de Kirchoff*: la **corriente** entrante en una conexión debe ser exactamente igual a la **corriente** saliente de la misma. Denotemos por $K_1(Corriente)$ a dicha ley, que se encuentra en la teoría Th_{CF}, y por R_{K1C} a su clausura deductiva. Dicha ley es la fórmula analógica φ.

Aplicando la suposición analógica, se obtiene la analogía, que aún no sabemos si es de utilidad, $K_1(Flujo)$, es decir, la ley: el **flujo** entrante en una conexión debe ser exactamente igual al **flujo** saliente de la misma.

Si añadimos $K_1(Flujo)$ a la teoría Th_{CF} la resolución del problema es inmediata.

En resumen, y utilizando la misma notación que en la presentación del modelo de Greiner, se tiene que

$$Th_{CF}\ , Flujo \sim Corriente \quad \underset{Q_c=?}{\sim} \varphi(Flujo)$$

que cumple las propiedades

Novedad:	$Th_{CF} \not\models \varphi(Flujo)$
Consistencia:	$Th_{CF} \not\models \neg\varphi(Flujo)$
Fundamentación:	$Th_{CF} \models \varphi(Corriente)$
Útil:	$Th_{CF} \cup \varphi(Flujo) \models Q_c =?$

3.4.1 Las complejidades de la inferencia analógica útil

El ejemplo mostrado en la sección anterior es demasiado simple para mostrar las complejidades que conlleva el razonamiento por analogía. En la presente sección, a partir de un ejemplo ligeramente más complejo, mostraremos las dificultades inherentes a un proceso de este tipo.

El ejemplo de la figura 3.5 muestra un problema similar al planteado en la figura 3.4. Supongamos la misma teoría Th_{CF} (que tampoco permite resolver el problema), la misma suposición analógica $Flujo \sim Corriente$, pero ahora el problema está planteado de forma diferente:

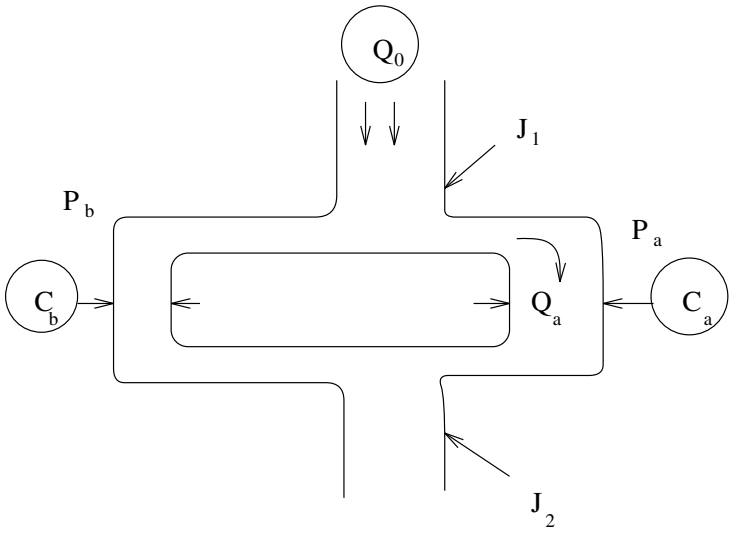


Figura 3.5: Un problema más complicado de hidráulica.

dadas las conexiones mostradas en la figura 3.5, las características C_a y C_b de los tubos, y el flujo de entrada Q_0, determinar el flujo Q_a a través del tubo P_a.

En este caso, la interpretación de suposición analógica consiste en que tanto las leyes de Ohm como las de Kirchoff deben ser **transferidas** al dominio hidráulico. Así pues `NLAG` debe conjeturar que:

Primera ley de Kirchoff: El **flujo** entrante en una conexión debe ser igual al saliente, ya que la **corriente** obedece dicha ley.

Segunda ley de Kirchoff: Existe una cantidad similar a la **diferencia de potencial** asociada a las conexiones en un sistema de tubos cuya suma en un circuito cerrado debe ser igual a cer; denominaremos **diferencia de presión** a esta cantidad en el dominio hidráulico.

Ley de Ohm: La **diferencia de potencial** (diferencia de presión) en los extremos de un dispositivo **resistor** (tubo) es proporcional a la **corriente** (flujo) entrante en él, según la constante de proporcionalidad denominada **resistencia** del dispositivo (característica del tubo).

Ley de conservación: No se pierde **corriente** (flujo) al atravesar un dispositivo **resistor** (tubo).

La **fórmula analógica** resultante es:

$$\varphi_{RKK}(t,c,r,l) \iff \left\{ \begin{array}{rcl} K_1(t) & = & \forall\, j\ \sum_{p:Conn(p,j)}\ t(j,p) = 0 \\ K_2(c) & = & \forall\, l\ \in loop \sum_{<i,j>\in l}\ c(i,j,[x]) = 0 \\ Ohm(t,c,r,l) & = & \forall\, l\ l(d) \Rightarrow c(j_d^1, j_d^2, [d]) = t(j_d^1, d)\ *\ r(d) \\ Cons(t,l) & = & \forall\, d\ l(d) \Rightarrow t(j_d^1, d)\ +\ t(j_d^2, d) = 0 \end{array} \right.$$

Las instanciaciones de dicha fórmula en los dominios de la base (electricidad) y del problema (hidráulica) son:

$$\varphi_{RKK}(Corriente,\ Potencial,\ Resistencia,\ Resistor)$$
$$\varphi_{RKK}(Flujo,\ Presión,\ Tubo,\ Caracter\acute{\subset}stica)$$

La solución del problema, una vez añadida la instanciación de la fórmula analógica en el dominio del problema a la teoría inicial (Th_{CF}) es:

$$\begin{array}{lcll}
Flujo(J_1,P_a)+Flujo(J_1,P_b) & = & Q_0 & \approx \textit{ Primera ley de Kirchoff} \\
Presión(J_1,J_2,[P_a]) & = & Presión(J_1,J_2,[P_b]) & \approx \textit{ Segunda ley de Kirchoff} \\
Presión(J_1,J_2,[P_a]) & = & Flujo(J_1,P_a)*C_a & \approx \textit{ Ley de Ohm} \\
Presión(J_1,J_2,[P_b]) & = & Flujo(J_1,P_b)*C_b & \approx \textit{ Ley de Ohm} \\
\hline
Q_a = Flujo(J_1,P_a) & = & \left\lfloor \frac{C_b}{C_a+C_b} \right\rfloor * Q_0 &
\end{array}$$

Uno de los principales problemas que ha de resolver el sistema `NLAG` consiste en, a partir de la suposición analógica, decidir las **correspondencias** existentes entre los objetos de los diferentes dominios. Por ejemplo, `NLAG` puede elegir como constante de proporcionalidad en la *ley de Ohm* el coste del tubo o el área de una sección longitudinal del mismo, entre otras. Además también podría intentar utilizar conceptos como capacitancia o inductancia en el dominio hidráulico.

El principal problema consiste en que la afirmación "el flujo es similar a la corriente", solamente significa que el flujo satisface **alguna** de las propiedades conocidas de la corriente; no indica ni cuál, ni cómo debe de ser extendida de un dominio a otro.

3.4.2 El algoritmo de inferencia analógica útil

El algoritmo de inferencia analógica útil del sistema `NLAG` consta de los siguientes pasos:

Buscar núcleo: El concepto base se substituye léxicamente en el problema objetivo, y un proceso de encadenamiento hacia atrás busca reglas en la teoría inicial que contribuyan a resolver el problema. El resultado es un conjunto (núcleo) de hechos necesarios para resolver el problema.

Instanciar base: Utilizando un procedimiento de búsqueda en anchura, y a partir de los hechos del núcleo, encontrar una fórmula tal que instanciada en el dominio de base, permita resolver el problema[2].

Instanciar objetivo: Encontrar una instanciación de la fórmula en el dominio objetivo, en la cual el concepto objetivo substituya al concepto de base.

Verificar: Añadir a la teoría inicial la fórmula instanciada en el dominio objetivo e intentar resolver el problema. Comprobar las condiciones de consistencia y utilidad de la conjetura.

[2]Es decir, que cumpla la propiedad de fundamentación.

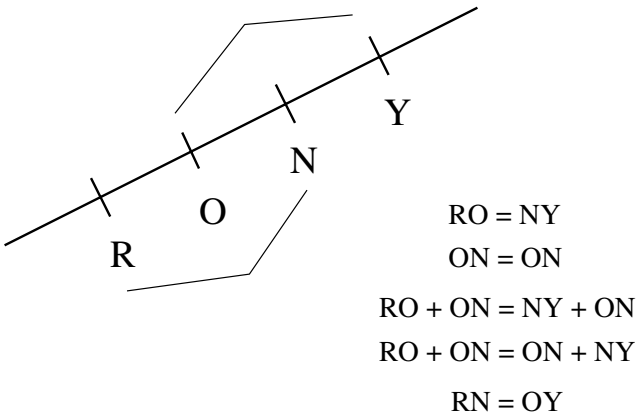


Figura 3.6: Problema original

Aprobación: Pedir al usuario aprobación de las conjeturas propuestas.

3.5 Analogía transformacional

Carbonell **[CARB83a]** introdujo en el mundo de la resolución de problemas la noción de analogía transformacional. La aportación de este sistema es que se considera que existe un espacio (**T-espacio**) en el cual la solución (conocida) puede ser transformada, usando unos operadores (**T–operadores**), hasta convertirla en la solución de un nuevo problema.

Se puede decir que este planteamiento para resolver problemas "no mira *cómo* se resuelve un problema, sino que la solución sea equivalente". Y éste es uno de sus puntos más débiles.

Por ejemplo, se puede querer *construir* una prueba para mostrar que los segmentos RN y OY (ver figura 3.6) son iguales, a partir del hecho de que RO y NY lo son.

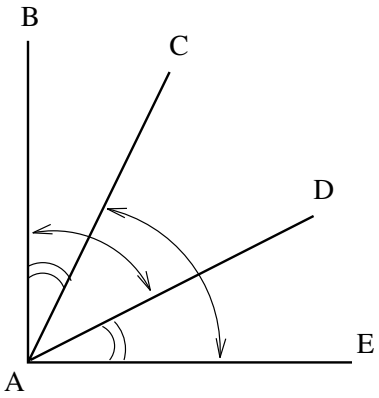
Para ello, en el dominio de los segmentos, se utilizarán operadores de demostración como el de **concatenación de segmentos**, *aditividad* de longitudes, etc.

Ahora bien, si se tiene el problema de mostrar la equivalencia de dos ángulos, mostrado en la figura 3.7, es fácil observar que la aplicación estricta del mismo conjunto de operadores, en el mismo orden, es una solución *analógica*. Se ha encontrado una analogía entre los segmentos de una línea y los ángulos.

La idea del modelo de analogía transformacional consiste en, aplicando algunos T–operadores, como por ejemplo substitución de parámetros, de operadores similares, reordenación, etc., transformar la solución dada en el problema de los segmentos para obtener la del problema de ángulos.

3.5.1 El espacio de búsqueda de las transformaciones

Los modelos tradicionales de **resolución de problemas**, normalmente, no aprovechan información alguna sobre las soluciones obtenidas para problemas similares. Uno de los métodos



BAC = DAE
CAD = CAD
BAC + CAD = DAE + CAD
BAC + CAD = CAD + DAE
BAD = CAE

Figura 3.7: Problema resuelto

clásicos es el conocido como **análisis medios–fines** (*means–ends analysis*). El modelo de **analogía transformacional** consiste, básicamente, en la adaptación de la estrategia de análisis medios–fines para poder aprovechar información del pasado. Por ello, antes de describir este modelo de analogía, enunciaremos los principios básicos de dicha estrategia de resolución.

En la estrategia medios–fines ([NEWE72]), se dispone de la siguiente información (englobada en lo que se conoce con el nombre de **espacio del problema**):

- El conjunto de estados posibles.
- El estado inicial.
- El estado final.
- El conjunto de operadores (y sus precondiciones) que permiten transformar un estado en otro.
- Una **función de diferencias** que computa las diferencias existentes entre dos estados dados.
- Una **tabla de diferencias** que, dada una diferencia, devuelve el operador, o los operadores, que permiten, mediante su aplicación, eliminarla.
- El conjunto de restricciones globales que deben ser satisfechas para que la solución sea aceptada como válida.

La resolución, por análisis medios–fines, en este espacio consiste en:

1. Comparar el estado actual y el final mediante la función de diferencias.
2. Elegir un operador, a partir de la información de la tabla de diferencias, que elimine alguna de las diferencias encontradas[3].

[3]Es por ello, que este método de resolución de problemas también se conoce con el nombre de **reducción de diferencias**.

3. Si en el estado actual es posible aplicar dicho operador, aplicarlo y obtener el nuevo estado actual. Si no, debido a que alguna de sus precondiciones no se cumple en el estado actual, guardar el estado actual y aplicar la estrategia de resolución al problema de satisfacer dichas restricciones.

4. Cuando un subproblema es resuelto, restablecer el último estado guardado y proseguir el trabajo en el problema original.

Como se ha indicado previamente, esta estrategia permite resolver tanto el problema de base como el problema objetivo, pero no utiliza en ningún momento información obtenida en la resolución del primero para resolver el segundo. Es razonable pensar, y de hecho en el caso humano es muy corriente, que información obtenida en la resolución de problemas previos sea de alguna utilidad en la resolución de un problema similar[4].

Para ver las posibles relaciones existentes entre dos resoluciones de problemas similares, deberemos tener en cuenta la comparación entre:

- Los estados iniciales tanto del problema actual como de los problemas utilizados como base.
- Los estados finales tanto del problema actual como de los problemas utilizados como base.
- Las restricciones que, sobre la solución, existan en ambos casos.
- La proporción de precondiciones de los operadores utilizados en la resolución de los precedentes que se satisfacen en la nueva situación (estado inicial del nuevo problema a resolver). Esta medida se conoce con el nombre de **aplicabilidad** del precedente.

Para generalizar la estrategia de medios fines, Carbonell propone un esquema consistente en dos fases diferenciadas:

1. Búsqueda de los precedentes.
2. Transformación de la solución.

En la primera fase, como **medida de similitud** entre problemas, Carbonell propone la utilización de la propia **función de diferencias** utilizada por el algoritmo clásico. Dicha función solamente permite calcular diferencias entre estados pero, según Carbonell, es generalizable para comparar las restricciones que, sobre la solución, se tengan en ambos casos.

La segunda fase es la más compleja y consiste en adaptar la secuencia de operadores en que consiste la solución del caso precedente en una que resuelva el nuevo problema y que, obviamente, cumpla las restricciones existentes en el mismo. La idea consiste en resolver dicha transformación mediante el esquema de análisis medios–fines, no en el **espacio del problema**, sino en el **espacio de soluciones** también llamado **espacio de transformación** o **T–espacio**. Las componentes del nuevo espacio son:

[4] De hecho, es por ello que los profesores insistimos tanto en la necesidad de hacer problemas antes de acudir al examen.

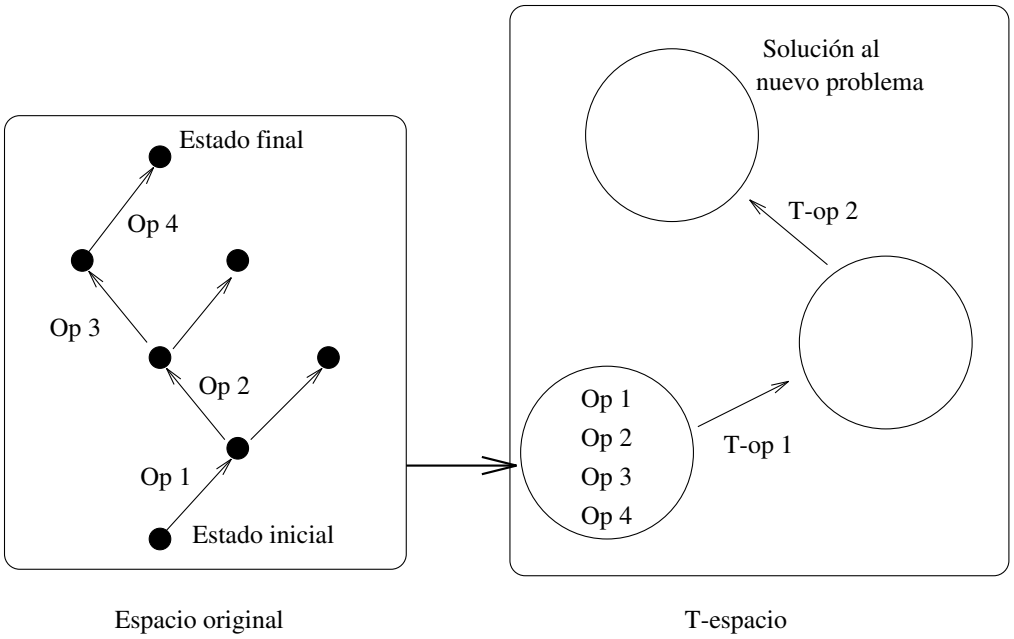


Figura 3.8: La analogía como proceso de búsqueda

Estados: Soluciones (secuencias de operadores) de los problemas planteados en el espacio original (el de problemas). El estado inicial es la resolución del problema precedente y el objetivo consiste en encontrar una solución que resuelva el nuevo problema. Los estados intermedios en el proceso de búsqueda no es necesario que se correspondan con soluciones correctas en el espacio inicial, es decir, las secuencias de operadores que representan pueden no ser ejecutables debido a violar una o más de sus precondiciones.

Operadores: Los operadores, denominados **T–operadores**, son métodos de modificación de soluciones (permutación, inserción, eliminación, etc. sobre la secuencia de operadores en el espacio original).

Función de diferencias: Las diferencias a reducir en el nuevo espacio de búsqueda son las calculadas por la **métrica de similitud** utilizada en la primera fase. Es decir, el progreso hacia la solución viene determinado por transiciones en el T–espacio hacia soluciones de problemas cada vez más parecidos al problema a resolver.

En resumen, en el **espacio de transformaciones**, el estado inicial es la solución base, el estado objetivo es la solución del nuevo problema, y los operadores modifican secuencias de operadores en el espacio original. La búsqueda en este espacio puede realizarse perfectamente con el método clásico de análisis medios–fines.

Debido a ello, podemos decir que el modelo de analogía transformacional resuelve el problema de **analogía** mediante un proceso de **búsqueda**, como indica la figura 3.8.

3.5.2 La arquitectura de la analogía transformacional

Para llevar a cabo el procedimiento indicado anteriormente, un sistema que utilice el modelo de analogía transformacional debe tener una arquitectura similiar a la mostrada en la figura 3.9, el funcionamiento de la cual sigue los siguientes pasos:

1. Se busca en la memoria uno o varios problemas que, de una forma u otra, se correspondan parcialmente con la descripción del problema actual.
2. Obtener la solución de cada uno de los problemas que utilizaremos como referente.
3. Transformar dichas soluciones por un proceso incremental de modificaciones de las mismas, reduciendo las diferencias entre lo que la solución obtenía y aquello que el problema actual requiere. Este proceso, como ya se ha indicado, se realiza por un procedimiento de **análisis medios–fines** en el espacio de soluciones (T–espacio).
4. Si el proceso de transformación es inviable, por ejemplo debido a diferencias irreducibles entre el problema actual y el que se utiliza como precedente, seleccionar un nuevo candidato a precedente y volver al punto anterior, o abandonar el procedimiento de analogía.

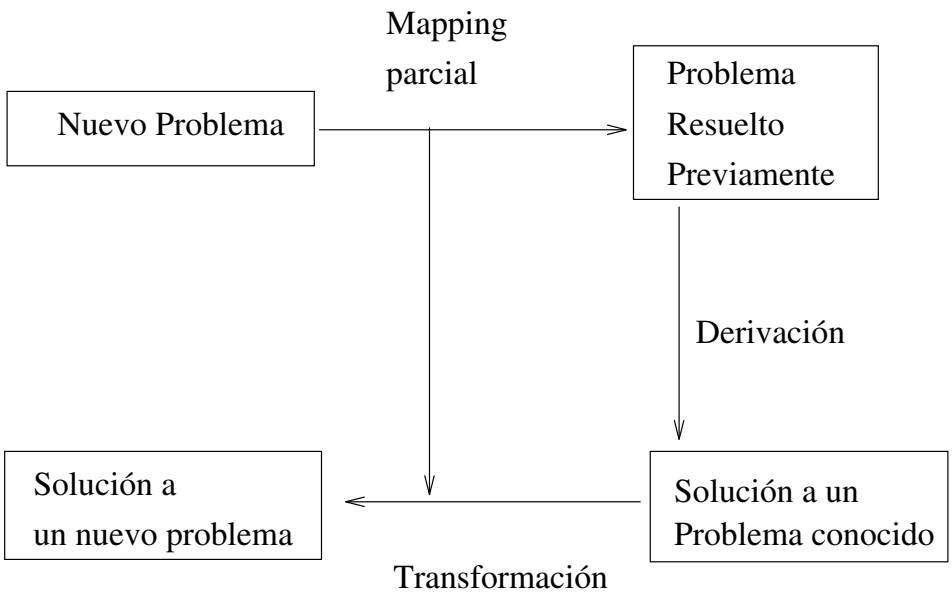


Figura 3.9: Proceso de analogía transformacional

3.5.3 El sistema ARIES

ARIES es una implementación del método de analogía transformacional. Las entradas del sistema incluyen:

- El espacio de problemas original.
- Una métrica de similitud utilizada para calcular el grado de semejanza entre el problema objetivo y los problemas base.
- El espacio de transformación y los T–operadores posibles.
- Los posibles problemas (y sus soluciones) a utilizar como precedentes.

La métrica de similitud depende la comparación entre estados iniciales, estados finales, restricciones a las soluciones y medida de aplicabilidad. Formalmente:

$$D_T =< D_O(S_{I,1}, S_{I,2}), D_O(S_{F,1}, S_{F,2}), D_R(RS_1, RS_2), D_A(SOL_1, SOL_2) >$$

donde

D_T Es la métrica de diferencias a utilizar en el T–espacio.

D_O Es la métrica de diferencias utilizada en el espacio original de problemas.

D_R Es la métrica de diferencias entre restricciones entre problemas.

D_A Es la métrica entre aplicabilidades.

$S_{I,1}, S_{I,2}$ Son los estados iniciales de ambos problemas.

$S_{F,1}, S_{F,2}$ Son los estados finales de ambos problemas.

RS_1, RS_2 Son las restricciones sobre la solución en ambos problemas.

SOL_1, SOL_2 Son las soluciones de ambos problemas.

En `ARIES` el T–espacio de búsqueda está formado por secuencias de aplicaciones de operadores en el espacio original. Por ello, cada T–operador tiene por objetivo modificar una secuencia de operadores. Ejemplos de los operadores más comunes son:

Inserción general: Inserta un nuevo operador en la secuencia solución.

Eliminación: Elimina un operador de la secuencia solución.

Expansión: Expande la resolución de un nuevo subproblema dentro de la secuencia de la solución. Este T–operador es útil cuando un operador de la solución base no es directamente aplicable al nuevo problema. Se resuelve la precondición (subproblema), y la secuencia que la establece se coloca en la secuencia solución.

Substitución equivalente: Substituir un operador en la solución original por otro, u otros, que reduzca las mismas diferencias. Puede ser de utilidad en situaciones en las que, o bien la precondición de un operador en la solución original no puede ser satisfecha, o bien si la presencia de un determinado operador en la solución original viola alguna de las restricciones impuestas en el nuevo problema.

Concatenación de prefijo: Aplicar el procedimiento de búsqueda para encontrar una secuencia de operadores que permitan pasar del estado inicial del problema actual al inicial del problema de base. Si se encuentra dicha secuencia, colocarla como prefijo de la solución al problema inicial.

Mezcla de secuencias: Mezclar las secuencias de operadores de dos soluciones base complementarias encontradas en el proceso de búsqueda de precedentes. La secuencia resultante diferirá de una solución al nuevo problema por la intersección de las diferencias entre cada uno de los precedentes y el nuevo problema a resolver. En caso de que dicha intersección sea nula, ya obtenemos una solución al problema a resolver.

Reordenación de operadores: Reordenar los operadores existentes en la solución actual. En algunos casos esto basta para resolver alguna de las precondiciones violadas por ésta.

Substitución de parámetros: Substituir los parámetros de los operadores de la ssolución precedente por objetos que se hallen en la descripción del nuevo problema a resolver.

Uno de los principales problemas de `ARIES` es el de la organización de la memoria de casos precedentes. Una forma es, mediante técnicas de **agrupación conceptual**, tener organizada la memoria en clases de soluciones, donde la clasificación se realiza mediante la métrica de diferencias descrita anteriormente.

La aproximación del modelo de analogía transformacional, y por tanto del sistema `ARIES`, presentaba muchos problemas, pues la búsqueda en el espacio de las transformaciones es mucho más compleja que la búsqueda en el espacio original del problema. Además, en ningún momento se utiliza información respecto de cómo fue obtenida una solución precedente. Por todo ello, Carbonell **[CARB86]** introdujo la noción de *analogía derivacional*.

3.6 Analogía derivacional

En esta nueva aproximación se describe la solución analógica de un problema como la meta de una tarea jerárquica que almacena información[5] detallada de la solución: es la llamada *información derivacional*. Además, el plan generado se descompone en subplanes - que indican metas intermedias y pueden ser también descompuestas - lo que permite trazar el curso de la solución. Así, se puede decir que la solución es incremental en el sentido de que a cada paso de la solución se resuelve un nuevo problema. El sistema incluso es capaz de emplear trazas de problemas resueltos previamente. Si se añaden a la figura 3.9 – que representa el proceso de analogía transformacional – más problemas resueltos previamente y más derivaciones entonces se tiene un mecanismo de analogía derivacional. Esto significa que el sistema ha de almacenar toda la información generada en cada paso, pero que da especial importancia a aquélla generada por los operadores instanciados sin analizar las razones de esa elección.

En la figura 3.10 se muestra cómo se construye la traza de un proceso de analogía derivacional. Así, dado un problema objetivo (*target problem*), el sistema intenta primero recuperar un plan, si existe, en cuyo caso, si resuelve el problema, termina. En otro caso ha de `Elaborar` una

[5] Esta información incluye subplanes, subobjetivos, alternativas, operadores, caminos que fallan asociados a una explicación, *etc.*

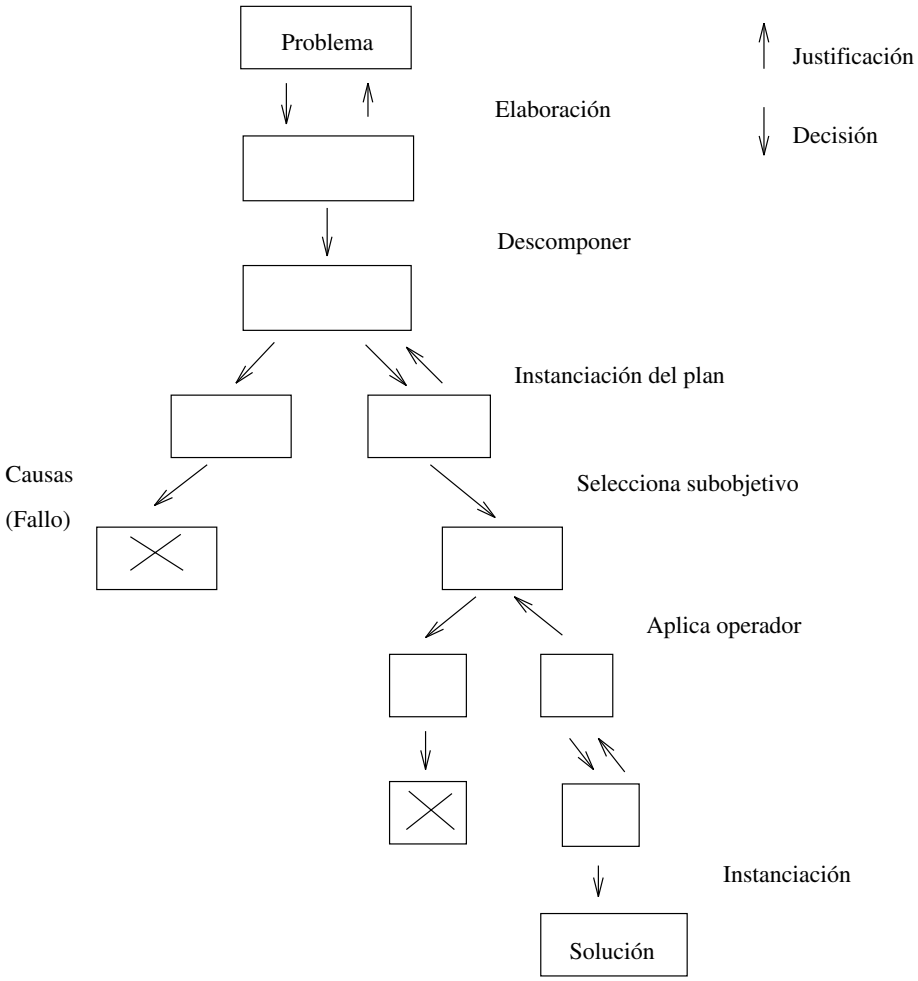


Figura 3.10: Traza derivacional

solución e intenta `Descomponer` el problema en subproblemas, que en principio son más fáciles de resolver.

Existe un mecanismo para `Seleccionar` un subproblema: si, al tratar el subproblema escogido se *falla* al intentar resolverlo, se almacena la secuencia y, si existe, se le asocia una explicación. En el caso de que la solución progrese positivamente se almacena la secuencia de `Operadores` hasta llegar a la solución. Luego se construye una `Justificación` para cada nodo.

La elección de un operador está relacionada con la información derivacional presente, así que, cuando una rama *falla*, el sistema intenta buscar en otras fuentes.

Una de las aplicaciones más notables de este tipo de sistemas es el de buscar nuevas soluciones a problemas ya resueltos para intentar optimizar las antiguas. La analogía derivacional puede ser vista como un sistema de inferencia que puede ser aplicado como un método operativo de razonamiento a partir de experimentos. Un aspecto relevante es la noción de `Justificación` no sólo para facilitar la reconstrucción de una solución sino porque permite evaluar lo "aprendido" [6]. Además resulta una herramienta poderosa de adquisición incremental de experiencias.

3.7 Resumen

En el presente capítulo se han presentado los conceptos de razonamiento y aprendizaje por analogía, y se han descrito diversos sistemas que hacen uso de ellos.

Dichos sistemas enfocan el uso de analogías desde muy diversos puntos de vista que van desde la generalización de los procesos de deducción a partir de la ampliación de los axiomas de una teoría inicial, como en el caso del sistema `NLAG` (sección 3.4), hasta la modificación de estrategias de búsqueda para poder incluir información de las resoluciones de problemas anteriores (secciones 3.5 y 3.6). Por ello también se ha presentado un modelo que pretende unificar a todos ellos (sección 3.3).

Es necesario remarcar que la construcción de sistemas que exploten todas las capacidades que el uso de la analogía permite es tema de frontera en la investigación actual tanto en el desarrollo de sitemas capaces de hacer uso de ello, como en el del estudio teórico de los procesos inductivos subyacentes.

Entre los aspectos por desarrollar, cabe destacar:

- Encontrar las estructuras de representación que permitan, de forma cómoda, detectar las posibles analogías existentes entre situaciones de dominios distintos.
- Métodos de organización de la memoria de casos predentes.
- Definir reglas heurísticas que, de entre todas las analogías posibles, permitan escoger aquéllas que, con cierta seguridad, conduzcan a razonamientos válidos.
- Definir claramente el concepto de *similitud* entre casos.

La analogía es un tipo de herramienta que hace patente el valor de la memoria, no sólo como recipiente, sino como un agente activo en el aprendizaje y la resolución de problemas.

[6] Una vez más cabe destacar que un agente debe ser capaz de reconocer las situaciones que conducen a un fracaso rápidamente y poder explicar el porqué.

3.8 Ejercicios

1. Aplica el modelo **NLAG** al caso de la analogía mecánica de un circuito RLC mostrada en la figura 3.1.
2. Relacionar las fases del modelo unificado de analogía (sección 3.3) y las propiedades del operador $\sim$ del modelo de greiner (sección 3.4).
3. Explica las principales semejanzas y diferencias entre la analogía y el razonamiento basado en casos (capítulo 8).

Capítulo 4 Aprendizaje basado en explicaciones

4.1 Introducción

Hasta los años 80, los métodos de aprendizaje automático más utilizados eran los inductivos o empíricos. Estos métodos se basan en la observación de muchos ejemplos y contraejemplos de un concepto para poder obtener una descripción general de ese concepto (ver 2.2). No obstante, los métodos inductivos tienen inconvenientes, uno de los cuales es que no preservan la veracidad y sí la falsedad. Esto significa que sólo podemos estar seguros de que si algo era falso antes de generalizar seguirá siéndolo después. Por el contrario, no hay la seguridad de que la generalización realizada sea cierta, puesto que, en general, sólo puede ser justificada por un número finito de ejemplos.

Para solucionar este problema, empezaron a estudiarse los métodos deductivos o analíticos. La idea es usar la deducción lógica de manera que la solución obtenida para un problema esté plenamente justificada y pueda ser generalizada. Para ello se necesita una gran cantidad de conocimiento del dominio y es suficiente un solo ejemplo de entrada para generalizar correctamente. Así, a partir del ejemplo y usando la teoría del dominio conocida, se obtiene de forma deductiva la descripción de un nuevo concepto. De este modo puede asegurarse que las generalizaciones son correctas puesto que los métodos deductivos preservan la veracidad.

Actualmente, hablar de métodos deductivos de aprendizaje es sinónimo del aprendizaje basado en explicaciones o *Explanation-based Learning* (**EBL**). **EBL** es un término que apareció en los años 80 ([**DEJO86**], [**SILV83**], [**MITC83**], [**CARB83b**]) como intento de etiquetar algunos métodos de aprendizaje usados en sistemas ya existentes como STRIPS ([**FIKE72**]), HACKER ([**SUSS75**]) o el jugador de póker de Waterman ([**WATE70**]). Así, podemos decir que el **EBL** es la convergencia de cuatro líneas de investigación:

Generalización Justificada. Dado un cierto conocimiento del dominio y un conjunto de ejemplos positivos y negativos, buscar un concepto que incluya los positivos y excluya los negativos. Este nuevo concepto debe ser una consecuencia lógica del conocimiento del dominio y de los ejemplos [**RUSS86**].

Chunking. Es la compilación de una secuencia de reglas u operadores en un solo macrooperador que tenga el mismo efecto que la secuencia original [**ROSE86**]. Como se verá en la sección 4.5.3, SOAR es una arquitectura que utiliza *chunking*.

Operacionalización. Consiste en convertir en operacional una expresión que no lo es. Entendemos por operacional una expresion escrita en términos de acciones que pueden ser ejecutadas directamente por el sistema [**MOST83**].

Analogía Justificada. Dado conocimiento del dominio, un ejemplo X y un concepto objetivo Y, se trata de buscar una característica F tal que si F(X) es cierta, se infiera de forma deductiva que F(Y) es cierta [**DAVI86**].

Así pues, a partir de ahora hablaremos del **EBL** como paradigma de los métodos de aprendizaje deductivos. A lo largo de este capítulo se describirán los métodos **EBL** primero de forma intuitiva y después más formalmente realizando un análisis a nivel de conocimientos. Tanto en esta descripción como en los ejemplos de sistemas que usan **EBL** (sección 4.5) se supone que la teoría del dominio es completa y consistente y que, por lo tanto, el **EBL** no presenta ningún problema. Como se verá en la sección 4.4 si la teoría del dominio no cumple estas condiciones pueden aparecer algunos problemas.

4.2 Descripción intuitiva del EBL

Supongamos que nos muestran un automóvil azul de una determinada marca y nos dicen que es un medio de transporte terrestre. Si después nos muestran un automóvil rojo y de distinta marca sabremos identificarlo igualmente como un medio de transporte terrestre. Esto es debido a que sabemos abstraer cuál es la información relevante de un ejemplo, de manera que no daremos importancia al color o a la marca de un automóvil sino, en este caso, a su función. De la misma manera, reconoceríamos como medio de transporte terrestre un autocar o un tren aunque pocas cosas tienen en común. Los métodos **EBL** intentan modelizar este tipo de aprendizaje mediante el cual con un solo ejemplo de un concepto somos capaces de reconocer un objeto de la misma clase.

Supongamos ahora que estamos aprendiendo a integrar. Sabemos las reglas de integración, la tabla de integrales inmediatas y los métodos que podemos usar para resolverlas. Al principio, cuando nos dan una integral para resolver, vamos probando métodos hasta encontrar uno que nos dé la solución de forma sencilla. Esto es, si decidimos aplicar un método y éste nos lleva a una expresión más complicada, lo descartamos y probamos con otro. A medida que aumentamos nuestra experiencia en la resolución de integrales sabremos "a simple vista" cuál es el método más apropiado para obtener la solución. Un método **EBL** puede asociarse a un sistema de resolución de problemas de manera que nos permitirá aprender reglas de control que mejorarán su eficiencia.

Obsérvese que en los dos casos mencionados anteriormente se posee una cierta experiencia o conocimiento del dominio (sobre vehículos o sobre integración) que va mejorando a medida que se resuelven nuevos problemas. De esta manera podemos reconocer que un objeto es o no un medio de transporte aunque no hayamos visto nunca ninguno igual o podemos reconocer ciertas características de la función a integrar que nos permitan elegir el método más apropiado.

Resumiendo, el principal objetivo de un método **EBL** es mejorar la teoría del dominio existente a partir de un solo ejemplo de entrada. Para ello necesita tener inicialmente una teoría

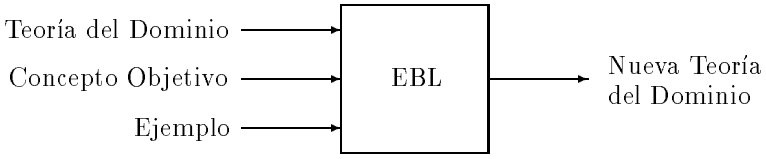


Figura 4.1: Entradas y salidas de un método EBL

del dominio completa que le permita demostrar de forma deductiva que nuevos conceptos pertenecen a la misma clase o bien que permita deducir nuevas reglas de control de manera que puedan resolverse problemas de forma más eficiente. Es importante destacar que un método **EBL**, a diferencia de los métodos inductivos, nunca incorpora nuevo conocimiento, sino que va obteniendo la clausura deductiva de la información inicialmente existente (véase la sección 4.6).

4.3 Descripción formal del EBL

Una vez introducido de manera intuitiva el tipo de aprendizaje que se pretende capturar con los métodos **EBL**, vamos a dar una descripción formal. Para ello se definirán primero los conceptos utilizados y después se realizará una descripción de los métodos **EBL**. Se evitará dar detalles de implementación para concentrar la explicación sólo en las tareas que deben realizarse y el conocimiento necesario para ello. En [**ARME93**] puede encontrarse más información sobre la metodología seguida para realizar este análisis así como una descripción detallada a nivel de conocimientos de sistemas representativos que realizan **EBL**, algunos de los cuales estudiaremos en la sección 4.5.

4.3.1 Definiciones

Dados una teoría del dominio , un concepto objetivo y un ejemplo, los métodos **EBL** tratan de mejorar la teoría del dominio (figura 4.1). Vamos a definir cada uno de estos conceptos.

Teoría del dominio o conocimiento de respaldo. Es información específica del dominio particular sobre el que trata nuestra aplicación (mecánica, integración, medicina, biología, etc). Supondremos que dicha teoría es completa y consistente, es decir, que contiene toda la información necesaria para deducir nuevas cosas sobre el dominio y que no contiene información incorrecta.

Concepto objetivo. El método **EBL** debe determinar una definición efectiva y operacional de este concepto. Entendemos por efectiva que permita llegar a la solucion más rápidamente y por operacional que pueda ser utilizada por el sistema. Dependiendo de la aplicación, este concepto objetivo puede ser una clasificación, un teorema a demostrar, un plan para conseguir un objetivo o una regla de control para la resolución de un cierto problema.

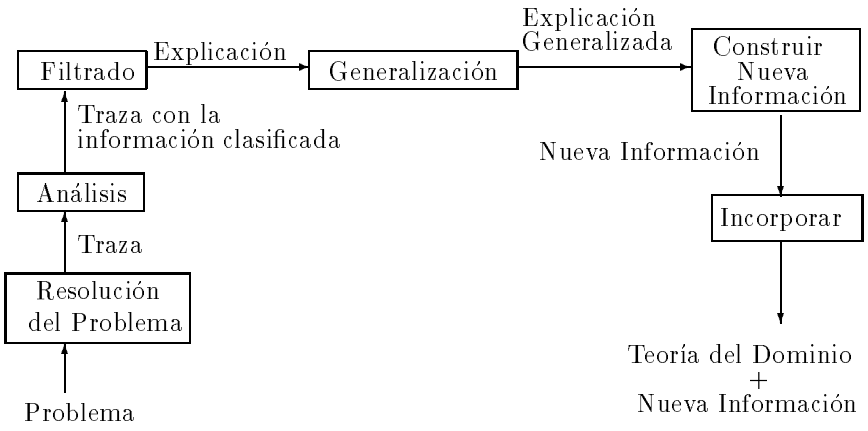


Figura 4.2: Descomposición de un método EBL

Ejemplo. Es una instancia positiva del concepto objetivo. Este ejemplo permite descubrir qué características del problema pueden ser más relevantes en el futuro.

Nueva teoría del dominio. Esta teoría es la misma que la que se tenía inicialmente, pero mejorada con el resultado de la resolución del nuevo problema. Así por ejemplo, puede contener una nueva definición operacional de un concepto ya conocido o una nueva regla de control.

4.3.2 Componentes de los métodos EBL

Los métodos **EBL** tienen dos pasos básicos. El primero de ellos es la construcción de una explicación que justifica porqué el ejemplo es una instancia positiva del concepto objetivo. El segundo paso consiste en generalizar esa explicación de manera que en el futuro pueda ser aplicada a situaciones similares. Analizando estos dos pasos a nivel de conocimientos encontramos las tareas de la figura 4.2. Vamos a describir ahora cada una de estas tareas ilustrándolas con el desarrollo de un ejemplo cuyas entradas pueden verse en la figura 4.3.

Resolución del problema.

La tarea de resolución de problemas debe ser capaz de encontrar el concepto objetivo ayudándose para ello de la teoría del dominio y del ejemplo. El resultado es una *traza* que contiene toda la información que se ha ido activando en el curso de la resolución del problema. Así pues, la traza contiene, además del camino hacia la solución, todos aquellos caminos que o no se han intentado o que se han intentado y han fracasado. El método aplicado para la resolución del problema puede ser cualquier método de búsqueda (en profundidad, en anchura, etc) y, por lo tanto, es independiente del dominio. En nuestro ejemplo se trata de encontrar una definición operacional del concepto `tigre` encadenando las reglas de la teoría del dominio de la figura 4.3. Usaremos un algoritmo en profundidad para intentar demostrar que el ejemplo

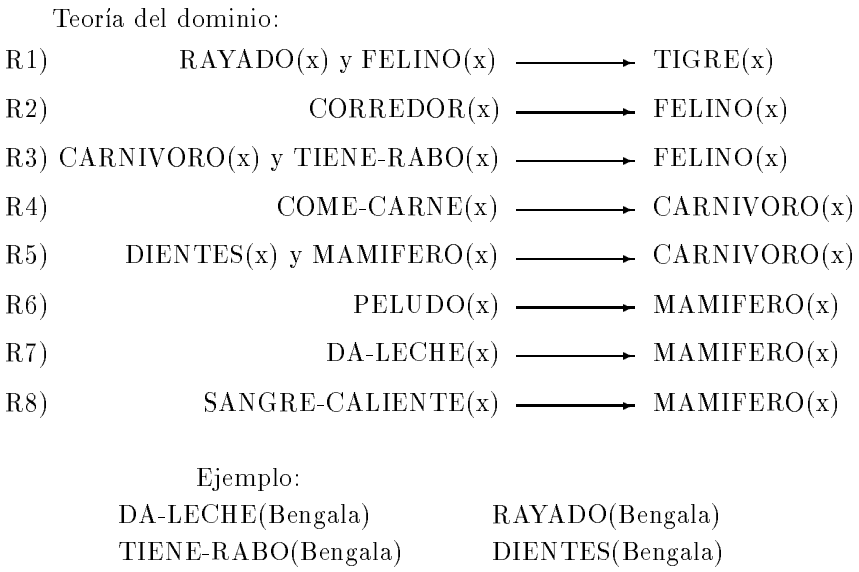


Figura 4.3: Entradas del ejemplo del tigre

de entrada es una instancia positiva de tigre. La traza representando esta demostración puede verse en la figura 4.4.

Análisis de la traza.

El análisis de la traza es el primer paso que permitirá la construcción de una explicación. Una *explicación* es aquella parte de la traza que contiene información que se prevé útil en un futuro para la resolución de nuevos problemas. Para este análisis se usan dos criterios: el de operacionalidad y el de relevancia. Todos los métodos **EBL** tienen de forma implícita o explícita un *criterio de operacionalidad* mediante el cual se indica cuáles son las acciones directamente ejecutables por el sistema. El *criterio de relevancia* es el que permite decidir qué información puede ser útil en el futuro. El más utilizado es aquél que considera relevante la información que forma parte del camino que lleva a la solución, aunque hay otros. En la Evaluación Parcial Perezosa o *Lazy Partial Evaluation* ([**CLAR92**]), por ejemplo, el árbol de la traza se expande en anchura utilizando el algoritmo en anchura de manera que cuando encuentra la solución pueden haber quedado nodos sin expandir. Interesa tener constancia de los nodos no expandidos totalmente de manera que en el futuro no tenga que recalcularse todo el camino sino que pueda aprovecharse lo ya calculado. Así pues, en este caso la información relevante será aquella que no forme parte de un camino fracasado. En nuestro ejemplo se considera relevante aquella información que pertenece al camino que lleva a la solución y que contiene sólo predicados operacionales (subrayado en la figura 4.4). En este caso el criterio de operacionalidad requiere que la expresión final esté descrita en términos de los predicados

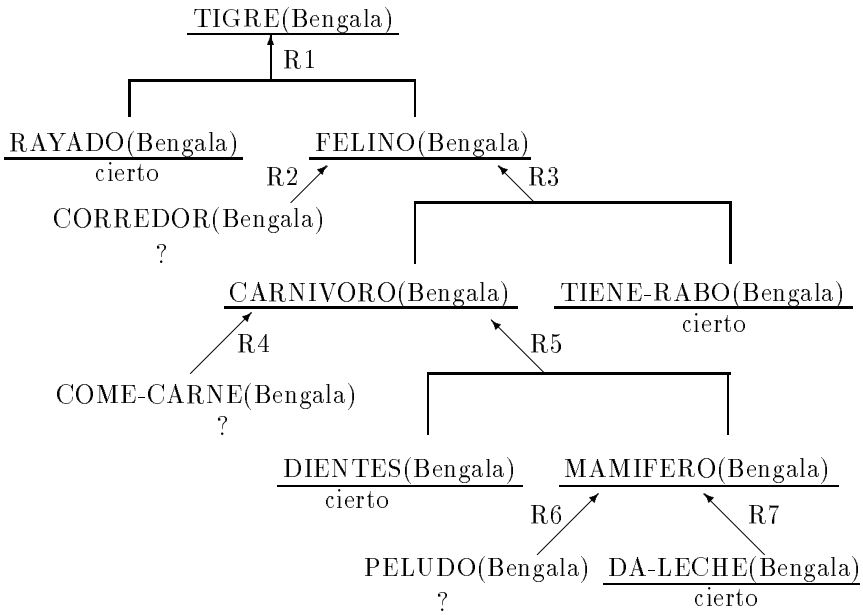


Figura 4.4: Traza de la resolución del problema.

usados para describir el ejemplo de entrada (`CORREDOR, RAYADO`, etc).

Filtrado.

La tarea de filtrado es la que construye la explicación separando la información marcada como relevante de la que no lo es. Así, para nuestro ejemplo, la explicación es la de la figura 4.5 que contiene sólo la información que forma parte del camino que lleva a la solución habiendo rechazado la restante.

Generalización de la explicación.

Normalmente, la generalización consiste en substituir constantes por variables o estructuras variables de manera que la explicación siga siendo válida. La mayoría de sistemas usan métodos basados en este criterio aunque pueden tener ciertas peculiaridades, como por ejemplo SOAR (vease sección 4.5.3). El método más utilizado es el *algoritmo de regresión de objetivos* propuesto por Waldinger ([**WALD77**]) y Nilsson ([**NILS80**]).

Formalmente, se dice que *regresionar* una fórmula F a través de una regla R es un mecanismo para determinar las condiciones necesarias y suficientes bajo las cuales puede usarse la regla R para inferir F. La aplicación de este algoritmo a nuestro ejemplo da la generalización de la figura 4.6. Dicha generalización se ha obtenido variabilizando primero la raíz de la explicación, en este caso `TIGRE(Bengala)` y propagando la substitución de `Bengala` por `X` a los hijos. Los hijos son las condiciones de todas las reglas que permiten deducir `TIGRE` y que, en este caso, sólo

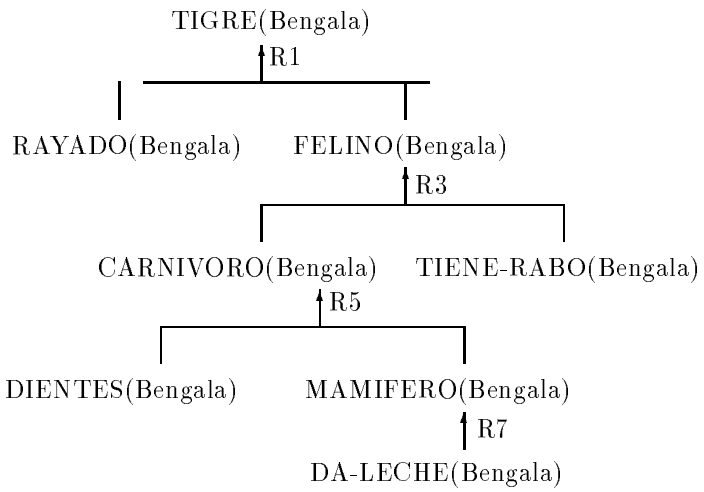


Figura 4.5: Explicación del ejemplo del tigre.

hay una. En caso de haber más de una regla, como pasa para deducir `CARNIVORO` o `MAMIFERO`, se utilizan las reglas R5 y R4 y R6, R7 y R8 respectivamente para variabilizar propagando la substitución a todas ellas. Así, se van obteniendo todas las sustituciones correspondientes a todos los predicados incluídos en la explicación. Nuestro ejemplo es sencillo puesto que la substitución es siempre la misma (`Bengala` por `X`). No obstante, esto no tiene porqué ser así. En **[MITC86]** puede encontrarse un ejemplo en el cual no todas las constantes pueden variabilizarse.

Construcción de nueva información.

Una vez generalizada la explicación, ésta debe ser operacionalizada, es decir, traducida al formato adecuado para que pueda ser usada en el futuro. Dicho formato pueden ser reglas de dominio o de control. Las reglas de dominio expresan nuevas definiciones de conceptos y las reglas de control expresan nuevas heurísticas que permitirán resolver el problema más eficientemente en el futuro. Normalmente, la parte izquierda de la nueva regla serán las hojas del árbol de explicación generalizado, mientras que la parte derecha será la raíz (concepto objetivo). Se obtiene una regla para cada una de las posibles combinaciones de reglas aplicables. Así, en nuestro caso a partir del árbol de la figura 4.6 se podrían construir las siguientes reglas:

NR1) si RAYADO(x) y DIENTES(x) y PELUDO(x) y TIENE-RABO(x) entonces TIGRE(x)

NR2) si RAYADO(x) y DIENTES(x) y DA-LECHE(x) y TIENE-RABO(x) entonces TIGRE(x)

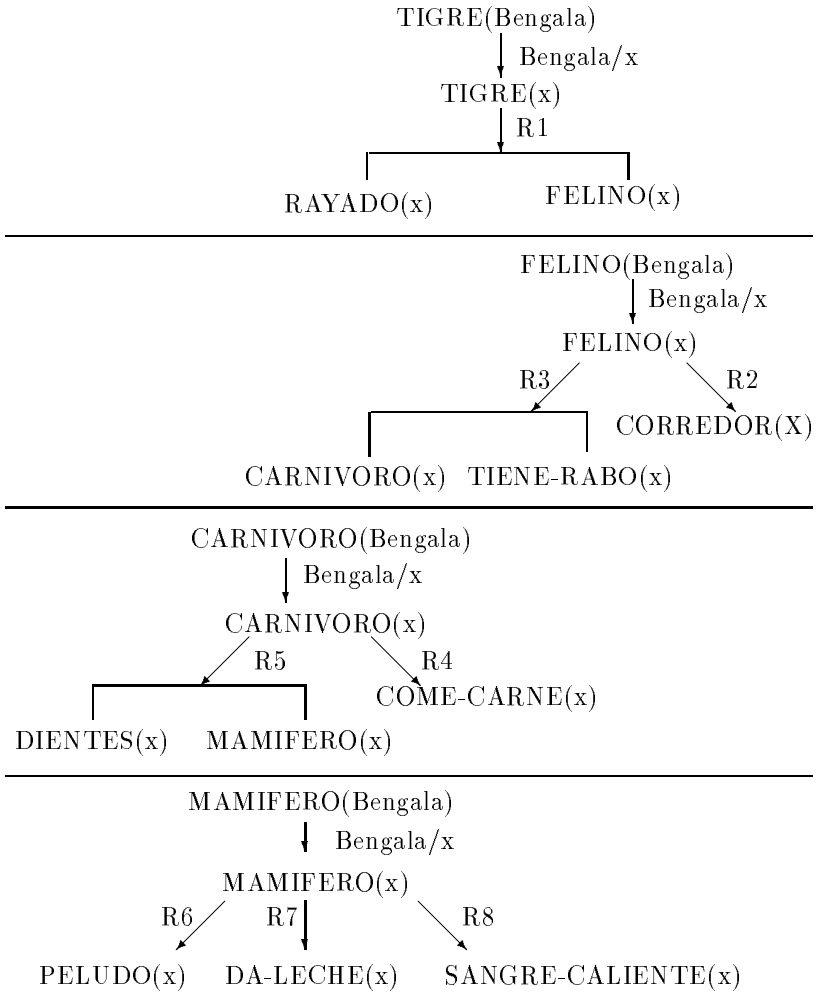


Figura 4.6: Explicación generalizada con el algoritmo de regresión de objetivos.

NR3) si RAYADO(x) y DIENTES(x) y SANGRE-CALIENTE(x) y TIENE-RABO(x) entonces TIGRE(x)

NR4) si RAYADO(x) y COME-CARNE(x) y TIENE-RABO(x) entonces TIGRE(x)

NR5) si RAYADO(x) y CORREDOR(x) entonces TIGRE(x)

La regla NR1 ha sido obtenida usando el camino del árbol generalizado que lleva de la raíz a las hojas pasando por R1, R3, R5 y R6. La segunda regla se ha obtenido del mismo modo pero con la regla R7 en lugar de la R6 y así sucesivamente. No obstante, y para mejorar la eficiencia del sistema, hay métodos que sólo construyen una regla (como el EBG de Mitchell), mientras que otros, como la Evaluación Parcial Perezosa, puede crear algunas reglas que no son totalmente operacionales debido a que corresponden a información de la traza que no ha sido completamente expandida.

Incorporación de nueva información.

El objetivo de esta tarea es hacer que las nuevas reglas creadas queden disponibles de manera que puedan utilizarse para la resolución de nuevos problemas. En la mayoría de sistemas la nueva información es añadida a la ya existente. Así, en el ejemplo, las cinco reglas obtenidas se añadirían a la teoría del dominio, con lo cual se tendrían seis definiciones de `TIGRE`. Este ejemplo hace evidente que la contínua incorporación de reglas (en el mejor de los casos una por cada problema resuelto), puede degradar la teoría del dominio de manera que puede ser difícil encontrar la regla adecuada. En algunos casos, las reglas construídas pueden ser inaplicables o aplicables a situaciones extrañas demasiado específicas que raramente se darán. Esta problemática (ya mencionada en el capítulo 1) se denomina *utilidad del conocimiento aprendido*. En la sección 4.4.1 se hablará en detalle de ella.

4.4 Problemas del EBL

Como ya se ha dicho, los métodos **EBL** son de gran ayuda para mejorar la eficiencia de la resolución de problemas, suponiendo siempre que hay una teoría del dominio completa y consistente. No obstante, cuando se quiere trabajar sobre dominios reales, por sencillos que éstos sean, aparecen ciertos problemas. Básicamente, estos problemas pueden agruparse en dos clases. La primera de ellas es la que Ellman **[ELLM89]** llama *Reformulación de la Teoría* y que consiste en garantizar que la información aprendida es realmente más útil. La segunda clase de problemas comprende la *Revisión de la Teoría*, es decir problemas que son consecuencia del contenido de la teoría del dominio disponible. Vamos a analizar cada una de estas clases.

4.4.1 Reformulación de la Teoría

El aprendizaje **EBL** permite reformular la teoría en el sentido de que se incorporan nuevas definiciones o reglas de control a la teoría existente. Sin embargo, la incorporación sistemática de la información aprendida puede plantear dos problemas. El primero es que es posible incorporar a la teoría algo que quizá nunca más será útil en el futuro pero que, en cambio,

aumenta mucho su tamaño. El segundo problema es que, en general, las reglas aprendidas son más complejas que las iniciales de manera que el coste de cotejar las condiciones para comprobar si una regla es aplicable puede ser mayor. Intuitivamente esto puede ser explicado de la siguiente manera.

El ciclo de control de la resolución de problemas es: buscar reglas aplicables, en caso de que haya más de una regla aplicable escoger una entre ellas y aplicarla. Este ciclo debe repetirse hasta encontrar la solución deseada y puede ser bastante costoso. Con el aprendizaje **EBL** se pretende reducir al máximo, y si es posible a una sola, las veces que tenga que ejecutarse este ciclo. Volviendo al ejemplo desarrollado en la sección 4.3.2, se ha visto que para demostrar que Bengala es un tigre ha hecho falta aplicar 7 reglas (algunas de ellas sin éxito). Gracias a las definiciones aprendidas, para un ejemplo de entrada parecido, bastará aplicar la nueva regla NR1 para obtener directamente la solución, de manera que el ciclo se habrá ejecutado una sola vez. En el mismo ejemplo se ha visto que se han generado cinco nuevas reglas a partir de un solo ejemplo de entrada. Esto ya da idea de que si siempre se incorpora toda la nueva información aprendida, encontrar cuál es la regla apropiada de entre todas las existentes no será una tarea sencilla puesto que significa recorrer todas las reglas de la teoría hasta encontrar una que sea aplicable. Así pues, es claro que el EBL no garantiza más eficiencia en la resolución del problema puesto que el conocimiento de control tiene un coste oculto (el de cotejar). Para producir una mejora real de la eficiencia, un método EBL debe generar conocimiento de control efectivo, es decir, que produzca más beneficios que costes evitando que, en algunos casos, sea más rápido usar la teoría del dominio inicial (aunque para ello se tuviera que ejecutar varias veces el ciclo de control) que buscar la regla adecuada de entre las muchas que la teoría puede contener.

Así, formalmente, el problema de la *degradación de la teoría* aparece básicamente por los tres motivos siguientes:

Baja frecuencia de aplicación. Una descripción aprendida puede ser útil cuando es aplicable, pero puede serlo muy raramente por ser demasiado específica. El coste de comprobar repetidamente su aplicabilidad puede no compensar su eficiencia aunque el tiempo de comprobación sea pequeño.

Alto coste de cotejar las reglas. Una descripción aprendida puede ser útil cuando se aplica y serlo frecuentemente pero el coste de determinar su aplicabilidad puede resultar demasiado alto como para ser ventajoso.

Bajo beneficio. Muchas veces el hecho de crear una nueva regla no mejora en mucho la eficiencia de la teoría inicial. Supongamos que se crea una regla que permite encontrar un camino para ir de A a B salvando obstáculos. Esta regla no producirá demasiado beneficio si el coste de salvar un obstáculo es pequeño o bien si hay pocos obstáculos que salvar.

De todo lo visto hasta ahora, es fácil deducir que se tendrá que llegar a un compromiso entre incorporar siempre la información y no incorporarla nunca. Este problema puede atacarse de varias maneras. Una puede ser estimando, antes de generalizar, si determinada información se prevé útil y si vale la pena su generalización. Así, si un objetivo es solicitado a menudo, será

interesante generalizar el camino que lleva a él. Otra forma, como hace el sistema PRODIGY, es generalizar siempre pero evaluar si dicha generalización será útil en el futuro. (Ver sección 4.5.4). SOAR generaliza y guarda siempre la nueva información pero antes reordena las condiciones de una regla de manera que las más restrictivas se evalúen primero (sección 4.5.3). De esta manera muchas veces no hará falta evaluar la condición entera. PRODIGY también trata este problema usando el proceso de compresión que se analiza en la sección 4.5.4

4.4.2 Revisión de la Teoría

El problema de la revisión de la teoría aparece especialmente cuando se trabaja con dominios reales en los cuales es difícil especificar absolutamente todo el conocimiento. En estos casos es corriente que la teoría sea incompleta, incorrecta, inconsistente o intratable, pudiéndose dar más de uno de estos casos a la vez. Vamos a estudiar cada uno de estos problemas y algunas de las soluciones propuestas.

Teoría Incompleta

Supongamos que en una teoría se tiene que las hojas de los árboles son verdes y que el ejemplo de entrada es una hoja amarilla. El sistema no sabrá explicar este ejemplo porque le faltaría una regla del tipo "si las hojas son amarillas es que es otoño". Este es el problema de la teoría incompleta y se da cuando la teoría no puede explicar algún ejemplo por no disponer de toda la información necesaria. La solución propuesta es intentar explicar al máximo el ejemplo e identificar y conjeturar nuevas reglas que completarían la explicación. Estas explicaciones parciales son más efectivas cuanto menos le falta a la teoría para ser completa o si un maestro le selecciona adecuadamente los ejemplos de entrada de manera que vaya aprendiendo la información que le falta. Hay varias técnicas para completar una explicación. Una de ellas es la propuesta por Wilkins **[WILK88]** en el sistema ODYSSEUS. En dicho sistema se intenta construir una explicación para cada ejemplo de entrada. Cuando no puede encontrar una explicación se presenta entonces una oportunidad para aprender ya que tratará de conjeturar modificaciones de la teoría. Si una de estas conjeturas le produce un buen resultado entonces será añadida a la teoría.

Obsérvese que la teoría se completa añadiendo información pero en ningún caso se borra o modifica la ya existente con lo cual debe suponerse implícitamente que la teoría inicial, aunque incompleta, era correcta. Cabe destacar también que la nueva información se obtiene de forma no deductiva con lo cual no se puede asegurar que la nueva sea una teoría correcta.

Teoría Incorrecta

Se dice que la teoría es incorrecta cuando el sistema comete algún fallo de predicción. En este caso primero hay que identificar la causa del fallo y después intentar modificar convenientemente la teoría. En general, para identificar la parte de la teoría causante del fallo se utiliza un *algoritmo de asignación de culpa*. Muchas veces se dispone de reglas específicas del dominio de manera que para cada tipo de error se da una manera de solucionarlo puesto que es capaz de identificar sus posibles causas.

Teoría Inconsistente

Se dice que la teoría es inconsistente cuando el sistema llega a predicciones contradictorias. La inconsistencia puede ser debida a la teoría, pero también puede tener su origen en inconsistencias entre la teoría y las observaciones. Los métodos para solucionar este problema son parecidos a los usados para detectar incorrecciones. Un ejemplo típico de teoría inconsistente que es amplio objeto de estudio son las llamadas *teorías promiscuas*. No vamos a estudiar aquí estas teorías, simplemente diremos que son aquellas capaces de dar una explicación plausible para cualquier situación. Pueden encontrarse ejemplos en **[LEBO86]** y **[RIES83]**.

Teoría Intratable

La teoría es intratable cuando para dar una predicción se necesitan más recursos de los que se dispone. Un ejemplo de teoría intratable es el juego de ajedrez. Si para cada jugada se examinasen todas las posibilidades de movimiento el juego sería interminable y además agotaría todos los recursos computacionales de espacio y tiempo. Las teorías intratables se comportan como si fueran incompletas o incorrectas puesto que pueden no dar solución o dar una solución errónea. Normalmente este problema se resuelve usando heurísticas que permiten restringir el espacio de búsqueda (piénsese en el ajedrez) aunque entonces se corre el peligro de convertir la teoría en inconsistente.

Así pues, para implementar un método **EBL** que funcione sobre un dominio real deben tenerse en cuenta estos problemas que no siempre son solucionables de manera fácil. Actualmente se tiende a la integración de métodos inductivos y deductivos de manera que puedan complementarse. Así, un método deductivo asegurará la corrección de la generalización y permitirá encontrar de manera dinámica las características relevantes mientras que un método inductivo puede solventar la falta de una teoría del dominio completa.

4.5 Ejemplos de sistemas que usan EBL

En esta sección analizaremos cuatro de los sistemas considerados como los más representativos de los que utilizan **EBL**. Con su estudio pretendemos dar una visión de las aplicaciones que puede tener el **EBL**. Como se verá, aunque estos sistemas comparten las ideas fundamentales (demostrar que el ejemplo es una instancia positiva del objetivo, extraer la información relevante, generalizarla y operacionalizarla), las implementaciones son muy distintas y no siempre fáciles de describir. No pretendemos dar una visión exhaustiva del funcionamiento de estos sistemas sino simplemente mostrar cómo las ideas del **EBL** pueden ser aplicadas a diversos campos. Así, primero se estudiará STRIPS que es un planificador y, aunque es un sistema ya antiguo, tiene interés por ser el precursor del **EBL**. Después se analizará la Generalización basada en Explicaciones de Mitchell a partir de la cual surgió la nomenclatura del **EBL** como paradigma de los métodos deductivos. Finalmente se estudiarán las arquitecturas SOAR y PRODIGY como ejemplo de aprendizaje de reglas de control. El estudio de los sistemas mencionados se realizará describiendo para cada uno de ellos los mismos componentes descritos para el **EBL** (fig 4.2). Asimismo, dichos componentes serán ilustrados con el desarrollo de un ejemplo para cada sistema.

4.5.1 STRIPS

STRIPS ([**FIKE72**]) es un planificador que genera y recuerda planes que le permiten conseguir un determinado objetivo. Se considera el precursor del **EBL** porque, una vez generado un plan, intenta explicar porqué cumple el objetivo. Las entradas de STRIPS son: un estado inicial, un estado objetivo que hay que alcanzar y un conjunto de operadores que permiten cambiar de estado. Para describir los estados se utilizan fórmulas del cálculo de predicados. Los operadores están compuestos por: una precondición que describe las condiciones de aplicabilidad del operador y dos listas especificando sus efectos. Una es la lista llamada *añadir* que contiene los hechos que se añaden a un estado como consecuencia de la aplicación del operador. La otra lista se llama *borrar* y contiene los hechos que dejan de ser ciertos y que, por lo tanto, deben borrarse del estado en curso. Podemos ver un ejemplo de estas entradas en la figura 4.7 ([**FIKE72**]). En el curso de la resolución del problema se construye un plan que permite alcanzar el estado objetivo. Después este plan será convertido en un macrooperador cuya precondición describirá las condiciones suficientes bajo las cuales puede alcanzarse el estado objetivo desde el estado inicial.

El proceso seguido por STRIPS puede considerarse como *chunking* puesto que obtiene macrooperadores, pero también puede verse como la reformulación de conceptos no operacionales puesto que se trata de obtener una descripción operacional de las condiciones de aplicación de un plan que consigue un determinado objetivo.

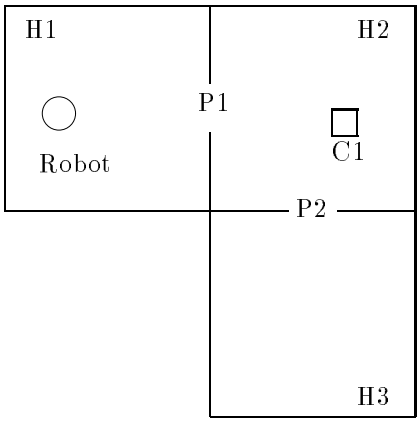
Vamos ahora a analizar los componentes de STRIPS.

Resolución del problema

El objetivo de la resolución de problemas en STRIPS es obtener un plan que permita conseguir el estado objetivo deseado desde el estado inicial. Este plan es una cadena de operadores que permiten pasar de un estado a otro hasta llegar al estado objetivo. Cada objetivo puede ser descompuesto en subobjetivos que pueden ser alcanzados por separado aplicando los operadores disponibles a los estados. Un operador es aplicable a un estado sólo si su precondición se satisface en dicho estado. El algoritmo seguido para la resolución del problema consta de los siguientes pasos:

1. Seleccionar un subobjetivo e intentar demostrar que es cierto en el estado en curso. Si es cierto ir al paso 4, en caso contrario continuar.
2. Considerar como aplicable un operador cuya lista *añadir* contenga alguna cláusula que permita concluir la demostración del paso 1.
3. La instanciación de la precondición del operador seleccionado será el nuevo subobjetivo. Ir al paso 1.
4. Si el subobjetivo conseguido es el objetivo inicial el proceso acaba, en caso contrario debe crearse un nuevo estado aplicando el operador cuya precondición es el subobjetivo que se ha establecido. Ir al paso 1.

El resultado obtenido del algoritmo anterior es una lista de operadores instanciados correspondientes a las acciones necesarias para alcanzar el estado objetivo. Veámoslo con un



Estado inicial: En-habitación(Robot, H1)
Conecta(P1, H1, H2)
Conecta(P2, H2, H3)
Caja(C1)
En-habitación(C1, H2)

Si Conecta(x, y, z) entonces Conecta(x, z, y)

Estado objetivo: Caja(x) y En-habitación(x, H1)

Operadores disponibles:

ATRAVESAR(p, h1, h2)
El robot pasa a través de la puerta p de la habitación h1 a la h2
Precondición: En-habitación(Robot, h1) y Conecta(p, h1, h2)
Lista borrar: En-habitación(Robot, *)
(se borran todas las formulas En-habitación(robot, *)
para cualquier *)
Lista añadir: En-habitación(Robot, h2)

TRASLADAR(c, p, h1, h2)
El robot traslada el objeto b de la habitación h1 a la h2
a través de la puerta p
Precondición: En-habitación(c, h1) y En-habitación(Robot, h1)
y Conecta(p, h1, h2)
Lista borrar: En-habitación(Robot, *)
En-Habitación(c, *)
Lista añadir: En-habitación(Robot, h2)
En-habitación(c, h2)

Figura 4.7: Ejemplo de entradas al sistema STRIPS.

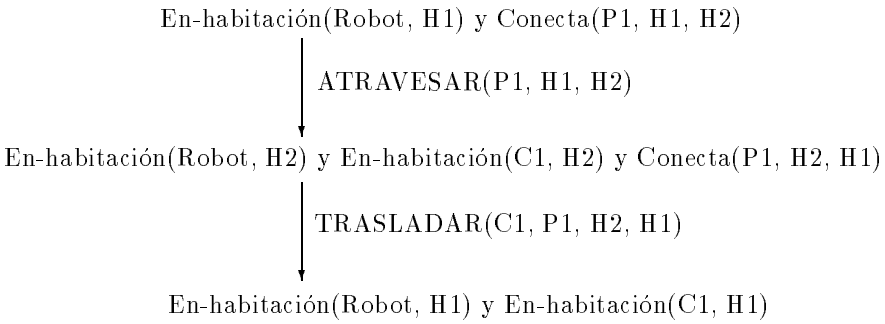


Figura 4.8: Plan obtenido por STRIPS.

ejemplo. Supongamos la situación de la figura 4.7 cuyo objetivo es conseguir `Caja(x)` y `En-habitación(x, H1)`. Al instanciar `x` en el estado inicial obtenemos `Caja(C1)` y `En-habitación(C1, H1)`. Dado que `Caja(C1)` es cierto en el estado inicial, nuestro subobjetivo es conseguir `En-habitación(C1, H1)`. Según el paso 2 del algoritmo anterior, debe buscarse un operador que tenga en su lista *añadir* una cláusula del tipo `En-habitación(x, h)`. Dicho operador es `TRASLADAR(c, p, h1, h2)` y al instanciar `c` por `C1` y `h2` por `H1` su precondición queda como sigue:

En-habitación(C1, h1) y En-habitación(Robot, h1) y Conecta(p, h1, H1)

Dado que no se ha conseguido el objetivo inicial, se debe volver al paso 1 para escoger un nuevo subobjetivo. Si se toma `En-habitación(C1, h1)` y se sustituye `h1` por `H2`, se obtiene que se cumple en el estado inicial, con lo cual quedan los siguientes dos subobjetivos:

En-habitación(Robot, H2) y Conecta(p, H2, H1)

Así, nuestro nuevo subobjetivo es conseguir ahora `En-habitación(Robot, H2)`. Un operador aplicable es `ATRAVESAR(p, h1, h2)` con lo que su precondición instanciada es la siguiente:

En-habitación(Robot, h1) y Conecta(p, h1, H2)

El nuevo subobjetivo ahora es `En-habitación(Robot, h1)`. Si se sustituye `h1` por `H1` vemos que es cierto en el modelo inicial con lo cual el subobjetivo se cumple y sólo nos queda comprobar que `Conecta(p, H1, H2)`. Si se sustituye `p` por `P1` vemos que también este subobjetivo se cumple. Todo este proceso puede resumirse en el plan de la figura 4.8, es decir, primero se aplica el operador `ATRAVESAR` al estado inicial y al estado resultante se le aplica el operador `TRASLADAR`.

1	*En-habitación(Robot, H1) En-habitación(C1, H2) *Conecta(P1, H1, H2) Conecta(P2, H2, H3) Caja(C1)	ATRAVESAR(P1, H1, H2)	
2	*Conecta(P1, H1, H2) Conecta(P2, H2, H3) Caja(C1) *En-habitación(C1, H2)	*En-habitación(Robot, H2)	TRASLADAR(C1, P1, H2, H1)
3			En-habitación(Robot, H1) En-habitación(C1, H1)
	0	1	2

Figura 4.9: Tabla triangular correspondiente al plan obtenido por STRIPS.

Análisis de la traza

Una vez se ha encontrado un plan se construye la llamada *tabla triangular* (fig 4.9) que describe la estructura del plan del robot en un formato que le será útil para generalizar las secuencias de operadores. Veámos cómo se construye dicha tabla. Para una secuencia de N operadores (dos en nuestro caso), habrá de 1 a N+1 filas y de 0 a N columnas. El operador i-ésimo se coloca en la posición (i, i) de la tabla. En la posición (i, 0) deben ponerse los hechos del modelo inicial que eran ciertos antes de la aplicación del operador i. En la posición (N+1, 0) se colocan los hechos del modelo inicial que siguen siendo ciertos después de añadir el operador. Las posiciones (i, j) restantes contienen los hechos añadidos por el operador i-ésimo que eran ciertos antes de aplicarse el operador j-ésimo. En cada celda de la columna i, fila N+1, hay que poner los hechos añadidos por el operador i-ésimo que quedan ciertos en el modelo final. Marcaremos con un asterisco los hechos de la fila j usados en la prueba de las precondiciones del operador j-ésimo. La tabla triangular es útil porque muestra cómo las precondiciones del operador dependen de los efectos del modelo inicial del mundo. Cualquier hecho marcado con un asterisco indica esta dependencia. Así, por ejemplo, en la tabla de la figura 4.9 el hecho `En-Habitación(Robot, H2)` de la columna 1 fila 2 indica que la precondición de `TRASLADAR` depende de un hecho añadido por el operador `ATRAVESAR`. Igualmente, la presencia de los hechos marcados en la columna 0, fila 2, indica que la precondición de `TRASLADAR` depende de hechos del modelo inicial.

Generalización de la información

Para generalizar sólo se tienen en cuenta aquellas cláusulas marcadas con un asterisco puesto que son las que se utilizan en las precondiciones para demostrar que un operador es aplicable. La generalización se hace en dos pasos. El primer paso consiste en substituir todas las

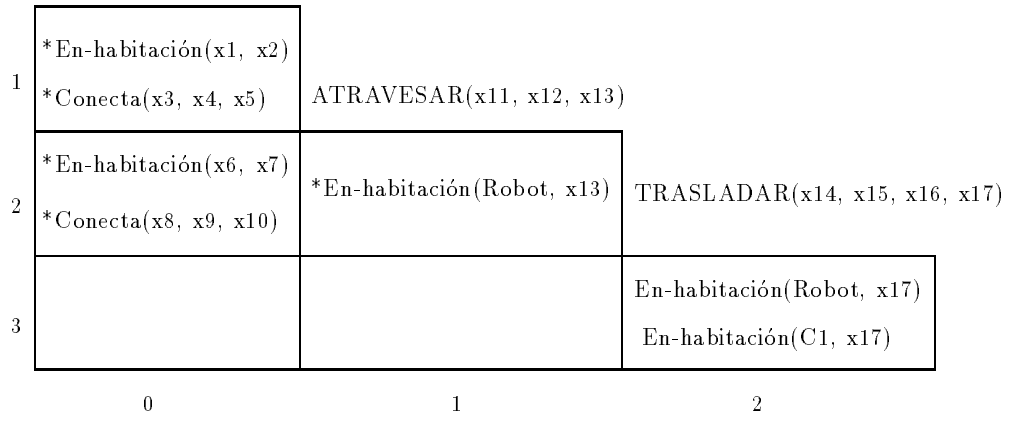


Figura 4.10: Tabla triangular sobregeneralizada.

constantes de la columna cero por variables (todas las variables deben ser distintas aunque pertenezcan a distintas ocurrencias de la misma constante). Las restantes columnas se variabilizan según la variabilización del operador. Aplicando este paso a la tabla triangular de la figura 4.9 obtenemos la de la figura 4.10 que está sobregeneralizada puesto que sería aplicable a cualquier situación. Así, pues, el segundo paso consiste en restringir la tabla para lo cual se usan dos criterios. El primero de ellos consiste en mantener las dependencias entre operadores de manera que el operador i-ésimo añadirá una cláusula soportando el operador j-ésimo en la tabla generalizada si y sólo si la misma dependencia existe entre los operadores i y j en la tabla original. El segundo criterio requiere que las precondiciones de los operadores en la tabla generalizada puedan demostrarse usando las mismas demostraciones que las que se usan para verificar las precondiciones en el plan original. Aplicando este segundo paso a la tabla de la figura 4.10 se obtiene el plan general representado en la tabla de la figura 4.11. En dicho plan vemos que el objeto a mover de una habitación a otra se ha generalizado de una caja a cualquier objeto. Las habitaciones inicial y final eran la misma en el plan original mientras que al generalizar son distintas. STRIPS también ha generalizado las condiciones de aplicabilidad de la secuencia de operadores. Las cláusulas marcadas en la columna cero de la tabla generalizada indican las condiciones generalizadas bajo las cuales la secuencia de operadores es aplicable.

Construcción de nueva información

El siguiente paso consiste en crear un macrooperador a partir de la tabla triangular generalizada de la figura 4.11. Este macrooperador tendría el mismo efecto en un solo paso que la aplicación de la secuencia de operadores encontrada en la resolución del problema (fig. 4.8). Su precondición serían las cláusulas de la columna cero marcadas con un asterisco. No obstante, STRIPS no construye físicamente este macrooperador sino que guarda la tabla triangular generalizada. Así, una vez comprueba que se cumple la precondición, los operadores que forman el macrooperador son aplicados uno a uno.

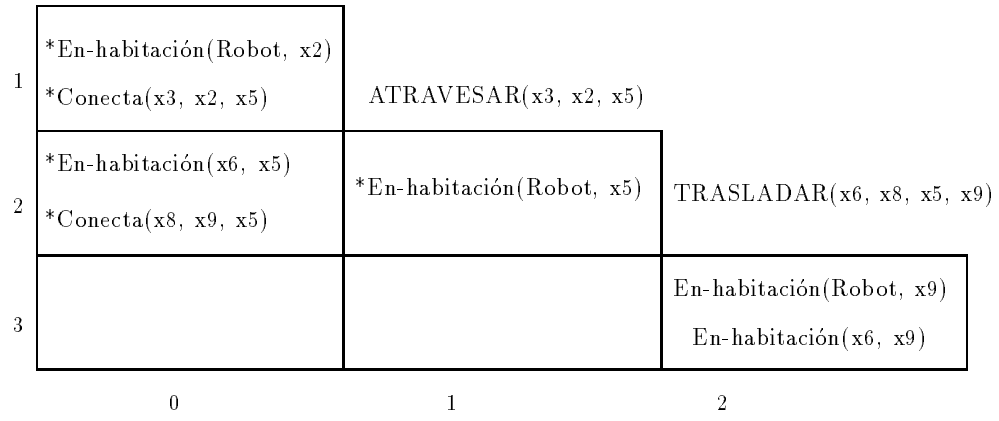


Figura 4.11: Otra tabla triangular generalizada.

4.5.2 EBG de Mitchell

La generalización basada en explicaciones o *Explanation-based Generalization* (EBG) es un formalismo propuesto por Mitchell y sus colaboradores en 1986 **[MITC86]**. Este formalismo es un intento de agrupar elementos esenciales de sistemas ya existentes. El propio Mitchell describe el EBG como un método independiente del dominio que usa conocimiento del dominio para guiar la generalización. Esto es, los mecanismos de resolución de problemas y de generalización, como se verá, son independientes del dominio mientras que las entradas al sistema forzosamente deben contener información del dominio. El objetivo del **EBG** es aprender nuevas descripciones operacionales de conceptos. Para ello necesita la siguiente información de entrada:

Concepto objetivo: Descripción no operacional del concepto a aprender.

Ejemplo: Instancia positiva del concepto objetivo.

Teoría del dominio: Conjunto de reglas y hechos usados para explicar cómo el ejemplo satisface el concepto objetivo.

Criterio de operacionalidad: Predicados sobre las definiciones de conceptos que especifican cómo debe expresarse el concepto objetivo.

El resultado es una nueva teoría del dominio a la que se habrá incorporado la nueva definición operacional obtenida. Esta nueva descripción es la generalización del ejemplo de manera que es una condición suficiente para el objetivo que además satisface el criterio de operacionalidad.

Así, dados un concepto objetivo y un ejemplo, el EBG debe demostrar primero que el ejemplo es una instancia positiva del concepto objetivo usando para ello la teoría del dominio. La traza obtenida contiene todos los caminos intentados y que han fracasado y un solo camino hacia la

solución. Este camino hacia la solución es el que constituye la explicación que posteriormente debe ser generalizada. El método de generalización utilizado es una modificación del algoritmo de regresión de objetivos consistente en usar sólo las reglas que han servido para demostrar el ejemplo de manera que la generalización es una condición suficiente bajo la que una regla R puede ser usada para inferir una fórmula F. A partir de la generalización obtenida se construye una sola regla operacional que será incorporada a la teoría del dominio existente. La diferencia de funcionamiento entre el EBG y los métodos **EBL** en general reside sólo en la generalización. Esta diferencia es importante puesto que es lo que hace que a partir del EBG se obtenga una sola regla operacional nueva y no varias como ocurre en el **EBL**. Describiremos el EBG desarrollando el mismo ejemplo que en la sección 4.3.2, de esta manera podremos compararlos. Así, dadas las entradas de la figura 4.3 al resolver el problema obtenemos la traza de la figura 4.4. Al analizar dicha traza obtenemos la explicación de la figura 4.5. Vamos a continuar la descripción del EBG de Mitchell a partir de la tarea de generalización de la explicación.

Generalización de la explicación

Para generalizar la explicación se sigue el mismo procedimiento que el explicado en general para los métodos **EBL**. La diferencia es que aquí sólo se utilizan las reglas que han contribuído a alcanzar la solución. Así, para generalizar TIGRE(Bengala), se usa la explicación (figura 4.5) y se substituye Bengala por x. Esta substitución se propaga usando la regla R1 de la teoría. El siguiente predicado a generalizar es FELINO(Bengala). En la teoría hay dos reglas (R2 y R3) que concluyen FELINO(x), pero en la demostración del ejemplo se ha utilizado R3, por lo tanto es ésta la que se utiliza para generalizar la explicación. Para generalizar CARNIVORO y MAMIFERO se usan las reglas R5 y R7 respectivamente que son las que se han empleado para demostrar el ejemplo. Así en la figura 4.12 podemos ver como queda la generalización de la explicación en el caso del EBG (Compárense las figuras 4.6 y 4.12).

Construcción de nueva información

La nueva información se construye de la misma manera explicada para los métodos **EBL**. Es decir, las hojas del árbol de explicación generalizado que cumplen el criterio de operacionalidad, constituirán las condiciones de la nueva regla, cuya acción será la raíz del árbol. En este caso, se obtiene una única regla que es la siguiente:

SI: RAYADO(x) y DIENTES (x) y DA-LECHE(x) y TIENE-RABO(x)

ENTONCES: TIGRE(X)

4.5.3 SOAR

SOAR ([**LAIR86a**]) es una arquitectura que combina aprendizaje y resolución de problemas. La unidad organizativa fundamental de SOAR es el espacio de problemas y su paradigma central es la búsqueda. La arquitectura de SOAR tiene cinco componentes básicos:

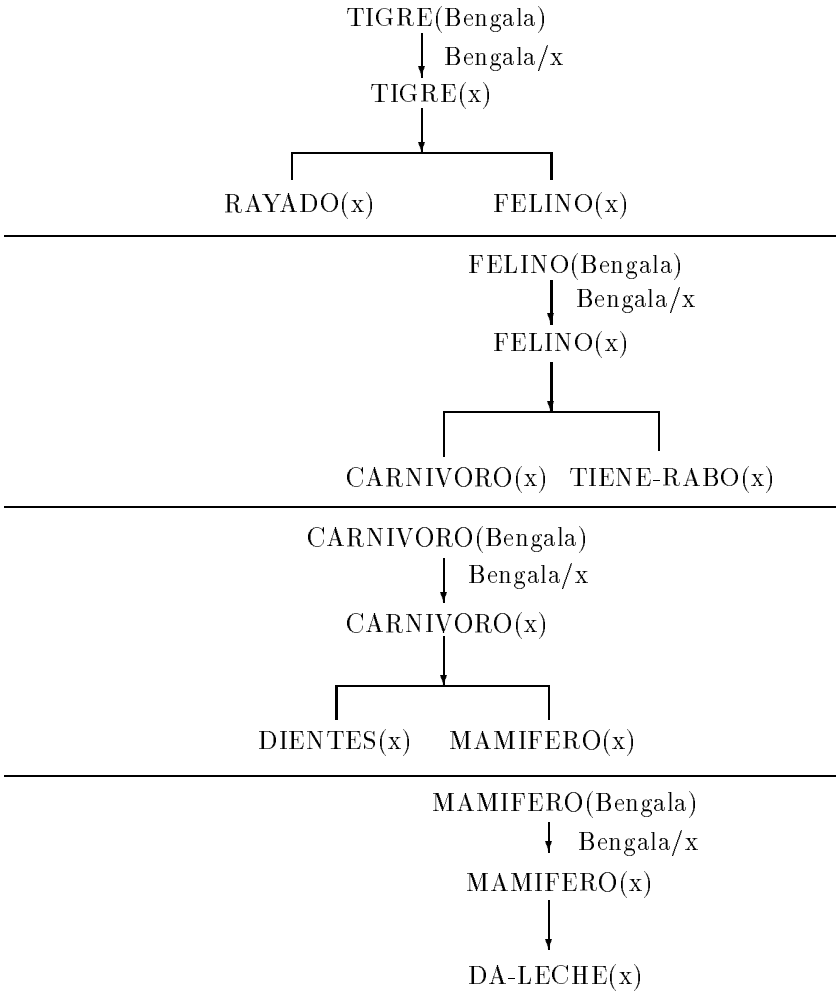


Figura 4.12: Explicación generada con el algoritmo de regresión de objetivos modificado.

- Una *memoria de reglas de producción* que contiene la experiencia acumulada por el sistema en la resolución de problemas
- Una *memoria de trabajo* que contiene la información relacionada con el problema que se está intentando resolver
- Un *gestor de la memoria de trabajo* que le permite borrar objetos de la memoria de trabajo
- Un *procedimiento de decisión* que le permite resolver conflictos
- Un *mecanismo de chunking* que le permite aprender.

La memoria de trabajo se compone de tres tipos de objetos: 1) una pila de contextos que especifican la jerarquía de objetivos activos, espacio de problemas, estados y operadores; 2) objetos tales como objetivos y estados; y 3) preferencias que codifican el conocimiento de control.

El objetivo principal de SOAR es resolver problemas y el método utilizado para ello es la subobjetivación universal (*universal subgoaling*). Así, para conseguir un objetivo, entendiendo como tal un conjunto de estados deseado, se realiza una búsqueda en un espacio de problemas. El funcionamiento de SOAR es similar a un ciclo de resolución de problemas general, es decir, primero busca los operadores que pueden ser aplicables al estado en curso y después escoge entre ellos. SOAR puede trabajar en varios espacios de problemas y, por tanto, para decidir los operadores aplicables a un determinado estado, debe saber primero en qué contextos buscarlos. Un contexto es un objeto que se compone de cuatro ítems: un objetivo, un espacio de problemas, un estado y un operador. Así, un contexto determina cuándo y dónde puede ser aplicable un operador para conseguir un determinado objetivo. Además, un contexto puede vincularse a contextos previos formando una jerarquía de objetivos y subobjetivos. Las componentes de cada contexto son anotaciones con información adicional llamadas *aumentos*. La jerarquía de contextos y sus aumentos asociados constituyen la memoria de trabajo de SOAR. La resolución de conflictos de la resolución de problemas general se ha substituído en SOAR por un ciclo de control compuesto por dos fases, una de elaboración y otra de decisión. En la *fase de elaboración* se activan en paralelo los operadores aplicables. En la *fase de decisión* se examinan los resultados de los operadores aplicados en la fase de elaboración y se escoge la mejor opción para un ítem del contexto basándose en ellos y en la semántica de las relaciones de preferencia. A partir de la opción elegida se construye una preferencia que será aplicable en una situación similar. Cuando en esta fase de decisión no ha sido posible decidir cuál es el mejor operador aplicable, o bien ninguno de los encontrados ha sido considerado adecuado, entonces se produce un *impásse*. Un *impásse* significa que el sistema no tiene suficiente información por lo que es una oportunidad para aprender nuevo conocimiento de control (preferencias) de manera que en una futura situación similar ese *impásse* pueda evitarse. Existen los siguientes cuatro tipos de *impasses*:

De vínculo (*tie impasse*): Se produce cuando hay varios operadores posibles y hay poco conocimiento para discriminarlos.

Conflicto: Hay varias opciones conflictivas en el sentido de que pueden llevar a estados contradictorios.

Sin cambio: Se mantiene sin cambio el valor del ítem sobre el que se ha producido el *impás*.

Rechazo: La opción en curso es rechazada y no hay ninguna opción más ya que se han probado todas las existentes.

El **EBL** se realiza al generalizar las situaciones en las que se puede usar una preferencia determinada.

Simplificando mucho su funcionamiento, podríamos decir que SOAR realiza el ciclo de control de la figura 4.2 para cada subobjetivo alcanzado. La nueva información construída son preferencias que le permiten evitar *impasses* con todo lo que ello significa (activación paralela de todos los operadores aplicables). Dada la complejidad de la arquitectura, no daremos más detalles sobre ella. Si se desea más información ver **[LAIR86a]** o **[LAIR86b]**. Vamos ahora a describir sus componentes.

Resolución del problema

Los métodos de resolución de problemas usados se basan en la subobjetivación universal y en el método débil universal (**[LAIR86b]**). La subobjetivación universal es una técnica para hacer todas las decisiones de control de manera uniforme. El método débil permite al sistema representar las funciones básicas de resolución de problemas. La estrategia de aprendizaje de SOAR se basa en la técnica de *chunking* de secuencias de reglas de producción desarrollada por Rosenbloom y Newell. Los autores de SOAR han dado las hipótesis de que el *chunking* es el método de aprendizaje universal y de que las técnicas de *chunking* son especialmente potentes cuando se combinan con los métodos de subobjetivación y débiles.

La resolución de problemas es la tarea más importante de SOAR y dentro de ella se realiza el **EBL**. Nuestro objetivo es describir el **EBL** por lo que vamos a simplificar al máximo la explicación de la tarea de resolución de problemas que, como ya hemos dicho, se realiza utilizando un ciclo elaboración-decisión.

Las entradas son la descripción de un estado inicial y del estado objetivo a conseguir. Vamos a ilustrar el funcionamiento de la resolución de problemas en SOAR intentando resolver el problema del puzzle (el mismo que puede encontrarse en **[LAIR86a]**) cuyas entradas son las de la figura 4.13. Vamos a suponer que hay cuatro operadores que describen el movimiento de la casilla en blanco: ARRIBA, ABAJO, IZQUIERDA y DERECHA. Aunque aquí para simplificar no hablaremos del espacio de problemas, hay que tener en cuenta que cada estado está asociado a un espacio de problemas, por lo que lo primero que habría que hacer sería buscar un espacio de problemas adecuado en el que se pudiera conseguir el estado objetivo deseado.

Una vez en el estado inicial, en la fase de elaboración se descubre que podemos aplicar tres operadores de manera que la casilla en blanco puede moverse hacia arriba, a la izquierda o a la derecha. No hay ninguna preferencia de cuál es mejor por lo tanto se activan los tres en paralelo obteniéndose tres estados distintos. En la fase de decisión se considera que un estado es mejor que otro si tiene mayor número de fichas colocadas en su posición correcta respecto del estado objetivo. En nuestro caso, esto ocurre cuando se utiliza el operador ARRIBA. En este momento es cuando se crearía una preferencia con lo cual pasaríamos a las tareas de análisis y generalización para encontrar la información relevante y generalizarla. Una vez hecho esto se

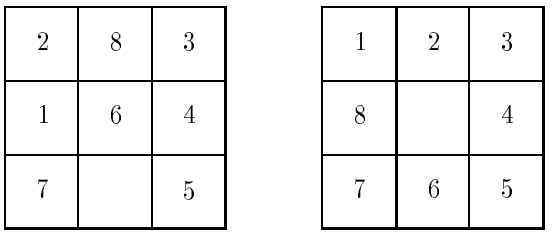


Figura 4.13: Ejemplo de funcionamiento de SOAR. Entradas.

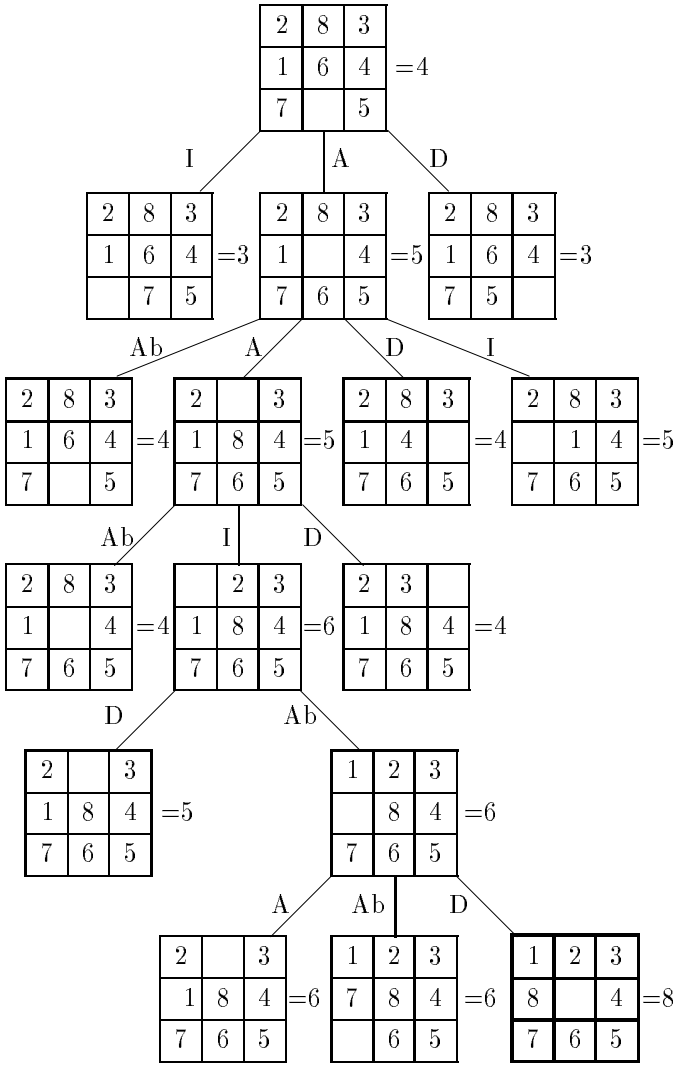
aplicaría el operador ARRIBA al estado inicial y, a partir del estado resultante, deberíamos encontrar un nuevo operador aplicable y así sucesivamente hasta llegar al estado objetivo con lo que se obtiene la traza de la figura 4.14. Los nodos de cada nivel se activan en paralelo y para el operador elegido en cada nivel (subobjetivo) se crea una preferencia. Podemos observar que en el nivel 3 podemos escoger dos operadores, IZQUIERDA y ARRIBA, y la fase de decisión no tiene suficiente conocimiento para discriminar entre ellos con lo cual se produce un *impás*. En este caso, y dado que *a priori* ambos estados son igualmente buenos, elige de forma aleatoria el siguiente operador a aplicar. De esta manera se van expandiendo los nodos hasta llegar al estado objetivo.

Análisis de la traza

Una vez conseguido un objetivo, la tarea de análisis debe identificar la información relacionada sólo con ese subobjetivo con la que se construirá la explicación a generalizar. La información relevante que nos servirá para construir un nuevo macrooperador son aquellos elementos de la memoria de trabajo que se han referenciado o creado durante la consecución de un subobjetivo. Así, las condiciones del nuevo macrooperador serán los elementos de la memoria de trabajo existentes antes del establecimiento del subobjetivo y que se han referenciado durante el proceso de consecución del subobjetivo. Las acciones serán los elementos que se han creado durante el proceso y que se utilizarán para el siguiente subobjetivo. No daremos aquí ningún ejemplo puesto que habría que tener en cuenta cómo ha quedado la pila de contextos, todos los *impasses* generados y todos los elementos creados y esto complicaría innecesariamente la descripción de la tarea. Para más detalles ver **[LAIR86a]** o **[LAIR86b]**.

Generalización de la explicación

La generalización consiste en substituir constantes por variables pero imponiendo tres condiciones: 1) el mismo identificador debe ser substituído siempre por la misma variable, 2) distintos identificadores deben ser substituídos por variables distintas y 3) distintas variables siempre corresponden a distintos identificadores. Estas tres condiciones producen una nueva regla que no está sobregeneralizada sino al contrario, normalmente está demasiado especial-



Operadores Aplicables

Ab: ABAJO
A: ARRIBA
I: IZQUIERDA
D: DERECHA

Figura 4.14: Ejemplo de funcionamiento de SOAR. Traza.

izada. Este proceso sería equivalente al realizado por STRIPS puesto que se obtendría una regla de control que nos describiría situaciones en las que se puede aplicar un determinado operador. En nuestro ejemplo, se obtendrían preferencias del tipo "siempre que se quiera mover el espacio en blanco de una casilla i a otra j es conveniente usar el operador OP" o bien macrooperadores que indicarían los movimientos a seguir.

Construcción de nueva información

A partir de la explicación generalizada debemos construir un macrooperador el cual es optimizado antes de incorporarlo a la memoria de producciones. Con el nuevo macrooperador se obtiene nuevo conocimiento de control creando nuevas reglas de producción (preferencias) que le ayudan a tomar decisiones más fácilmente. Las nuevas reglas permiten a SOAR tomar decisiones directamente a través de la fase de elaboración y de decisión descritas anteriormente de manera que se producen menos *impasses* evitando así la necesidad de subobjetivación.

Optimización de la nueva información

La optimización consta de dos fases. En la primera se elimina información equivalente y duplicada y en la segunda se reorganizan las condiciones del macrooperador de manera que se reduzca el coste de cotejarlas. El macrooperador que puede construirse como resultado de la obtención de un subobjetivo tendrá muchos elementos comunes con los macrooperadores asociados a estados anteriores al que se ha producido el *impás* del que ha salido el subobjetivo en curso. De hecho, muchas veces la diferencia son los nombres de las variables y que el nuevo macrooperador tiene algunas condiciones más. Así, en la primera fase de la optimización se borran las condiciones ya existentes en macrooperadores anteriores. En la segunda fase debemos reducir su coste de cotejamiento reorganizando las condiciones del nuevo macrooperador. Dado que cada condición del macrooperador actúa como una consulta y que retorna todos los elementos de la memoria de trabajo comparables con la condición, si primero se hacen las consultas más restrictivas y que, por tanto, tienen más probabilidad de fallar, puede que no sea necesario explorar todas las condiciones obteniendo así más eficiencia.

4.5.4 PRODIGY

PRODIGY ([**MINT88**]) es una arquitectura integrada que intenta unificar resolución de problemas, planificación y múltiples métodos de aprendizaje (**EBL**, analogía derivacional, abstracción de planes y experimentación). El núcleo central de PRODIGY es un resolvedor de problemas general cuyo comportamiento está determinado por el conocimiento del dominio (objetivos, relaciones, operadores y reglas de inferencia) y por el conocimiento de control para dirigir la búsqueda. Se entiende por conocimiento de control aquella información que permite reducir la búsqueda modificando la estrategia usada por defecto (en el caso de PRODIGY, la búsqueda es en profundidad). En PRODIGY hay varios tipos de conocimiento de control: reglas, funciones de evaluación heurísticas, planes abstractos, soluciones de problemas análogos (en una librería) y macrooperadores. El módulo **EBL** de PRODIGY analiza la traza de resolución del problema para extraer la información relevante a partir de la que construirá una explicación. A partir de esta explicación se obtendrá una regla de control que será poste-

riormente simplificada y cuya utilidad será evaluada. En función de la utilidad estimada será incorporada o no al conocimiento existente.

Para resolver problemas en un dominio particular, PRODIGY necesita la especificación de este dominio en forma de un conjunto de operadores y de reglas de inferencia. Un operador se compone de una precondición que determina las condiciones bajo las que es aplicable y de una lista de los efectos que produce su aplicación sobre el estado en curso. Las reglas de inferencia tienen la misma especificación que los operadores, es decir, una precondición que debe ser cierta para que la regla sea aplicable y una lista de los efectos que produce su aplicación. La diferencia entre los operadores y las reglas de inferencia estriba en que los operadores corresponden a acciones externas que permiten pasar de un estado a otro, mientras que las reglas de inferencia incrementan el conocimiento explícito del estado en curso puesto que añaden fórmulas a la descripción del estado pero no pueden borrar. PRODIGY separa el conocimiento del dominio que especifica los operadores y reglas de dominio disponibles, del conocimiento de control que describe cómo resolver problemas en el dominio. El **EBL** en PRODIGY ha sido diseñado para evitar los problemas de degradación mencionados en la sección 4.4.1, intentando que el conocimiento de control aprendido sea efectivo y útil. Para ello tiene especificados declarativamente los siguientes cuatro conceptos objetivo (*target concept*), cada uno de los cuales existe para nodos, objetivos, operadores y vínculos (instanciaciones):

Éxito. Una decisión de control tiene éxito si lleva a la solución.

Fracaso. Una decisión falla si no hay ninguna solución consistente con la decisión.

Única alternativa. Si los demás candidatos han fallado.

Interferencia de objetivos. Si todas las soluciones consistentes con esta decisión borran una condición que tendrá que volver a conseguirse.

PRODIGY puede crear tres tipos de reglas de control (selección, rechazo o preferencia), cada una de las cuales está relacionada con un tipo de concepto objetivo (ver figura 4.15). Gracias a estos conceptos objetivo, PRODIGY puede no sólo explicar porqué un camino ha tenido éxito sino también porqué ha fallado o porqué tiene ciertas propiedades como que es más corto que otro.

Una vez introducidos todos los elementos necesarios, vamos ahora a describir las componentes de PRODIGY.

Resolución del problema

Dado un estado objetivo, un estado inicial, una teoría del dominio y una teoría de control, PRODIGY debe obtener una plan para conseguir el estado objetivo a partir del estado inicial. La solución es una secuencia de operadores instanciados que, aplicados al estado inicial, nos dan el estado objetivo. El método de resolución de problemas construye un árbol de búsqueda a partir de un nodo que contiene el estado inicial y el objetivo que se quiere conseguir. Este árbol es expandido en un ciclo que se compone de dos fases:

CONCEPTOS OBJETIVO	REGLA CONTROL
Éxito	Preferencia
Fracaso	Rechazo
Única alternativa	Selección
Interferencia de objetivos	Preferencia

Figura 4.15: Conceptos objetivo de PRODIGY y reglas de control asociadas.

Fase de decisión. En esta fase deben tomarse cuatro tipos de decisiones. La primera es cuál es el siguiente nodo a expandir (por defecto se hace en profundidad). Cada nodo es un conjunto de objetivos y un estado describiendo el mundo, por lo que la siguiente decisión es cuál de los objetivos del nuevo nodo hay que alcanzar. Una vez seleccionado un objetivo debemos seleccionar un operador aplicable. Finalmente deben decidirse los vínculos de los parametros de los operadores (instanciaciones).

Fase de expansión. Si el operador obtenido en la fase anterior ha podido instanciarse completamente entonces se aplica al estado en curso. En caso contrario se crea un nuevo nodo cuyos objetivos serán los mismos que los del padre a los cuáles se habrán añadido las precondiciones no instanciadas.

Este ciclo se acaba cuando se crea un nodo que satisface el estado objetivo. Para tomar una decisión de control dado un conjunto de candidatos (nodos, objetivos, operadores o vínculos, según la decisión), primero se aplican las reglas de selección obteniéndose así un subconjunto de operadores (si no hay reglas de selección se obtienen todos los operadores aplicables). Después se aplican las reglas de rechazo que eliminan algunos de los candidatos. Finalmente se aplican las reglas de preferencia. En caso de fallar el operador seleccionado, se vuelve atrás hasta el punto de decisión anterior y se escoge el siguiente candidato preferido y así sucesivamente hasta agotar la lista de candidatos o hasta encontrar uno que funcione. Las reglas de control se aplican en la fase de decisión ayudando a incrementar la eficiencia de la resolución, mejorar la calidad de la solución encontrada y dirigir la resolución hacia caminos que de otra manera quizá no serían explorados. Veamos la resolución del problema con un ejemplo que es la simplificación del descrito en **[MINT90]**. Supongamos que tenemos el dominio de las máquinas que permiten dar forma a objetos (figura 4.16) y tenemos en el estado inicial un objeto A que no está pulido, está frio y tiene forma oblonga. Nuestro objetivo es conseguir que tenga forma cilíndrica y que esté pulido. El primer paso es crear un nodo, que será la raíz del árbol de búsqueda y que contiene los estados inicial y objetivo (véase figura 4.17). Para expandir el árbol debemos decidir qué subobjetivo queremos alcanzar primero. Dado que no tenemos reglas de control, escogemos primero el hacer que el objeto A adquiera forma cilíndrica. La aplicación del operador LAMINAR nos hace cambiar el estado con lo que A tendrá forma cilíndrica y estará caliente. El siguiente subobjetivo será ahora pulirlo. Para ello primero debemos cogerlo (precondición del operador PULIR) pero como no sabemos si puede cogerse, crearemos un nuevo nodo (el número 3) que tendrá como subobjetivo coger

```
operador: PULIR (obj)
    precondiciones: (es-un-objeto obj)
                    (o (se-puede-coger obj pulidora)
                       (forma obj rectangular))
    efectos: (borrar (condiciones-superficie obj cond))
             (añadir (condiciones-superfície (obj pulido)))

operador: LAMINAR (obj)
    precondiciones: (es-un-objeto obj)
    efectos: (borrar (forma obj forma1))
             (borrar (temperatura obj temp))
             (borrar (condiciones-superfície obj condiciones))
             (borrar (pintado obj))
             (añadir (temperatura obj caliente))
             (añadir (forma obj cilíndrica))

operador: TORNEAR (obj forma)
    precondiciones: (es-un-objeto obj)
                                (forma obj forma1)
    efectos: (borrar (forma obj forma1))
             (borrar (condiciones-superfície obj condiciones))
             (borrar (pintado obj))
             (añadir (forma obj cilíndrica))
             (añadir (condiciones-superfície obj rugoso))


regla de inferencia: SE-PUEDE-COGER
    parámetros: obj maquina
    precondiciones: (puede-coger máquina)
                    (temperatura obj frío)
    efectos: (añadir (se-puede-coger obj máquina))
```

Figura 4.16: Ejemplo de representación de operadores y reglas de inferencia en PRODIGY.

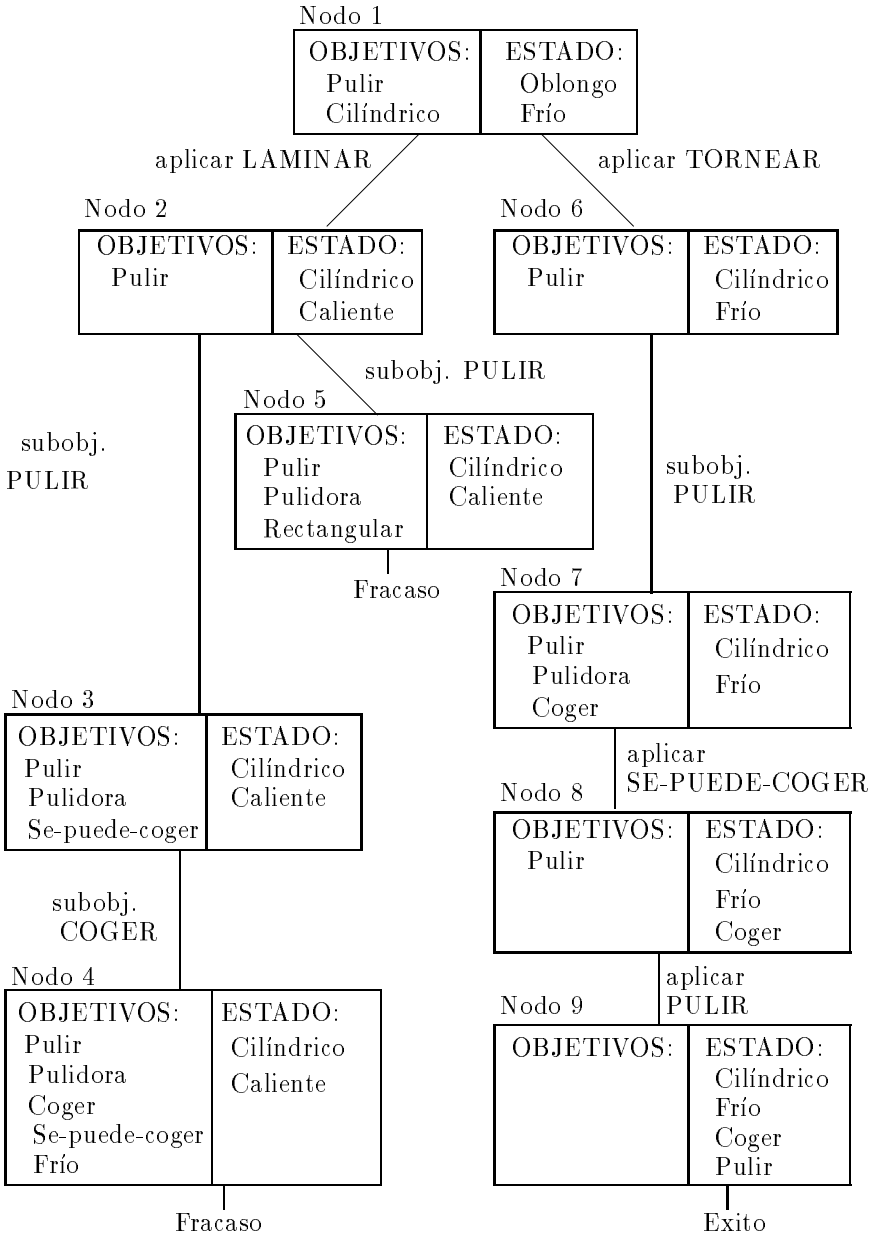


Figura 4.17: Traza producida por PRODIGY al resolver un problema.

el objeto. Vemos en la precondición de COGER que el objeto debe estar frío para poder cogerlo, cosa que no se cumple en el estado en curso, por lo tanto falla. Debemos volver al último punto de decisión (el nodo número 2) y reconsiderar la decisión correspondiente. Para ello debemos hacer que la forma del objeto sea rectangular pero esto no se cumple puesto que A es cilíndrico y hacerlo rectangular desharía el primer subobjetivo conseguido. Esto nos lleva a reconsiderar la primera decisión, es decir, en lugar de intentar alcanzar el subobjetivo cilíndrico usando el operador LAMINAR, intentaremos conseguirlo usando TORNEAR con lo cual el objeto queda frío y se puede pulir. Así, el plan primero es hacerlo cilíndrico usando el torno y después pulirlo con la pulidora. La traza obtenida en la resolución del problema es la de la figura 4.17.

Análisis de la traza

La traza formada por el árbol de búsqueda es analizada intentando extraer nodos que puedan usarse como ejemplos de entrenamiento. Dado que puede haber varios ejemplos de entrenamiento para un mismo concepto objetivo, éstos tienen unas listas de especificación que contienen heurísticas de selección que se usan para escoger los ejemplos más prometedores para producir reglas de control. Estas heurísticas de selección son independientes del dominio y específicas para cada tipo de concepto objetivo. Una vez seleccionada la información relevante, debe construirse la explicación usando la teoría del dominio. El árbol de búsqueda puede ser muy grande con lo cual se tardaría mucho en explorarlo entero. Es por esto que se ponen restricciones de tiempo se manera que sólo se analiza una parte. Analicemos ahora la traza producida en nuestro ejemplo. El nodo 4 es un ejemplo del concepto objetivo *fracaso* y el 9 de *éxito*. El nodo 4 ha fallado debido a que la temperatura de A no es fría. Propagando al nodo 3 vemos que esto hace que no se cumpla el objetivo de coger A. En el nodo 5, en cambio, el fracaso se ha producido porque no se cumple que A tenga forma rectangular. Así, el fracaso del nodo 2 se produce porque el objeto A ni está frío ni es rectangular. Todo ésto nos lleva al fracaso del operador aplicado sobre el nodo 1 debido a que la forma que consigue es incompatible con el siguiente objetivo. Este fracaso lleva a la construcción de una regla de rechazo como la siguiente:

SI: (NODO-ACTUAL nodo) y
(OBJETIVO-ACTUAL nodo (FORMA objeto forma)) y
(OPERADOR-CANDIDATO nodo LAMINAR) y
(OBJETIVO-ADJUNTO nodo (PULIR objeto))

ENTONCES: (Rechazar operador LAMINAR)

De la misma forma se procedería a partir del nodo 9 para conseguir en este caso una regla de preferencia.

Generalización de la explicación

Para generalizar la explicación, PRODIGY usa dos tipos de axiomas: los de la arquitectura y los del dominio. Los *axiomas de la arquitectura* contienen definiciones de conceptos en términos

de esquemas de dominio más primitivos que sirven como teoría en la resolución del problema. Los *axiomas del dominio* describen conceptos en términos de efectos y precondiciones de operadores, reglas de inferencia y reglas de control del dominio. Todos los axiomas se representan como reglas. También se utilizan dos tipos de conceptos: los primitivos y los definidos. Los *conceptos primitivos* son aquellos directamente observables o de mundo cerrado y pueden ser añadidos o borrados por los operadores. Los *conceptos definidos* son inferidos bajo demanda usando reglas de inferencia y representan abstracciones útiles en el dominio permitiendo que las precondiciones de los operadores se expresen más concisamente. El algoritmo utilizado se denomina Especialización basada en explicaciones o *Explanation-based Specialization* (EBS) que, como su nombre indica, no generaliza la explicación sino que la especializa en el sentido de que todos los conceptos definidos son reescritos en funcion de conceptos primitivos. El algoritmo EBS (figura 4.18) recupera axiomas que implican el concepto y recursivamente especializa los conceptos no primitivos. Dado que puede haber varios axiomas disponibles, existen unas *funciones de discriminación* que le permiten decidir cuál de ellos es el más apropiado. El resultado es una descripción plenamente especializada del concepto.

Un concepto está representado por una fórmula atómica.
Para especializarlo hacemos:

1. Si el concepto es primitivo (ningún axioma lo implica) se retorna tal como está.
2. Llamar a la función discriminante asociada al concepto para recuperar un axioma consistente con el ejemplo.
   Cada fórmula atómica no negada del axioma es un subconcepto.
   Mientras haya subconceptos en el axioma que no hayan sido especializados hacer:
   - 2.1 Especializar el subconcepto
   - 2.2 Renombrar las variables en la descripción especializada para evitar conflictos de nombres
   - 2.3 Substituir la descripción especializada del subconcepto en el axioma y simplificar

Figura 4.18: Algoritmo EBS.

Optimizacion de la nueva información

Basicamente la optimización se realiza para reducir los costes de cotejar las reglas. Para ello se utiliza un *módulo de compresión* que primero usa evaluación parcial y transformaciones lógicas simples y después, mediante un demostrador de teoremas que usa axiomas de simplificación dependientes del dominio, puede simplificar la descripción aprendida. PRODIGY intenta combinar las descripciones aprendidas que comparten el mismo concepto objetivo. Este proceso de optimización puede verse como una búsqueda a través del espacio de formulaciones alternativas de una explicación para encontrar la formulación que sea más eficiente al

cotejarla.

Construcción de nueva información

Cada concepto objetivo tiene en su especificación una plantilla (*template*) que le permite construir el tipo de regla de control asociada a él. Así, la descripción aprendida es insertada en dicha plantilla.

Evaluación de la utilidad de la nueva información

PRODIGY tiene una *métrica de utilidad* específica para evaluar las reglas de control y medir la relación entre búsqueda y conocimiento. La utilidad de una regla se obtiene comparando el coste de cotejar la regla con el coste de explorar la porción del árbol de búsqueda que no es necesario considerar con ella. Cuando se aprende una regla a partir de un ejemplo, los costes y beneficios para esta regla pueden estimarse a partir del ejemplo y ser validados durante la resolución de problemas manteniendo estadísticas sobre su uso. Sólo se guardan las reglas con alta utilidad. Si en algún momento se estima utilidad negativa para una regla entonces será eliminada. Para la estimación de la utilidad se usa la fórmula

$$\text{Utilidad} = (\text{Ahorro} * \text{Frecuencia}) - \text{Coste-cotejo}$$

donde *Ahorro* es el ahorro medio si se aplica la regla; *Frecuencia* representa las veces que la regla es comprobada y aplicada y *Coste-cotejo* es el coste medio de cotejar la regla.

4.6 Comparación con otros métodos de aprendizaje

Los métodos deductivos necesitan una teoría del dominio completa y consistente para funcionar bien. Esto significa que en dominios del mundo real muchas veces será inaplicable (pensemos por ejemplo en dominios médicos o legales donde no hay una teoría del dominio bien definida).

Los métodos inductivos son *a priori* mejores candidatos para tratar con el mundo real pero presentan varios inconvenientes. El primero es que los ejemplos de entrada deben ser suficientemente representativos como para asegurar un buen aprendizaje. Otro inconveniente es que hay que dar las características relevantes de los conceptos, tarea no siempre fácil de decidir. Y otro es que las generalizaciones no están justificadas. Todos estos inconvenientes son salvados por los métodos deductivos ya que la información relevante es extraída del ejemplo de entrada y si los ejemplos de entrada son escogidos el aprendizaje es mejor pero ello no es imprescindible.

No obstante, los métodos **EBL** presentan un problema de fondo y es que hay autores que no lo consideran como un método de aprendizaje puesto que no aprenden nuevo conocimiento sino que explicitan conocimiento que ya tenían de forma implícita. Otros autores defienden que sí realiza aprendizaje en el sentido de que la respuesta del sistema puede ser distinta (y no solo más eficiente) en función de la experiencia que haya ido incorporando. De hecho, muchas veces para las personas, el hecho de explicitar conocimiento a partir de otro conocimiento

puede no ser trivial. En cualquier caso, esta explicitación de conocimiento implícito puede darnos herramientas para mejorar la eficiencia de un sistema de resolución de problemas.

Otra limitación del **EBL** es que sólo pueden producirse reglas correctas según la teoría del dominio e independientemente del contexto y muchas veces podría interesar aprender reglas que, aunque no son lógicamente correctas sí lo son heurísticamente. Las reglas heurísticas constituyen conjeturas plausibles que son útiles en algunos contextos pero no válidas en otros.

Una diferencia interesante entre los métodos inductivos y deductivos es que en los métodos inductivos se generaliza a partir de los ejemplos usando las características contenidas en éstos mientras que en los métodos deductivos lo que se generaliza son los caminos deductivos que llevan a la solución de un problema, de manera que se obtienen situaciones generales para las que se puede aplicar la misma forma de solucionar el problema.

De lo dicho hasta el momento podemos concluir que para aplicaciones del mundo real será bueno intentar integrar los aprendizajes inductivo y deductivo de manera que podamos aprovechar las ventajas que nos da cada uno de ellos al mismo tiempo que evitamos algunos de los respectivos problemas. En este sentido se ha abierto una nueva línea de investigación y ya existen algunos sistemas como UNIMEM que integran **EBL** y **SBL** (aprendizaje basado en similaridades o *Similarity-based Learning*) [**LEBO86**]. En general, la integración de métodos deductivos e inductivos puede permitir completar una teoría mediante la utilización de casos que el sistema haya resuelto anteriormente. Asimismo, a partir de los casos se puede obtener una nueva teoría del dominio.

4.7 Conclusiones

El aprendizaje deductivo tiene como paradigma básico el aprendizaje basado en explicaciones o **EBL**. Este tipo de aprendizaje pretende capturar el tipo de aprendizaje humano mediante el cual con un solo ejemplo de entrada y con gran cantidad de experiencia (teoría) de un dominio somos capaces de generalizar. El **EBL** tiene como entradas una teoría del dominio completa, un ejemplo y un concepto objetivo. El resultado es la mejora de la teoría del dominio inicial que normalmente consiste en la incorporación de una nueva definición operacional del concepto objetivo, un plan más eficiente o una nueva heurística. La ventaja de los métodos deductivos frente a los inductivos es que las generalizaciones propuestas siempre son correctas. El principal inconveniente es la gran cantidad de conocimiento de dominio que se necesita. Además, dicha teoría debe ser completa pues en caso contrario no se garantiza el comportamiento del sistema en los nuevos problemas.

Las nuevas tendencias son la construcción de arquitecturas que integran aprendizaje deductivo e inductivo. Con esta integración se pretende completar una teoría con la utilización de casos ya resueltos.

4.8 Ejercicios

1. El EBL en general necesita tener como entradas un objetivo, un ejemplo y una teoría del dominio completa. En la Evaluación parcial se sigue un método exhaustivo para la resolución del problema, esto es, los nodos de la traza se expanden utilizando todas las

reglas posibles. Así pues, en este caso no es necesario tener un ejemplo de entrada. ¿Por qué?.

2. A fín de evitar los inconvenientes de los métodos inductivos y deductivos, se ha propuesto la integración de ambos tipos de métodos. ¿En qué forma puede ayudar la incorporación del EBL en un sistema de razonamiento basado en casos?. ¿En qué forma un sistema de razonamiento basado en casos puede ayudar a un sistema que utiliza EBL?. (En ambas preguntas se supone que la teoría del dominio no es completa).

Capítulo 5 Conexionismo

5.1 Introducción

Bajo este título genérico se estudia un modelo de cálculo que ha cobrado –de hecho, ha recobrado– gran auge en la última década: las redes neuronales. Buena parte de este interés proviene del lado de investigadores de campos ajenos a la Inteligencia Artificial. Las razones de esta popularidad hay que situarlas básicamente en el notable incremento de la capacidad de computación, en la posibilidad de crear *chips* neuronales, el advenimiento de máquinas masivamente paralelas sobre las que realizar simulaciones realistas, el regusto amargo derivado del abandono de los perceptrones y el consiguiente ansia de superarlo y, por último, la relación existente con un campo que nunca ha dejado de estudiarse y que sigue siendo fascinante: los sistemas no lineales. Por supuesto, no quisiéramos dejar de señalar la razón quizá subyacente a todas las anteriores: el resurgimiento del conexionismo en general como alternativa viable a la IA clásica.

Ya desde sus inicios, y hasta mediada la década de los 80, la inmensa mayoría de los sistemas de procesamiento de la información han estado –y siguen estando– basados en un mismo paradigma: la llamada *computación programada*. La aproximación clásica para resolver un problema consiste en derivar un algoritmo o un conjunto de reglas (o ambos) que lo resuelva para posteriormente realizarlo en un lenguaje de programación.

Ahora bien, es éste un método que se puede llevar a cabo solamente si se *conoce* de una manera precisa y no ambigua el proceso a describir (esto es, si se conoce *a priori* un algoritmo que lo resuelva). En caso contrario, la alternativa es intentar construir uno sobre la marcha, modificándolo y adaptándolo según sea su comportamiento, probablemente por el método de prueba-y-error, lo que suele ser una tarea tremendamente costosa para la mayoría de aplicaciones no triviales.

Al otro enfoque se le conoce como *computación neuronal*, *neurocomputación* o, simplemente, *conexionismo*. A diferencia del anterior, no necesita algoritmo específico alguno para la resolución del problema y requiere muy poco *software* en comparación con la complejidad de aquél. No importa si los algoritmos y/o reglas no son conocidos (o lo son, pero muy caros de realizar). Un precio a pagar, eso sí, es su clara orientación a campos esencialmente numéricos o de muy bajo nivel –su extensión a problemas de más "alto nivel" es objeto de intensa investigación hoy por hoy– , como el procesamiento de la señal, reconocimiento de formas, análisis de datos, control de procesos, etc.

De cara a precisar qué tipo de sistemas se trata en este capítulo, se ofrece la siguiente definición informal de un sistema conexionista:

Definición 5.1. Neurocomputación: disciplina que trata con sistemas de procesamiento de la información adaptativos, paralelos y distribuidos, y que desarrollan sus capacidades de procesamiento en respuesta a las señales provenientes de su entorno.

Las estructuras principales son las *redes neuronales*, aunque el espectro es bastante amplio, valiendo la pena citar, entre otras:

- Las memorias asociativas
- Las memorias auto-adaptativas direccionables por contenido
- Los sistemas de enfriamiento simulado (*simulated annealing*)
- Los sistemas genéticos
- Los sistemas de aprendizaje difusos
- Los autómatas aprendices

De entre ellas estudiaremos principalmente –junto con las propias redes– las tres primeras, mientras que los sistemas genéticos son tratados en el capítulo 7. En cuanto a los sistemas de aprendizaje difusos y su relación con las redes, [KOSK92] es una buena referencia. Los autómatas aprendices caen ya fuera del propósito de este libro.

Según la definición anterior, las redes neuronales corresponderían a sistemas de neurocomputación formados por *elementos de proceso* interconectados a través de canales de transmisión unidireccionales llamados *conexiones*. Cada elemento o unidad de proceso tiene un número cualquiera de conexiones de entrada y una única de salida (aunque puede conectarse a tantas unidades como se quiera). La computación realizada por cada elemento también es arbitraria, con la única restricción de ser *local* a la unidad, esto es, debe depender solamente de los valores de sus conexiones de entrada y, eventualmente, del valor almacenado en la memoria local de la unidad.

5.2 El modelo biológico

Las redes neuronales están claramente inspiradas en un modelo tomado del cerebro. Esto quiere decir que se han intentado plasmar los aspectos esenciales de una neurona real a la hora de diseñar una neurona "artificial". Ahora bien, la manera en que se organizan estas neuronas es (casi) completamente desconocida con lo que los modelos artificiales presentan una distribución (llamada *arquitectura* de la red) totalmente diseñada *ad hoc*, sin ninguna pretensión de semejanza biológica. Además, la propia neurona artificial suele ser un modelo bastante simplificado –como se verá– de la real. Por consiguiente, la pretensión de que una red neuronal pueda imitar el funcionamiento del cerebro no pasa de ahí, si bien es cierto que, aún con su simplicidad, ha conseguido grandes éxitos en diversos campos concretos de aplicación.

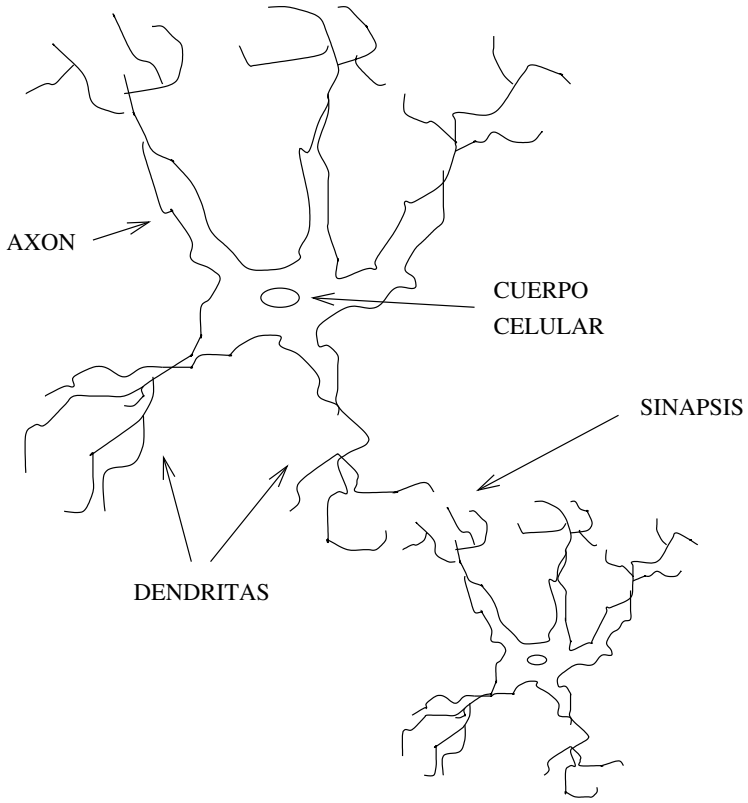


Figura 5.1: Dibujo esquemático de una neurona real.

La neurona es el elemento básico del sistema nervioso humano: participa en él en un número aproximado de 10^{11}, que comparten unas 10^{15} conexiones. Lo que hace únicas a las células del sistema nervioso –en comparación con las de otros sistemas del cuerpo humano– es su capacidad de recepción, proceso y transmisión de señales electroquímicas a través de ellas mismas y de sus conexiones.

La mayoría de las neuronas consisten en un **cuerpo celular** unido a un **axón** y a varias **dendritas** (véase figura 5.1). Funcionalmente hablando, las señales llegan a las dendritas procedentes de los axones de otras células a través de puntos de conexión llamados **sinapsis**. Una vez allí, las señales recibidas pasan al cuerpo celular donde son combinadas con otras señales (provenientes de otras dendritas de la célula). Si, en un cierto período de tiempo, el resultado de esta combinación excede un cierto valor límite, la célula se activa, lo que se traduce en un impulso de salida que recorre el axón y se propaga por las sinapsis correspondientes. De esta manera, cada neurona recibe –vía sus dendritas– impulsos de cientos de otras neuronas y envía su propio pulso –resultado de éstos– a cientos de otras más. Es esta complejidad de conexión –más que el trabajo realizado por cada neurona– la que posibilita la realización de tareas como las que habitualmente son adscritas a los seres inteligentes.

5.3 Perspectiva histórica

Los inicios de la neurocomputación en lo que a redes neuronales se refiere se remontan a 1943, donde los trabajos de Warren McCulloch y Walter Pitts [PITT43] mostraron cómo redes en apariencia simples podían realizar cualquier función lógica o aritmética. También –cómo no– John von Neumann, junto con Norbert Wiener y Arthur Rosenblueth, [ROSE53] se interesó por el tema, imaginando la posibilidad de construir ordenadores inspirados en arquitecturas que semejaran el cerebro humano.

Clave en esta época fue el libro de Donald Hebb [HEBB49] **The organization of behavior**, donde propone la idea –por otro lado, no nueva– de estudiar las conexiones neuronales como formas del condicionamiento clásico en psicología. Su aportación consistió en proponer, por primera vez, una fórmula concreta de cálculo del condicionamiento entre pares de neuronas, juntamente con una explicación cualitativa del proceso.

El primer neurocomputador construído con éxito se desarrolló entre 1957 y 1958 bajo la dirección de Frank Rosenblatt, siendo aplicado principalmente al reconocimiento de patrones [ROSE61] (como ejemplo típico de la época, el reconocimiento de caracteres).

Las reglas de cálculo seguían mejorándose, destacando la propuesta por Bernard Widrow y Ted Hoff que –conocida como la regla de Widrow-Hoff [WIDR60]– fue la base del algoritmo de *backpropagation* (ver §5.8) y se sigue utilizando hoy en día.

Pero los problemas empezaron a aparecer mediados los 60, principalmente debido a la falta de rigor en los experimentos, primando el empirismo por encima de la analítica (parecía que se volvía a los tiempos de los alquimistas). Todo esto, unido a que las grandes esperanzas depositadas en el paradigma (se especulaba ya con la idea de construir ordenadores de capacidades similares a la del cerebro humano) pronto se vieron frustradas por la realidad, propiciaron la típica historia de "auge y caída" de las redes neuronales.

La guinda a este período la pusieron Minsky y Papert en su libro **Perceptrons** [MINS69b]. En él, los autores dan una prueba formal de la incapacidad de un perceptrón (máximo exponente del cálculo neuronal en aquel momento) de realizar la función lógica "O exclusiva" (ni otras varias).

El período de 1967 a 1982 corresponde a una investigación latente, casi nula en los Estados Unidos y más activa en Europa, Japón y la URSS. Investigadores clave como Anderson [ANDE81] , Grossberg [GROS76] , Amari [AMAR67] , Willshaw [WILL69] , von der Malsburg [MALS73] , Fukushima [FUKU75] , Hinton [HINT81] , y Kohonen [KOHO77] realizaron sus primeros trabajos en esta época.

El resurgimiento –mediados los años 80– viene de la mano de dos factores: por un lado, la DARPA (*Defense Advanced Research Projects Agency*) americana se interesó por los pocos artículos que se escribían en aquel entonces, ayudando a formar grupos de trabajo. Por otro, las contribuciones de John Hopfield, mediante un par de artículos ([HOPF82] y [HOPF83]) que tuvieron gran aceptación y –más importante si cabe– más distribución hacia la comunidad científica en general, consiguiendo captar un buen número de adeptos a la disciplina, que rápidamente se pusieron a trabajar a su vez. Todo esto condujo a la serie de libros PDP (*Parallel Distributed Processing*) que, editados por David Rumelhart y James McClelland, canalizaron las experiencias obtenidas y centraron el tema ([RUME86a] y [RUME86b]). En

1987 se celebró la primera conferencia de la "nueva era": la *IEEE International Conference on Neural Networks*.

5.4 Nociones preliminares

Las redes neuronales son un tipo de arquitectura de flujo de datos. Éstas son a su vez arquitecturas MIMD (*Multiple-instruction Multiple-Data*) sin memoria global o compartida donde cada elemento opera solamente cuando tiene presente toda la información que le es necesaria.

Veamos primero una definición más formal:

Definición 5.2. Un **grafo dirigido** es una estructura compuesta por un conjunto de puntos (llamados nodos) y un conjunto de segmentos de línea dirigidos (llamados arcos o aristas) que los conectan.

Definición 5.3. Una **red neuronal** es una estructura procesadora de la información, distribuida y paralela, que tiene la forma de un grafo dirigido donde, además:

1. los nodos se denominan *elementos de proceso*.
2. los arcos se denominan *conexiones* (caminos propagadores de la señal, instantáneos y unidireccionales).
3. cada elemento puede tener tantas conexiones de entrada como se quiera, y sólo una de salida, aunque ésta puede estar conectada a una o más neuronas.
4. los elementos pueden tener *memoria local*.
5. cada elemento tiene una *función de transferencia* que, en su forma más general, puede expresarse como:

$$Conexiones\ entrada \times Memoria\ local \longrightarrow Conexión\ salida \times Memoria\ local$$

6. a la descripción matemática de la red se le llama *arquitectura*.

Todas las redes neuronales se pueden dividir en *niveles*, donde cada uno puede tener su propia arquitectura, función de transferencia o tipo de mecanismo de actualización. Este último término se refiere a la manera en que se actualizan todos los elementos de un mismo nivel, existiendo dos métodos básicos:

- Continuo: se actualizan continuamente, de manera asíncrona, y según un orden prefijado.
- Episódico: ha de existir una señal de sincronía, que provoca una actualización simultánea.

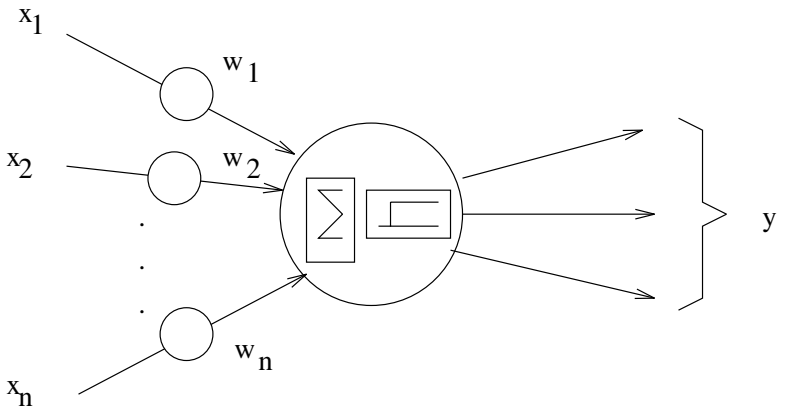


Figura 5.2: Modelo básico de neurona artificial.

5.4.1 El modelo básico

Aquí se examina una neurona artificial más de cerca (figura 5.2). En esencia, el modelo más simple es aquel al que se proporciona un vector de valores a través de sus conexiones de entrada (equivalentes a las dendritas), disponiendo éstas de un *peso* asociado (equivalente a la conexión sináptica) y de una función de transferencia –la suma *ponderada* de las entradas seguida de un corte mediante la comparación con un cierto valor límite–, correspondiente a la combinación realizada en el cuerpo celular y a la decisión sobre si la neurona enviará un pulso o no, respectivamente. Ésta última función –que suele ser más general– se denomina *función de activación*.

Sea n el número de conexiones de entrada, $\vec{x} = (x_1, x_2, \ldots, x_n)$ el vector de entrada y $\vec{w} = (w_1, w_2, \ldots, w_n)$ el vector de pesos de una neurona concreta. Entonces,

$$y' = \sum_{i=1}^{n} x_i w_i$$

o bien –en notación vectorial– $y' = \vec{x}^t \vec{w}$, resultado al que se aplica la función de activación para producir la salida y de la neurona. La forma más sencilla de esta función es una lineal –cuyo único propósito es un escalado– del estilo:

$$y = \alpha y' + \beta$$

donde α y β son constantes. Una función muy simple pero que da problemas (como se verá más adelante) es la función lindero, con L de nuevo constante prefijada,

$$y = \begin{cases} 1 & \text{si } y' > L \\ 0 & \text{si } y' \leq L \end{cases}$$

Así, cada w_i representa la fuerza de la conexión sináptica correspondiente, y' la combinación del cuerpo celular e y el recorrido por el axón.

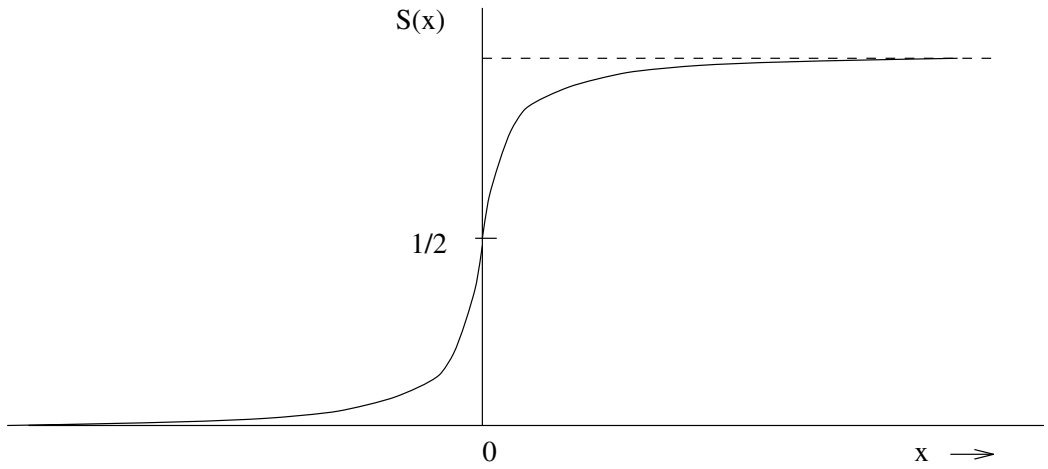


Figura 5.3: Función sigmoidal.

Una función de activación muy utilizada es la sigmoide S (llamada así por su forma, ver figura 5.3), que proporciona siempre un valor entre cero y uno:

$$y = S(y') = \frac{1}{1 + e^{-y'}}.$$

La introducción de esta función provoca (al igual que la función lindero) una ganancia (pendiente de la curva en un punto) no lineal, pero de tal manera que da ganancias muy bajas para valores de y' muy negativos (es decir, cuando la neurona está muy lejos de activarse), pasando por un valor alto a excitación cero ($y' = 0$), volviendo a valores bajos a medida que y' aumenta en los positivos. La ventaja de esta función es que resuelve un típico problema de las redes neuronales, presente en la función lindero: la saturación frente al ruido. La pregunta es: ¿ cómo puede una neurona tratar con valores muy pequeños y muy grandes siempre con la misma función de activación? Las señales pequeñas (cercanas a cero en valor absoluto) requieren gran ganancia si han de ser en algo significativas, mientras que una secuencia de neuronas conectadas en serie puede ir produciendo resultados muy altos que saturen la salida de una neurona, además de amplificar enormemente el ruido, siempre presente. De esta manera, sin embargo, valores altos en valor absoluto tendrán ganancias moderadas previniendo la saturación.

5.4.2 Redes monocapa

Como se vió anteriormente, la capacidad de computación de las neuronas viene dada por su uso masivo. La arquitectura más simple es aquella formada por una sola capa, conjunto usualmente alineado de neuronas con algún rasgo arquitectónico distintivo y uniforme para todas ellas (figura 5.4).

Las entradas a la red no constituyen capa alguna ni realizan ningún tipo de computación, sino que son tomadas como simples distribuidoras del vector de entrada, mostradas por

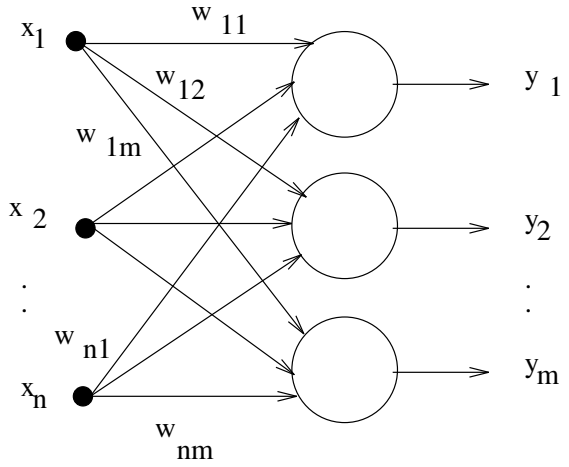


Figura 5.4: Red monocapa.

conveniencia[1]. La salida de la capa es un vector que, a su vez, constituye la salida final de la red. En este caso se tiene una matriz de pesos W de m filas por n columnas, donde n es el rango del vector de entrada y m el número de neuronas de la capa. Así, w_{12} correspondería al peso dado a la primera entrada de la segunda neurona. La salida $\vec{y}$ será ahora

$$\vec{y} = F(W\vec{x}),$$

donde F es la función de activación que se utilice e $\vec{y}, \vec{x}$ vectores columna[2].

5.4.3 Redes multicapa

La estructuración en capas consecutivas incrementa notablemente el poder de las redes neuronales. A este tipo de configuración se le denomina *en cascada*, y a las capas para las cuales no tenemos acceso directo a su salida se les llama *ocultas* (es decir, todas menos la última) y se obtiene simplemente conectando la salida (vectorial) de una capa a la entrada (vectorial) de la siguiente (véase figura 5.5). Nótese que la conectividad no tiene porqué ser total.

Las redes multicapa no añaden capacidad computacional a menos que la función de activación de alguna de las capas sea no lineal. La demostración intuitiva es sencilla: una capa viene representada por su matriz de pesos asociada. Así, siendo $\vec{x}$ el vector de entrada y $W_1, W_2, \ldots$ las diferentes matrices, la salida de la primera capa será $W_1\vec{x}$. Si no hay función de activación no lineal, la salida de la segunda será $W_2(W_1\vec{x})$, y así sucesivamente. Como el producto de matrices es asociativo, la expresión anterior es equivalente a $(W_2W_1)\vec{x}$, lo cual indica que una red bicapa sería equivalente a una monocapa con matriz de pesos W_1W_2. En las siguientes secciones se estudiará más en profundidad este resultado.

[1] Aunque algunos autores la denominan *capa de entrada*.
[2] A lo largo de todo el texto se utilizarán vectores columna.

5.4.4 Entrenamiento de redes neuronales

Vistas hasta ahora, las redes no tendrían gran interés si no fuera por su capacidad de *aprender*, esto es, de modificar sus matrices de pesos de manera que la aplicación de una entrada dé como resultado una cierta salida deseada, para un conjunto (potencialmente) infinito de entradas y salidas. El aprendizaje en sí se lleva a cabo presentando secuencialmente a la red vectores de entrada y ajustando los pesos según una cierta *regla de aprendizaje*. Se espera que el conjunto de pesos converja hacia unos valores estables que minimicen el error. Entre los tipos de aprendizaje destacan tres grandes grupos:

Aprendizaje supervisado. Junto con cada vector de entrada, se suministra a la red el vector de salida deseado, de manera que ésta pueda corregir los pesos en base a su diferencia. A cada par formado por un vector de entrada y su salida esperada se le denomina *par de entrenamiento*. El proceso se repite hasta que el error para cada uno de los pares de entrenamiento es aceptable bajo cierto criterio.

Aprendizaje no supervisado. Por diversas razones, entre las que se encuentran la "irrealidad" –no siempre se tiene a mano el resultado deseado– o su implausiblidad biológica, el aprendizaje supervisado no es el único utilizado. El método no supervisado es bastante más plausible psicológica y biológicamente. Ideado, entre otros, por Kohonen [KOHO88], no necesita de la respuesta correcta para realizar las correciones en los pesos. En vez de eso, se busca la *consistencia* entre entradas y salidas, esto es, entradas similares deben tener salidas iguales o similares. Por tanto, el método agrupa las entradas en *clases* según criterios estadísticos. Lo que no se sabe –a diferencia del método supervisado– es el aspecto que tendrá cada salida hasta que ésta no es calculada por la red, lo que provoca las necesarias interpretaciones a posteriori hacia un formato más comprensible. A este método se le denomina también *autoorganización*.

Aprendizaje gradual o por refuerzo. En cierta manera, este tipo de aprendizaje está a caballo entre los dos anteriores, acercándose más a uno o a otro dependiendo del criterio utilizado en cada caso. En su forma más general, consiste en proporcionar a la red –de cuando en cuando– una valoración *global* de cuán bien (o mal) lo está haciendo desde la última valoración.

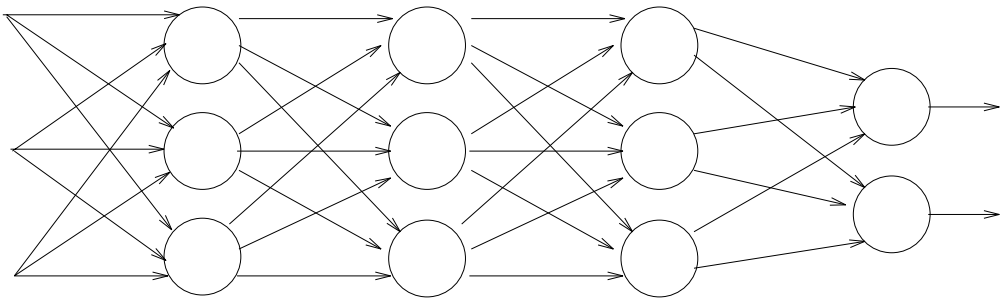


Figura 5.5: Red multicapa.

5.4.5 Algoritmos básicos de aprendizaje: asociadores lineales

Buena parte de las reglas de aprendizaje utilizadas hoy en día tiene su origen en la ya mencionada regla de Hebb. Ésta pertenece al tipo no supervisado, donde la conexión sináptica (el peso) es reforzado si las neuronas a ambos lados de la conexión están activadas. Es decir, caminos formados por neuronas que se activan en secuencia son ayudados a hacerlo. Claramente es un premio al hábito y al aprendizaje por repetición. A partir de este momento, se asume una actualización de tipo episódico, lo que lleva a considerar el tiempo de manera discreta. Considérese un par de neuronas i, j de manera que la salida de i sea una de las entradas de j. Sea entonces:

$$\begin{cases} w_{ij}(t) & \text{el peso entre la neurona } i \text{ y la } j \text{ en tiempo } t \\ y_i(t) & = \sum_{k=1}^{n} w_{ki}(t) y_k(t), \\ & \text{la salida de la neurona } i \text{ en tiempo } t, \\ 0 < \alpha \leq 1 & \text{la razón de aprendizaje} \end{cases}$$

Así,

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\, y_i(t) y_j(t),$$

donde los valores iniciales de los pesos son tomados como cero. Esto enseña a la red a comportarse como un *asociador de patrones*, es decir, la presentación de un patrón de entrada o estímulo da como respuesta otro patrón, correspondiente al aprendido por la red.

Se dice que una red es un asociador *lineal* si su función de activación lo es. Este tipo de redes no utiliza, por tanto, funciones de corte o lindero ni sigmoidales. El caso más frecuente es la ausencia de función de activación, esto es, el cómputo realizado es ya la salida de la neurona.

La regla de Hebb es un caso particular de asociador lineal. La forma general se puede expresar de la manera siguiente:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\, r_{ij}(t),$$

siendo $r_{ij}(t)$ la señal de refuerzo entre ambas neuronas. En el caso de aprendizaje hebbiano, se tiene $r_{ij}(t) = y_i(t) y_j(t)$. El problema es que una neurona relega la responsabilidad de activarse a otra precedente, que a su vez hará lo propio con sus neuronas de entrada, y así sucesivamente.

Un desarrollo posterior lo constituye la regla de Widrow-Hoff, también llamada *regla delta*,

$$r_{ij}(t) = [\hat{y}_j(t) - y_j(t)]\, y_i(t)$$

donde $\hat{y}_j(t)$ es un número real[3] correspondiente a la salida *deseada* de la neurona j en tiempo t. Ésta es la esencia de los asociadores de patrones: a medida que se suceden las presentaciones de los pares de entrenamiento $< \vec{p}, \vec{q} >$, se espera que la red *converja*, esto es, para cada patrón $\vec{p}$ la salida deberá ser $\vec{q}$; es, claramente, un proceso iterativo.

A pesar de su apariencia, hay dos diferencias fundamentales entre la regla de Hebb y la regla delta, y merece la pena comentarlas:

1. La primera de ellas es efectivamente no supervisada. Sin embargo, un vistazo a la regla delta permite ver que necesita de la salida esperada, por lo que es claramente una regla

[3] Recuérdese que no hemos puesto limitación alguna a la salida.

de aprendizaje supervisado. Esto tiene, a su vez, una importante consecuencia: la regla delta sólo es adecuada para redes de una sola capa, pues desconoceríamos *a priori* la salida correcta para neuronas de capas intermedias. Esta situación, en cambio, no se da si utilizamos aprendizaje hebbiano tal y como lo hemos definido.

2. Una red muestra recuerdo perfecto (*perfect recall*) si responde de manera exacta a todos los patrones aprendidos. En este sentido, la regla delta asegura recuerdo perfecto si los patrones de entrada usados en el entrenamiento son linealmente independientes, mientras que la regla de Hebb necesita para ello que formen un conjunto ortogonal. La regla delta *ortogonaliza* este conjunto en caso que no lo fuera: es el llamado *aprendizaje ortogonal*.

En vez de como asociadores de patrones –a un patrón de entrada le corresponde uno de salida– una manera alternativa de utilizar las redes es la siguiente: dada una porción de un patrón de entrada, se espera de la red que lo complete o *reconstruya*: es la base del *direccionamiento por contenido* y de las *memorias asociativas*. Ésta es una tarea muy difícil para una clásica computación en serie, pues se trataría de generar todos los posibles patrones susceptibles de estar asociados con la entrada e iterar sobre ellos para encontrar el más parecido al original; eso suponiendo que la información en él contenida –aún siendo parcial– sea correcta. Para realizar esta tarea mediante una red, piénsese en una situación en que cada elemento de proceso está conectado mutuamente de una manera *excitatoria* (peso asociado altamente positivo) con todos los elementos relacionados con él[4]. Así, la activación de una o varias neuronas (por un patrón de entrada) provocará la del resto. Un problema típico de estas redes es la activación masiva, debida a la propagación de la actividad por la red. Se suele resolver con conexiones *inhibitorias* (peso asociado altamente negativo) entre neuronas o grupos de ellas (y, por ende, entre los conceptos que representan) mútuamente excluyentes. A esta técnica se le llama *inhibición lateral* y forma la base del aprendizaje por competición.

5.5 El perceptrón

Una red monocapa donde cada neurona responde al modelo básico estudiado en el apartado 5.4.1 se conoce por el nombre de perceptrón. La función de activación es la lindero (figura 5.6). Los perceptrones se explican usualmente por razones históricas, pues han sido ya casi completamente abandonados en aplicaciones prácticas. En su momento significaron un gran avance [ROSE61] y, un poco después, una fuerte desilusión al comprobarse sus limitaciones. El desconocimiento por entonces de algoritmos de entrenamiento multicapa contribuyó a este hecho. De todas maneras, son siempre un buen punto de partida para el estudio básico de las redes neuronales.

Se estudiará –por ser más didáctica– una neurona ligeramente diferente de la básica, donde la función lindero ha sido modificada y donde se ha añadido una entrada extra. Se considerarán perceptrones de una sola neurona[5], con entradas y salida binarias. La fórmula que se utiliza será la siguiente, donde P es la salida del perceptrón:

$$P = \begin{cases} 1 & \text{si } \sum_0^n x_i w_i \geq 0 \\ 0 & \text{si } \sum_0^n x_i w_i < 0 \end{cases}$$

[4] Por ejemplo, porque forman parte de la representación de la misma propiedad: se trata de redes con el conocimiento distribuido.

[5] Por doble motivo: por simplicidad y porque así era el perceptrón original.

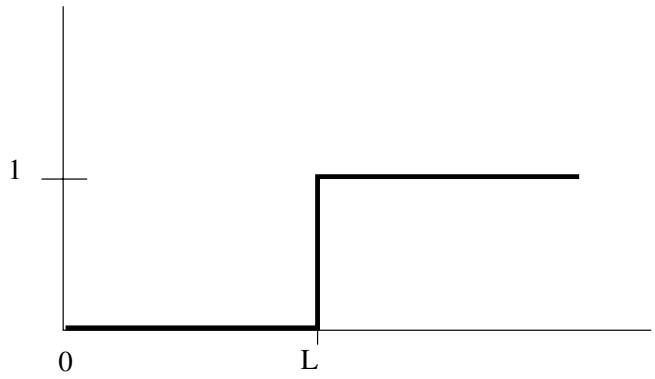


Figura 5.6: Función de corte o lindero.

Obsérvese que se trata básicamente del modelo visto en 5.4.1, con dos puntos dignos de mención:

1. Es una función no lineal, debido al corte proporcionado por el lindero.
2. Se ha considerado un lindero igual a 0 y se ha añadido una entrada extra (x_0) fijada a la constante 1. Esto proporciona un lindero *entrenable*, representado por el peso asociado w_0.

Además de la neurona, el perceptrón dispone de *cajas lógicas* que realizan una función lógica cualquiera de sus entradas, estando sus salidas conectadas a las entradas de la neurona. En la figura 5.7 se aprecia un perceptrón obteniendo las entradas de una parrilla bidimensional —a la que se llamó, un poco ingenuamente, *retina*. Con referencia a las cajas lógicas, es conveniente notar que no pueden soportar un gran número de entradas, pues su complejidad se haría excesiva y poco acorde con la simplicidad del perceptrón en sí. Esta limitación llevó a clasificar los perceptrones según ciertos criterios, entre los cuales estaban:

Orden: un perceptrón de orden n no tiene ninguna caja lógica de más de n entradas.

Diámetro: en el caso de entradas conectadas a una retina o plano, establece el diámetro del círculo en el que todas las entradas de una caja deben caer.

Gamba: nombre dado[6] a los perceptrones sin limitación de orden en los que cada caja lógica es ella misma computada por un perceptrón de orden 1.

Directo: perceptrón sin cajas lógicas, es decir, las entradas son directamente las de la neurona.

La tarea del perceptrón se puede observar en la figura 5.8. Al tratarse de una salida binaria, se puede considerar ésta como indicadora de dos posibles clases a la que pertenezca cada entrada, la cual podemos ver como un punto en un espacio n-dimensional[7].

[6] No es una traducción: es el nombre original.

[7] Se puede dar una interpretación a cada coordenada del vector, como `altura`, `color`, *etc*, aunque no es necesaria para la discusión que nos ocupa.

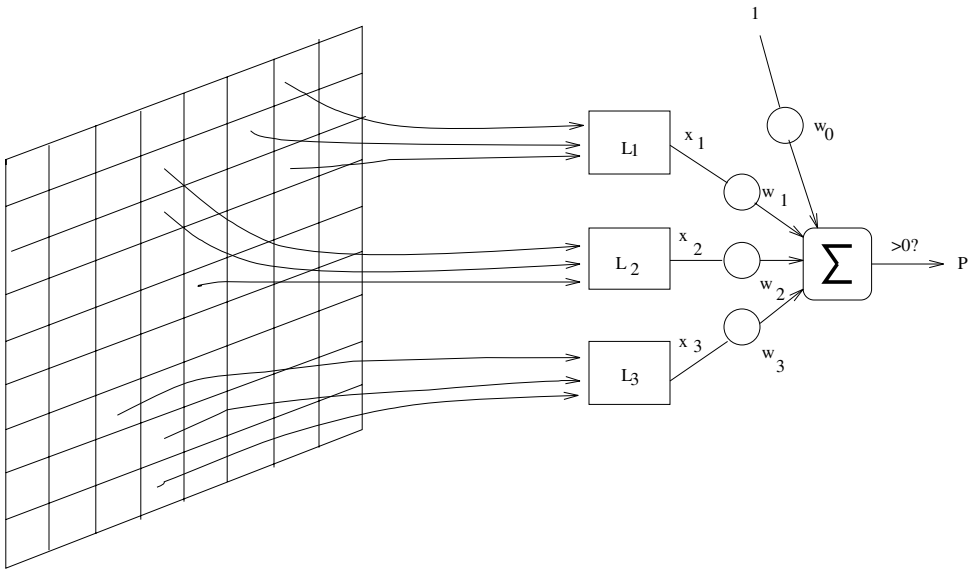


Figura 5.7: Un perceptrón reconocedor de imágenes.

La figura 5.8 muestra una situación en que los dos conjuntos de puntos (correspondientes a las dos clases) pueden separarse por un hiperplano[8]. Las clases que presentan esta propiedad se denominan *separables linealmente*. Así pues, se trata de *orientar* el hiperplano –encontrando valores adecuados de los pesos– de manera que separe las dos clases, si es que tal cosa es posible.

La manera de determinar dichos pesos –es decir, la regla de aprendizaje– es la llamada *regla del perceptrón*, que pertenece al tipo supervisado. Cada presentación de una entrada se acompaña de su salida (la clase) esperada C. Así,

$$w_i(t+1) = w_i(t) + (C - P)x_i$$

Es decir, si se acierta la clase, nada se modifica y, sino, se alterará cada peso con la suma o la resta del valor de la entrada correspondiente x_i, según sea el signo de $C - P$. Los pesos iniciales son iguales a 0. La explicación de este proceder es muy intuitiva: si, por ejemplo, la salida P del perceptrón es 1 cuando debiera ser 0, una manera de rebajar el valor del sumatorio es rebajar el de los pesos mediante sus valores de entrada correspondientes, y simétricamente para el caso contrario. El proceso se repite hasta que se obtiene la respuesta correcta para cada patrón de entrada.

Veamos un ejemplo: seguiremos el proceso de un perceptrón para aprender la función lógica "o". En la tabla siguiente se observa la evolución de los pesos a medida que se van presentando ejemplos a un perceptrón directo de dos entradas (más la correspondiente al lindero, x_0).

[8] En dos dimensiones un hiperplano es una línea, en 3 un plano común y en un espacio n-dimensional una superficie plana de dimensión $n - 1$.

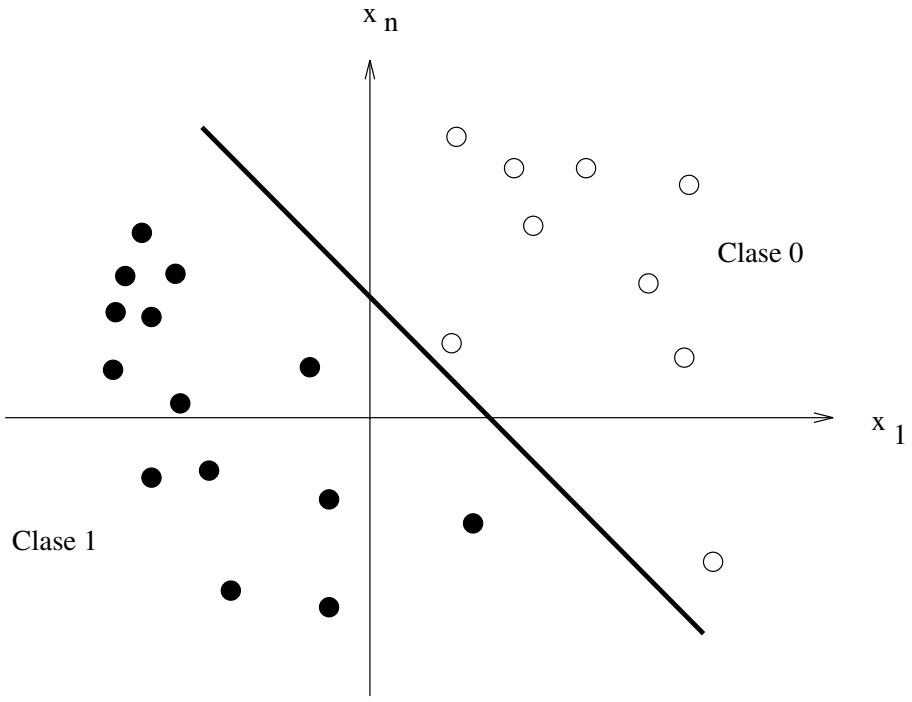


Figura 5.8: Separabilidad lineal.

El hiperplano $w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = 0$ debe separar completamente las dos nubes de puntos, correspondientes a las dos clases, 0 y 1.

$x_0 x_1 x_2$	C	P	$w_0 w_1 w_2$
- - -	-	-	0 0 0
1 0 0	0	0	0 0 0
1 0 1	1	0	1 0 1
1 1 0	1	1	1 0 1
1 1 1	1	1	1 0 1
1 0 0	0	1	0 0 1
1 0 1	1	1	0 0 1
1 1 0	1	0	1 1 1
1 1 1	1	1	1 1 1
1 0 0	0	1	0 1 1

Es éste un proceso de convergencia iterativo, en el que el paralelismo viene dado por la computación simultánea de todas las cajas lógicas. El mismo Rosenblatt demostró el llamado *Teorema de convergencia del perceptrón*: si las clases son linealmente separables, el perceptrón aprenderá a separarlas *a la perfección* en un número finito de entrenamientos, independientemente de los valores iniciales de los pesos, lo cual es un resultado francamente

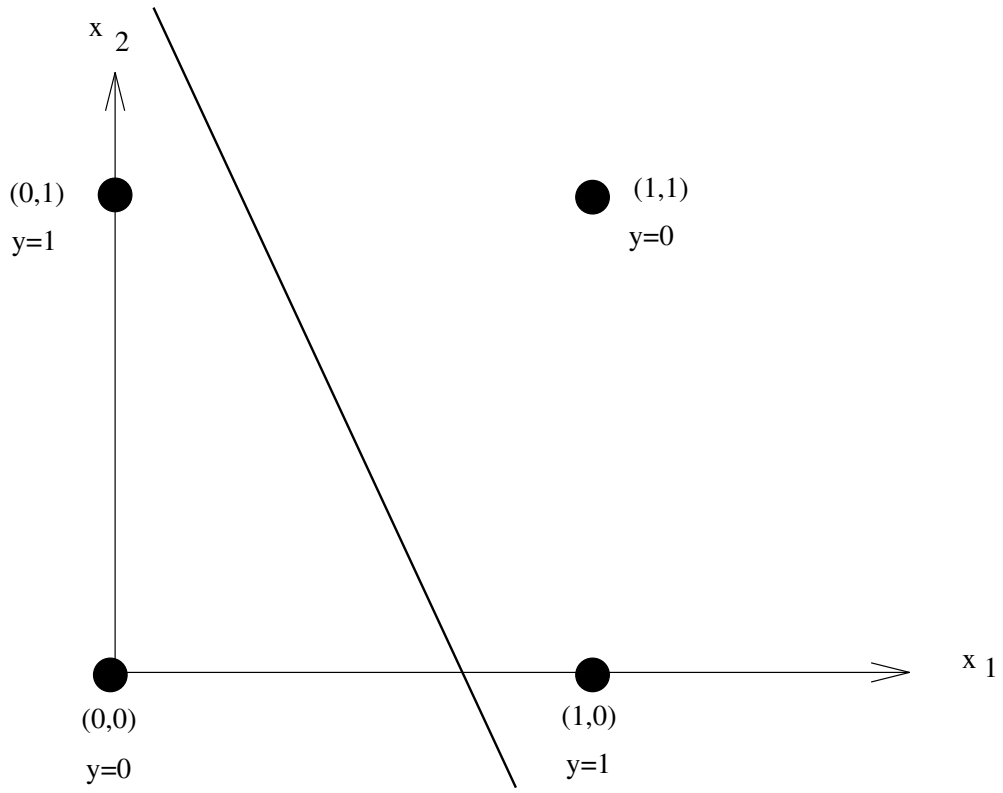


Figura 5.9: La función XOR no puede ser computada por un perceptrón al no ser linealmente separable.

alentador. El problema consiste en saber *a priori* si la clases son separables o no. Como ejemplo paradigmático, considérese el problema de aprender la función lógica XOR ("o" exclusiva), dada por la tabla siguiente:

x_1	x_2	XOR
0	0	0
0	1	1
0	1	1
1	1	0

Un simple vistazo a la figura 5.9 muestra la imposibilidad de dibujar una línea recta que deje a un lado los puntos (0,0) y (1,1) y al otro los puntos (0,1) y (1,0).

Desgraciadamente, éste no es un caso aislado: existen infinidad de funciones no separables linealmente. En general, una neurona con n entradas binarias puede aceptar 2^n patrones diferentes, pudiendo tener cada uno 2 posibles salidas, con lo que el número total de funciones de n variables es 2^{2^n}. De ellas, como se muestra en la figura 5.10, sólo unas pocas cumplen la propiedad a medida que n aumenta.

n	2^{2^n}	FLS
1	4	4
2	16	14
3	256	104
4	65 536	1 882
5	4.3×10^9	94 572
6	1.8×10^{19}	5 028 134

Figura 5.10: Número de funciones linealmente separables (FLS).

En los años 60, el caso concreto de la XOR fue suficiente para dar al traste con las redes monocapa, y se empezó a estudiar cómo añadir más de manera que se ampliase la capacidad de cómputo y se resolviese este problema. Por ejemplo, una red de dos capas es ya capaz de separar puntos situados dentro y fuera de regiones convexas, sean abiertas o cerradas. Una región es convexa si cualquier par de puntos de su interior puede ser unido por una recta sin salir de la región. Ésta es cerrada si todos los puntos están dentro de unos límites, y abierta si hay partes de la región sin límite definido (por ejemplo, los puntos entre dos paralelas). Véase para un mejor entendimiento la figura 5.11. Considérese ahora una red bicapa como la de la figura 5.12, compuesta por tres neuronas, con los pesos especificados en ella. De esta manera, la neurona de la segunda capa efectúa una "y" lógica, ya que se necesita un 1 en ambas entradas para que dé a su vez salida 1 (recuérdese la función de la entrada extra, cuyo peso w_0 es el valor actual del lindero de la neurona de la segunda capa). Así, y suponiendo que una de las neuronas de la primera capa da como resultado un 1 para entradas situadas por debajo de una cierta recta r_1 (y 0 en otro caso) y que la otra neurona tiene como salida 1 para puntos por encima de una recta r_2 (y 0 en otro caso), se tiene que la red distingue entre puntos dentro y fuera de una región convexa (figura 5.13), con lo que puede computar cualquiera de las 2^{2^n} funciones de n entradas binarias.

Similarmente, y a base de añadir neuronas en la primera capa y aumentar el número de entradas de la neurona de la segunda (con 3 podríamos distinguir triángulos) se puede llegar a separar el interior y el exterior de cualquier polígono convexo. Por supuesto, la función

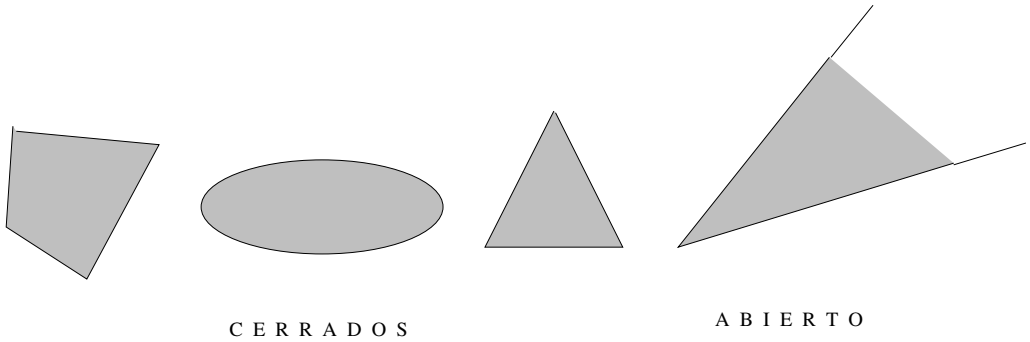


Figura 5.11: Polígonos convexos abiertos y cerrados.

realizada por esta neurona no tiene porqué limitarse a la función "y". Otras funciones darán lugar a comportamientos globales diferentes. En el caso de 2 líneas de entrada –según se aprecia en la figura 5.10– pueden simularse 14 de las 16 posibles (todas menos la XOR y su negado). Por otra parte, las entradas no tienen porqué ser binarias. Si consideramos valores continuos, el problema se reformula teniendo que distinguir entre regiones y no entre conjuntos de puntos. Por tanto, para separar dos regiones A y B, todos los puntos de A deberían poderse englobar en un polígono convexo que no contuviera ningún punto de B (o viceversa).

¿ Qué ocurre si añadimos una capa más? La respuesta es que se elimina la necesidad de que los polígonos sean convexos. Para entenderlo, piénsese que las entradas de la tercera capa serán grupos de polígonos convexos que –pasados por el "turmix" de la función lógica correspondiente a la neurona de la tercera capa– podrán así ser combinados unos con otros. Por ejemplo, si dicha función es una $A \vee B$, se obtendrá el equivalente a una unión, si $A \wedge \neg B$, un recorte, etcétera, obteniendo así superficies poligonales generales de cualquier precisión (dependerá del número de neuronas de la primera capa).

Como colofón a la discusión, veamos cómo se puede computar la función XOR, simplemente con el uso de una neurona oculta. La figura 5.14 muestra una red bicapa (formada por tan sólo dos neuronas) con la convención usual de los linderos como entradas extra fijadas a 1. Así, la primera neurona se activará solamente si sus dos entradas son 1. Si no lo son, la segunda se activará en cualquier caso y, si lo son, la activación de la primera evita que se active. Nótese que, desde el punto de vista de la segunda neurona, ésta tiene tres entradas, sin distinción de si provienen de la entrada original o de otra neurona. El problema que quedó abierto en aquel momento era el de *cómo* ajustar los pesos de redes de *más de una capa*, problema cuya solución representó en gran parte el auge renovado del tema y será estudiado en secciones posteriores.

5.6 La regla Delta

Pasaremos ahora a analizar en detalle uno de los métodos de aprendizaje introducidos en la sección 5.4.5: la regla delta. De su apariencia se aprecia que se basa en la corrección del error existente entre los patrones de entrada y los de salida. De hecho, lo que hace es asociar estos patrones de una manera biunívoca. Al igual que los perceptrones –y a diferencia de la regla de Hebb– sólo modifica pesos cuando hay discordancia entre ellos. Dado que la regla delta

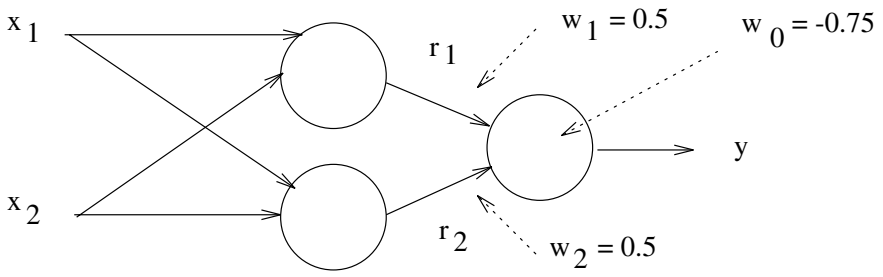


Figura 5.12: Red bicapa que *construye* un polígono abierto mediante la intersección de dos rectas.

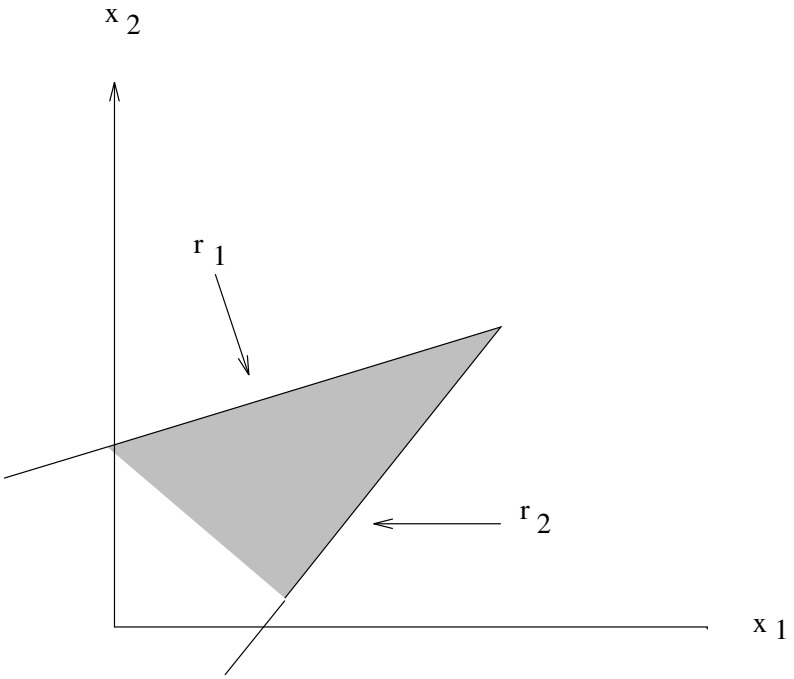


Figura 5.13: Polígono abierto reconocido por la red bicapa.

sólo puede realizar estas modificaciones si se conoce el patrón de salida, simplificaremos la notación de manera que refleje que trabajamos sobre redes monocapa, como la de la figura 5.4.2. Así, siendo $\hat{y}_{pj}$ y y_{pj} las salida esperada y real de la neurona j para el patrón p, y x_{pi} la entrada i del patrón p, el incremento del peso $\Delta_p w_{ij}$ valdrá:

$$\Delta_p w_{ij} = \alpha(\hat{y}_{pj} - y_{pj})x_{pi} = \alpha\,\delta_{pj}\,x_{pi}$$

En otras palabras, se suma al peso una cantidad proporcional a la entrada. En el caso más común de unidades lineales, esta regla minimiza los cuadrados de las diferencias de todos los errores acumulados. Si llamamos E_p al error de un patrón y E_T al error total, tendremos:

$$E_p = \frac{1}{2}\sum_j (\hat{y}_{pj} - y_{pj})^2$$

$$E_T = \sum_p E_p,$$

siendo el valor 1/2 un factor de conveniencia. La regla realiza un descenso del gradiente –yendo siempre por la mayor pendiente posible– en la superficie generada por el espacio de pesos, cuya altura en un punto dado es igual al error total para los pesos correspondientes a ese punto. Calcularemos para mostrarlo el incremento del error para un patrón dado respecto un peso cualquiera, y demostraremos que es:

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pj}\,x_{pi},$$

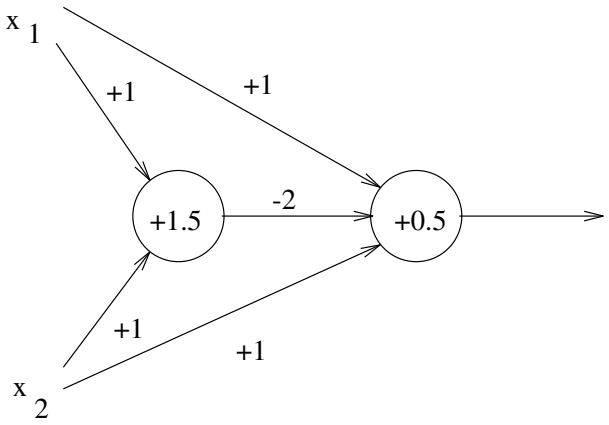


Figura 5.14: Reconocimiento de la función XOR.

es decir, proporcional a $\Delta_p w_{ij}$. Usando la regla de la cadena:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial w_{ij}} \tag{5.1}$$

Las derivadas parciales son sencillas de calcular:

$$\frac{\partial E_p}{\partial y_{pj}} = -(\hat{y}_{pj} - y_{pj}) = -\delta_{pj}$$

mientras que, al ser

$$y_{pj} = \sum_i w_{ij} x_{pi},$$

tendremos que

$$\frac{\partial y_{pj}}{\partial w_{ij}} = x_{pi}.$$

Por consiguiente, substituyendo en (5.1):

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pj} x_{pi}$$

Finalmente,

$$\frac{\partial E_T}{\partial w_{ij}} = \sum_p \frac{\partial E_p}{\partial w_{ij}} = -\sum_p \delta_{pj} x_{pi} \tag{5.2}$$

Como se puede ver, después de una presentación entera de todos los patrones, el incremento total en un peso es proporcional a la derivada calculada en (5.2) y, por tanto, la regla delta realiza un descenso del gradiente de E, que lleva a minimizar el error.

Comparemos este criterio con el del perceptrón. Se estableció que éste es capaz de un perfecto recuerdo si y sólo si los patrones de entrada eran separables linealmente. Ahora bien, si no lo son, el perceptrón generará un hiperplano que dará la salida correcta para ciertos casos, y

la incorrecta para otros, dependiendo del orden de presentación de los patrones, siendo por tanto una mala elección. Ahora bien, la regla delta procura en todo momento *minimizar el error total* con lo que, si bien no está asegurada la convergencia[9] del método –y, por tanto, el recuerdo perfecto– sí se comportará aceptablemente bien para *todos* los patrones presentados.

5.7 Funciones discriminantes lineales

Ésta sección está adaptada de [DUDA73], y puede obviarse en una primera lectura. Define los conceptos básicos de las superficies de decisión para dos categorías, establece los criterios que llevan a la separabilidad lineal y los métodos de convergencia asociados, entre ellos la regla del perceptrón y la regla delta.

5.7.1 Categorización binaria

Se dice que una función discriminante (FD) es lineal respecto de un vector $\vec{x}$ si se puede expresar como:

$$g(\vec{x}) = \vec{w}^t\vec{x} + w_0$$

donde $\vec{w}$ es el vector de pesos y w_0 el peso utilizado como lindero. Un clasificador lineal en dos categorías c_1 y c_2 realiza la siguiente función:

$$\begin{cases} c_1 & \text{si } g(\vec{x}) > 0 \\ c_2 & \text{si } g(\vec{x}) < 0 \end{cases}$$

Que es equivalente a decir:

$$\begin{cases} c_1 & \text{si } \vec{w}^t\vec{x} > -w_0 \\ c_2 & \text{si } \vec{w}^t\vec{x} < -w_0 \end{cases}$$

Si $g(\vec{x}) = 0$ la clase queda indefinida. Así, la ecuación $g(\vec{x}) > 0$ define la superficie de decisión que separa los puntos asignados a c_1 de los asignados a c_2. Si g es lineal –como es nuestro caso– la superficie es un hiperplano. Entonces, si dos puntos $\vec{x_1}$,$\vec{x_2}$ están sobre el hiperplano, se tiene:

$$\vec{w}^t\vec{x_1} + w_0 = \vec{w}^t\vec{x_2} + w_0$$

$$\equiv$$

$$\vec{w}^t(\vec{x_1} - \vec{x_2}) = 0,$$

lo que equivale a decir que $\vec{w}$ es siempre un vector normal al hiperplano. Además, $\vec{w}$ apunta hacia el lado positivo.

La FD $g(\vec{x})$ proporciona una manera algebraica de computar la distancia de $\vec{x}$ a un hiperplano H. Para ver esto, expresemos primero $\vec{x}$ como:

$$\vec{x} = \vec{x}_p + r\frac{\vec{w}}{||\vec{w}||},$$

[9] Se ha preferido en esta sección hacer hincapié en los métodos en sí mismos y no en su explicación o fundamentación matemática. A ésta se reserva la sección 5.7.

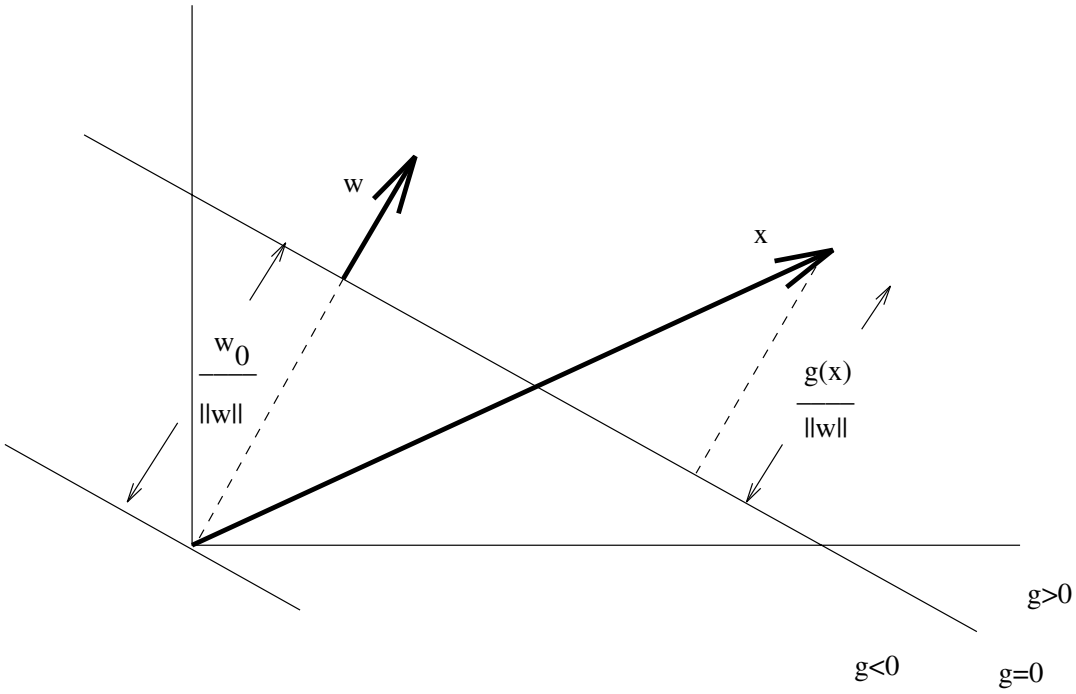


Figura 5.15: La superficie lineal de decisión $g(\vec{x}) = \vec{w}^t\vec{x} + w_0$.

donde $\vec{x}_p$ es la proyección normal de $\vec{x}$ en H y r es la distancia deseada, positiva si $\vec{x}$ cae en el lado positivo y negativa en caso contrario. Siendo $g(\vec{x}_p) = 0$, tenemos[10]:

$$\begin{aligned} g(\vec{x}) &= \vec{w}^t(\vec{x}_p + r\frac{\vec{w}}{||\vec{w}||}) + w_0 \\ &= \vec{w}^t\vec{x}_p + \vec{w}^t r\frac{\vec{w}}{||\vec{w}||} + w_0 \\ &= g(\vec{x}_p) + \frac{r\vec{w}^t\vec{w}}{||\vec{w}||} \\ &= r||\vec{w}|| \\ \Rightarrow r &= \frac{g(\vec{x})}{||\vec{w}||} \end{aligned}$$

En particular, la distancia del origen a H es $\frac{w_0}{||\vec{w}||}$. Si $w_0 > 0$ el origen cae en la parte positiva de H, y en la negativa si $w_0 < 0$. Si $w_0 = 0$, el hiperplano pasa por el origen (ver figura 5.15). En resumen, una FD lineal divide el espacio de entrada mediante un hiperplano –cuya orientación y posición vienen dadas por $\vec{w}$ y w_0, respectivamente– y su valor $g(\vec{x})$ es positivo si y sólo si $\vec{x}$ está en la parte positiva, siendo proporcional a la distancia (con signo) de $\vec{x}$ al hiperplano.

Supongamos ahora que disponemos de un conjunto de ejemplos $\vec{x}_1, \ldots, \vec{x}_n$ (donde unos pertenecerán

[10] Recuérdese que $||\vec{w}|| = \sqrt{\vec{w}^t\vec{w}}$.

a c_1 y otros a c_2) y pretendemos utilizarlos para calcular el vector de pesos $\vec{w}$ de una FD lineal del estilo $g(\vec{x}) = \vec{w}^t\vec{x}$, sin pérdida de generalidad. Si tal vector existe, diremos que los ejemplos son separables linealmente.

Así, un ejemplo $\vec{x}_i$ será clasificado correctamente si $g(\vec{x}_i) > 0$ y su clase era c_1, o bien si $g(\vec{x}_i) < 0$ y su clase era c_2. Podemos simplificar bastante esta expresión cambiando el signo de todos los ejemplos pertenecientes a c_2, de manera que nuestro objetivo sea siempre encontrar aquel $\vec{w}$ tal que $\vec{w}^t\vec{x}_i > 0$. Este vector es un punto en el *espacio de pesos*, donde cada ejemplo $\vec{x}_i$ es una restricción de este espacio.

Además, la ecuación $\vec{w}^t\vec{x}_i = 0$ define un hiperplano que pasa por el origen del espacio de pesos y tiene a $\vec{x}_i$ como vector normal. Así, el vector solución –si existe– debe estar en la cara positiva de cada hiperplano y, por tanto, en la intersección de n medios espacios, que define la *región solución*. Por otro lado, este vector no es único: cualquiera que caiga dentro será solución. El ideal sería encontrar aquél más "en el medio" de la región solución. De todas maneras, nos conformaremos con evitar que el proceso iterativo de búsqueda nos lleve hacia uno de los límites de la región, problema que puede ser resuelto introduciendo un *margen b* tal que

$$\forall i \quad \vec{w}^t\vec{x}_i \geq b > 0. \tag{5.3}$$

5.7.2 Descenso de gradientes

El método que seguiremos para encontrar una solución al conjunto de inecuaciones lineales $\vec{w}^t\vec{x}_i$ será definir una *función de criterio* $J(\vec{w})$, que sea mínima si $\vec{w}$ es un vector solución. Esta manera de proceder tiene una ventaja: se reduce el problema al de minimizar una función escalar, que normalmente puede resolverse mediante el método de descenso del gradiente. El proceso es el siguiente:

1. Empezar con un vector de pesos arbitrario $\vec{w}_1$ y computar el vector gradiente $\nabla J(\vec{w}_1)$.
2. El valor $\vec{w}_{k+1}$ se obtiene moviéndose cierta distancia desde $\vec{w}_k$ en la dirección de descenso más pronunciada, es decir, a lo largo del negativo del gradiente:

$$\vec{w}_{k+1} = \vec{w}_k - \rho_k \nabla J(\vec{w}_k), \tag{5.4}$$

   donde ρ_k es un factor de escala que establece el tamaño del paso.

Se espera así que $\vec{w}_k$ tienda al vector solución. El problema más grave con el que nos habremos de enfrentar será la elección de ρ_k. Si éste es demasiado pequeño, la convergencia será innecesariamente lenta, mientras que si es demasiado grande el proceso de corrección puede ir dando bandazos e incluso llegar a diverger.

5.7.3 La función de criterio del perceptrón

Vayamos ahora a la construcción de una función de criterio para las desigualdades $\vec{w}^t\vec{x}_i$. Si denominamos $M(\vec{w})$ al conjunto de ejemplos mal clasificados por $\vec{w}$, una función obvia puede venir dada por:

$$J(\vec{w}) = \#M(\vec{w}),$$

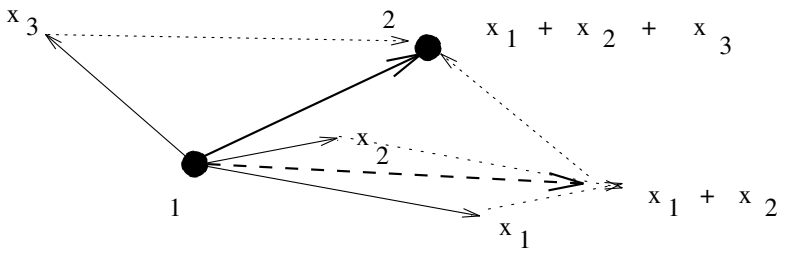


Figura 5.16: Primer paso de búsqueda.

donde el operador # denota la cardinalidad de un conjunto. Ésta es una mala función para nuestros propósitos, pues puede llegar a dar valores en exceso similares o constantes para diferentes vectores. Una elección mejor es la *función de criterio del perceptrón*:

$$J_P(\vec{w}) = \sum_{\vec{x} \in M(\vec{w})} (-\vec{w}^t \vec{x}), \tag{5.5}$$

función que –al ser $\vec{w}^t \vec{x} \leq 0$ para los $\vec{x}$ mal clasificados– nunca es negativa, siendo igual a cero en los siguientes casos:

1. Si $\vec{w}$ es un vector solución, esto es, si $M(\vec{w}) = \emptyset$.
2. Si $\vec{w}$ está en el límite de la región solución.

Geométricamente, $J_P(\vec{w})$ es proporcional a la suma de distancias de los ejemplos mal clasificados al límite de la región solución. El componente j del gradiente de J_P es $\partial J_P / \partial w_j$ y, por tanto, de (5.5) se obtiene:

$$\nabla J_P(\vec{w}) = \sum_{\vec{x} \in M(\vec{w})} (-\vec{x}),$$

con lo que el algoritmo de descenso del gradiente (5.4) queda:

$$\vec{w}_{k+1} = \vec{w}_k - \rho_k \nabla J_P(\vec{w}_k) = \vec{w}_k + \rho_k \sum_{\vec{x} \in M(\vec{w}_k)} \vec{x}.$$

En otras palabras, para encontrar el vector solución se va añadiendo cada vez una cantidad proporcional a la suma de los ejemplos mal clasificados. Las figuras 5.16 y 5.17 muestran de manera intuitiva un sencillo ejemplo para dos dimensiones, para el que se tiene $\vec{w}_1 = \vec{0}$ y $\rho_k = 1$.

5.7.4 Convergencia del cálculo

Demostraremos la convergencia de esta función de criterio. Emplearemos para ello una variante más sencilla de analizar, con las siguientes convenciones:

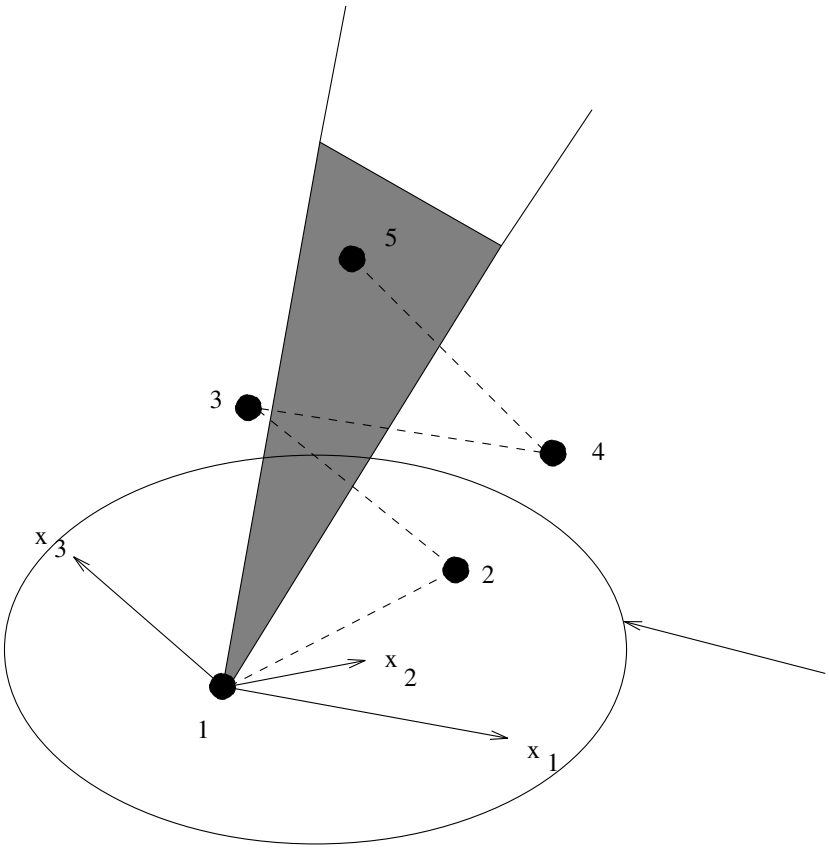


Figura 5.17: Búsqueda de un vector solución mediante el descenso del gradiente.

- En vez de calcular $M(\vec{w}_k)$ cada vez, consideraremos los ejemplos en forma de secuencia modificando el vector de pesos siempre que clasifique mal uno de ellos. A efectos de la prueba, esto es irrelevante siempre que todos los ejemplos aparezcan en la secuencia tantas veces como se desee (la manera más fácil es repetirlos cíclicamente).

- Nos concentraremos en el caso de *incremento fijo*, es decir, cuando ρ_k es constante. Tomaremos $\rho_k = 1$ sin perder generalidad.

- Ya que sólo los ejemplos mal clasificados intervienen en el cálculo, trabajaremos directamente con secuencias de ellos, denotándolos por superíndices, de la forma:

$$\vec{x}^1, \vec{x}^2, \vec{x}^3, \ldots, \vec{x}^2, \vec{x}^4, \vec{x}^1, \ldots$$

  En concreto, para el ejemplo de las figuras 5.16 y 5.17 la secuencia sería:

$$\vec{x}^1, \vec{x}^2, \vec{x}^3, \vec{x}^3, \vec{x}^1, \vec{x}^3$$

Así, podemos escribir la regla de incremento fijo como:

$$\vec{w}_{k+1} = \begin{cases} \vec{w} \text{ arbitrario} & \text{si } k = 0 \\ \vec{w}_k + \vec{x}^k & \text{si } k \geq 1 \end{cases} \tag{5.6}$$

donde $\forall k \ \vec{w}_k^t \vec{x}^k \leq 0$. Esta regla es la más simple de entre las propuestas para resolver sistemas de inecuaciones lineales y la demostración de su convergencia es el ya presentado *Teorema de convergencia del perceptrón*. Su interpretación geométrica es clara: si $\vec{w}_k$ clasifica mal $\vec{x}^k$, $\vec{w}_k$ no puede estar en la cara positiva del hiperplano $\vec{w}_k^t \vec{x}^k = 0$, con lo que el hecho de sumar $\vec{x}^k$ a $\vec{w}_k$ mueve el vector de pesos directamente hacia (y quizá a través de) el hiperplano. Así, el nuevo producto $\vec{w}_{k+1}^t \vec{x}^k$ es *mayor* que $\vec{w}_k^t \vec{x}^k$, siendo la diferencia $\|\vec{x}\|^2$.

Demostraremos ahora que, si los ejemplos son linealmente separables, la secuencia de vectores peso acabará en un vector solución. La manera intuitiva de hacerlo –y la que seguiremos– es probar que cada corrección lleva el vector en curso más cerca de la región solución. Formalmente, si $\hat{w}$ es un vector solución, entonces deberá cumplirse la relación $\|\vec{w}_{k+1} - \hat{w}\| < \|\vec{w}_k - \hat{w}\|$. En realidad, en general sólo será cierto para vectores solución suficientemente largos en módulo.

Sea $\hat{w}$ un vector solución (con lo que $\forall i \ \hat{w}^t \vec{x}_i > 0$), y sea $\alpha > 0$ un factor de escala. De (5.6):

$$(\vec{w}_{k+1} - \alpha\hat{w}) = (\vec{w}_k - \alpha\hat{w}) + \vec{x}^k$$

con lo que

$$\|\vec{w}_{k+1} - \alpha\hat{w}\|^2 = \|\vec{w}_k - \alpha\hat{w}\|^2 + 2(\vec{w}_k - \alpha\hat{w})^t \vec{x}^k + \|\vec{x}^k\|^2.$$

Como $\vec{x}^k$ está mal clasificado, $\vec{w}_k^t \vec{x}^k \leq 0$ y, por tanto,

$$\|\vec{w}_{k+1} - \alpha\hat{w}\|^2 \leq \|\vec{w}_k - \alpha\hat{w}\|^2 - 2\alpha\hat{w}^t \vec{x}^k + \|\vec{x}^k\|^2.$$

Si ahora definimos

$$\left.\begin{array}{ll} \beta^2 & = \underset{i}{Max} \ \|\vec{x}_i\|^2 \\ \gamma & = \underset{i}{Min} \ \hat{w}^t \vec{x}_i > 0 \end{array}\right\},$$

entonces

$$-2\alpha\hat{w}^t \vec{x}^k + \|\vec{x}^k\|^2 \leq -2\alpha\hat{w}^t \vec{x}^k + \beta^2 \leq -2\alpha\gamma + \beta^2$$

y, si definimos $\alpha = \beta^2/\gamma$:

$$\|\vec{w}_{k+1} - \alpha\hat{w}\|^2 \leq \|\vec{w}_k - \alpha\hat{w}\|^2 - \beta^2.$$

Esto significa que la distancia se va reduciendo al menos en β^2 en cada paso y, al cabo de k pasos:

$$\|\vec{w}_{k+1} - \alpha\hat{w}\|^2 \leq \|\vec{w}_1 - \alpha\hat{w}\|^2 - k\beta^2.$$

Al no ser nunca negativo el cuadrado de la distancia se sigue que, al cabo de no más de k_0 correcciones, la secuencia debe llegar a su fin, siendo

$$k_0 = \frac{\|\vec{w}_1 - \alpha\hat{w}\|^2}{\beta^2}.$$

Por consiguiente, al darse cada corrección en caso de un ejemplo mal clasificado y aparecer éstos un número infinito de veces en la secuencia, se deduce que el vector resultante los clasifica todos correctamente.

El número k_0 proporciona un límite superior del número de pasos necesario. En particular, con la simplificación $\vec{w}_1 = \vec{0}$ se tiene:

$$k_0 = \frac{\alpha\|\hat{w}\|^2}{\beta^2} = \frac{\beta^2\|\hat{w}\|^2}{\gamma^2} = \frac{\underset{i}{Max}\,\|\vec{x}_i\|^2\|\hat{w}\|^2}{\underset{i}{Min}^2\,\hat{w}^t\vec{x}_i}.$$

Desafortunadamente, no da ninguna idea en el caso de no conocer ningún $\hat{w}$ *a priori*.

Dos variantes (de hecho, generalizaciones) de la regla de incremento fijo son merecedoras de mención, aunque sólo sea de pasada: la regla de *incremento variable* y la regla de *relajación*. La primera, como su nombre indica, introduce un incremento que depende de k y un margen b y realiza una corrección siempre que $\vec{w}_k^t\vec{x}^k$ no lo exceda. La regla es la siguiente:

$$\vec{w}_{k+1} = \begin{cases} \vec{w} \text{ arbitrario} & \text{si } k = 0 \\ \vec{w}_k + \rho_k\vec{x}^k & \text{si } k \geq 1 \end{cases}$$

donde $\forall i\ \vec{w}_i^t\vec{x}^i \leq b$. La elección de ρ_k ha de cumplir ciertos criterios simples. En general, basta con que sea positiva. Una buena elección es hacer que decrezca según $1/k$.

La segunda (relajación) utiliza un criterio de concepción distinta:

$$J_R(\vec{w}) = \sum_{\vec{x}\in M(\vec{w})} (\vec{w}^t\vec{x})^2$$

Su principal ventaja es que presenta un gradiente continuo y, por tanto, más suave. No obstante, es tan suave cerca de los los límites de la región solución que la secuencia de pesos tiene tendencia a converger hacia un sólo punto del borde (en particular, hacia $\vec{w} = \vec{0}$). Otro problema es que se ve muy afectada por vectores de ejemplo largos (en módulo).

Estos inconvenientes desaparecen con la función de criterio siguiente (que es simplemente una mejora de la anterior, por lo que mantendremos el nombre):

$$J_R(\vec{w}) = \frac{1}{2}\sum_{\vec{x}\in M(\vec{w})} \frac{(\vec{w}^t\vec{x} - b)^2}{\|\vec{x}\|^2}$$

Aquí, la resta de b se hace para evitar la mencionada aproximación a los límites de la región solución (recuérdese el concepto de *margen* de la página 176) y la normalización por $\|\vec{x}\|^2$ restringe la elección de ρ_k al intervalo abierto $(0, 2)$. En esta fórmula, $M(\vec{w})$ representa el conjunto de ejemplos tales que $\vec{w}^t\vec{x} \leq b$. Si $M(\vec{w}) = \emptyset$, se define $J_R(\vec{w}) = 0$. En esas condiciones, tenemos que $J_R(\vec{w})$ nunca es negativo, y es cero si, y sólo si, $\vec{w}^t\vec{x} \geq b$ para todos los ejemplos $\vec{x}$. El gradiente de $J_R(\vec{w})$ viene dado por:

$$\nabla J_R(\vec{w}) = \sum_{\vec{x}\in M(\vec{w})} \frac{\vec{w}^t\vec{x} - b}{\|\vec{x}\|^2}\vec{x}$$

con lo cual el algoritmo de descenso queda:

$$\vec{w}_{k+1} = \begin{cases} \vec{w} \text{ arbitrario} & \text{si } k = 0 \\ \vec{w}_k - \rho_k\nabla J_R(\vec{w}_k) & \text{si } k \geq 1 \end{cases}$$

La prueba de convergencia –junto con sus consideraciones adicionales– se puede encontrar en [DUDA73].

Todos los métodos vistos hasta el momento (incrementos fijo y variable, relajación) dan una manera de encontrar un vector solución si se dan ciertas condiciones (principalmente, separabilidad lineal), y son por ello utilizados cuando se tiene la casi completa certeza de que son aplicables con éxito. Ahora bien, dos son los problemas que limitan grandemente su uso:

1. No generalizan bien. El hecho de encontrar una solución para una muestra o subconjunto de ejemplos no garantiza que un nuevo ejemplo sea clasificado correctamente, pues quizá rompa la separabilidad lineal.

2. En el caso de no darse la separabilidad lineal, el método puede no acabar, al no haber ningún vector que clasifique correctamente todos los ejemplos, con lo que el hiperplano irá moviéndose de un lado a otro sin estabilizarse, no quedando más remedio que detener el proceso en un punto más o menos arbitrario, para el que existen ciertas técnicas de estimación.

Parece natural, pues, intentar modificar la función de convergencia de manera que se obtengan resultados aceptablemente buenos para todos los ejemplos en caso de no separabilidad lineal[11] y se mantengan los anteriores si ésta existe. Ello nos lleva a los métodos de mínimo error cuadrático.

5.7.5 Métodos de mínimo error cuadrático

Estos métodos –en contraste con los vistos hasta ahora– utilizan para el cómputo todos los ejemplos disponibles, bien clasificados o no. La diferencia fundamental es que, allá donde antes buscábamos vectores peso tales que hicieran los productos $\vec{w}^t\vec{x}_i$ positivos, ahora forzaremos que cumplan $\vec{w}^t\vec{x}_i = b_i$, donde los b_i representan constantes especificadas *a priori*. La ventaja es clara: hemos pasado a un sistema de ecuaciones lineales que, si bien es más restrictivo que uno de inecuaciones, también es mucho más fácil de resolver. Introduciremos para ello notación matricial.

Sea X una matriz $n \times d$, donde n es el número de ejemplos y d la dimensión del espacio donde nos movemos y, por tanto, su fila i es el vector $\vec{x}_i^t$. Sea también $\vec{b}$ el vector columna $\vec{b} = (b_1, \ldots, b_n)^t$. El problema es encontrar un vector $\vec{w}$ tal que:

$$X\vec{w} = \vec{b}.$$

Si X es cuadrada y no singular, se tiene la solución directa $\vec{w} = X^{-1}\vec{b}$. Desafortunadamente, en general X será no cuadrada[12], normalmente con $n \gg d$. En este caso (más ecuaciones que incógnitas) $\vec{w}$ está *sobredeterminado* y no suele haber solución exacta. No obstante, se puede hallar un vector $\vec{w}$ que minimice cierta función de error entre $X\vec{w}$ y $\vec{b}$. Podemos definir así un vector de error $\vec{e}$ de la forma:

$$\vec{e} = X\vec{w} - \vec{b},$$

[11] Obsérvese que insistimos en todo momento en separabilidad *lineal*. Un caso aparte y que cae fuera del propósito introductorio de esta sección es la separabilidad *cuadrática* (puede consultarse para ello [DUDA73]).

[12] Y, por tanto, no singular.

y hacer mínima la longitud al cuadrado de $\vec{e}$, que es equivalente a minimizar el llamado *criterio de la suma del error cuadrático*:

$$J_S(\vec{w}) = \|\vec{e}\|^2 = \|X\vec{w} - \vec{b}\|^2 = \sum_{i=1}^{n} (\vec{w}^t \vec{x}_i - b_i)^2.$$

Esto se puede realizar de nuevo mediante el descenso del gradiente, que analizaremos enseguida, y que da lugar a la regla delta. Antes, nos detendremos un instante en su análisis directo. El gradiente es:

$$\nabla J_S(\vec{w}) = \frac{\partial J_S(\vec{w})}{\partial \vec{w}} = \sum_{i=1}^{n} 2(\vec{w}^t \vec{x}_i - b_i)\vec{x}_i = 2X^t(X\vec{w} - \vec{b}),$$

el cual igualamos a cero, lo que da lugar a la condición necesaria:

$$X^t X\vec{w} = X^t \vec{b}, \tag{5.7}$$

habiendo convertido así el problema de resolver $X\vec{w} = \vec{b}$ a resolver $X^t X\vec{w} = X^t\vec{b}$. La diferencia está en que la matriz $(d \times d)$ $X^t X$ es cuadrada y, frecuentemente, no singular, lo que reduciría la tarea a calcular

$$\vec{w} = X^\star \vec{b},$$

de solución única, donde $X^\star = (X^t X)^{-1} X^t$ es la llamada matriz *pseudoinversa* de X que, si X es cuadrada y no singular, coincide con la inversa. Notar también que $X^\star X = I$ pero, en general, $XX^\star \neq I$.

Si $X^t X$ es singular la solución a (5.7) no es única, aunque siempre existe una solución que minimiza el error cuadrático. Por ejemplo, utilizando la definición alternativa de pseudoinversa

$$X^\star = \lim_{\epsilon \to \infty} (X^t X + \epsilon I)^{-1} X^t,$$

se puede demostrar que este límite existe siempre y que efectivamente $\vec{w} = X^\star \vec{b}$ es solución de $X\vec{w} = \vec{b}$. La única cuestión pendiente es la elección de $\vec{b}$. En realidad, éste es un vector de margen, que da lugar a aproximaciones diferentes según lo elijamos. No entraremos en detalles; lo único que interesa es saber que la minimización del error cuadrático es un método que trabaja bien tanto si X está compuesta de vectores linealmente separables como si no.

Pero volvamos a la tarea de encontrar un método de descenso del gradiente para minimizar $J_S(\vec{w}) = \|X\vec{w} - \vec{b}\|^2$. Si nos afanamos en encontrarlo es por tres razones:

- No presenta los problemas que aparecen cuando $X^t X$ es singular.
- Elimina la necesidad de cargar con matrices grandes.
- Su cálculo recurrente mejora la pérdida de precisión debida a truncamientos y redondeos cuando se implementa en un computador.

Directamente calculamos $\nabla J_S(\vec{w}) = 2X^t(X\vec{w} - \vec{b})$, obteniendo la regla de descenso siguiente:

$$\vec{w}_{k+1} = \begin{cases} \vec{w} \text{ arbitrario} & \text{si } k = 0 \\ \vec{w}_k - \rho_k X^t(X\vec{w}_k - \vec{b}) & \text{si } k \geq 1 \end{cases}$$

Se puede demostrar que, dada una constante inicial $\rho_1 > 0$, la relación $\rho_k = \rho_1/k$ genera una secuencia de vectores peso que converge a un $\vec{w}$ que satisface $X^t(X\vec{w} - \vec{b}) = 0$. Así, este algoritmo encuentra siempre una solución sea X^tX singular o no.

Una derivación final, que toma los ejemplos secuencialmente y no necesita tanto cómputo[13] es la siguiente:

$$\vec{w}_{k+1} = \begin{cases} \vec{w} \text{ arbitrario} & \text{si } k = 0 \\ \vec{w}_k + \rho_k(b_k - \vec{w}_k^t \vec{x}^k)\vec{x}^k & \text{si } k \geq 1 \end{cases}$$

que no es ni más ni menos que la regla de Widrow-Hoff o regla delta. Nótese que la regla del perceptrón es una especialización, donde los dos valores entre paréntesis son binarios (b_k por definición y $\vec{w}_k^t \vec{x}^k$ a causa de la función de corte).

En la mayor parte de los casos no es posible satisfacer todas las ecuaciones $\vec{w}_k^t \vec{x}^k = b_k$, con lo que $\vec{w}_k$ puede no estabilizarse. La solución es jugar con el factor ρ_k de manera que decrezca con k. Una posibilidad –ya vista– muy común es tomar $\rho_k = \rho_1/k$. Todos estos métodos se pueden generalizar al caso de convergencia hacia vectores, simplemente considerando el hasta ahora vector $\vec{b}$ como una matriz B. Obsérvese que en este caso ya no hay una analogía directa con la función realizada por una simple neurona, pues ésta tiene como salida un valor escalar, normalmente binario, sino que estaríamos hablando ya de una red monocapa.

5.8 El algoritmo de Backpropagation

Nos centraremos ahora en el estudio de las redes multicapa. La manera de calcular la salida de estas redes ya fue vista en 5.4.3. Allí se expresó también la idea de que sólo representan un incremento de capacidad de aprendizaje respecto a las monocapa si alguna capa introduce funciones de activación no lineales. La función sigmoide proporciona esta no linealidad aunque, en general, tan sólo se necesita que sea diferenciable en cualquier punto. Asimismo, el valor inicial de los pesos puede establecerse de manera aleatoria, siempre que no sean excesivamente grandes, para prevenir saturaciones. También deben ser diferentes, ya que existen problemas que requieren valores iniciales diferentes a riesgo de que la red no converja.

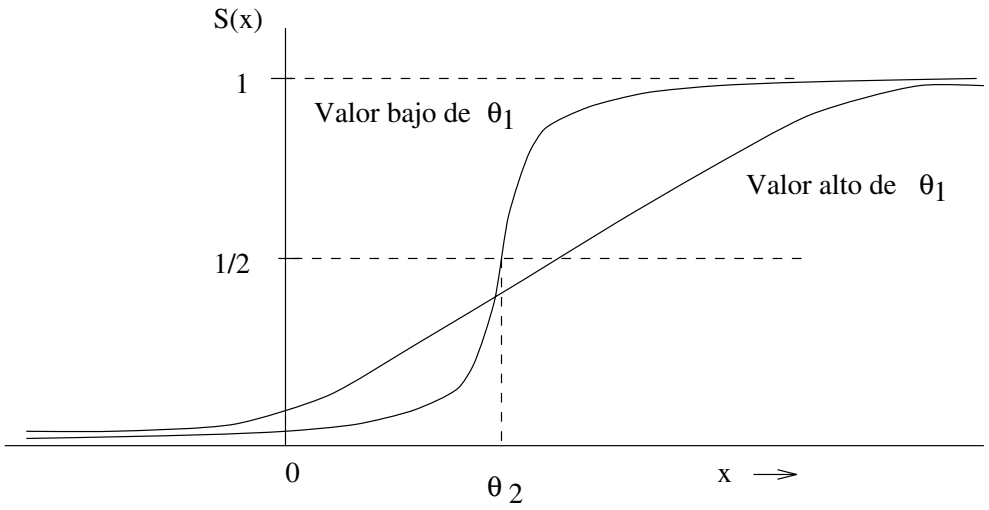
Lo único que falta por ver es cómo generalizar los métodos de aprendizaje vistos hasta el momento para que puedan trabajar con capas ocultas. Ahora bien, la función de corte o lindero no es buena para nuestros propósitos (para el método del gradiente) pues, al no ser continua, el método no puede aplicarse bien dado que hay muchos altibajos, lo que hace difícil establecer pequeños cambios en los pesos. En cambio, la función sigmoide (figura 5.3) tiene como ventaja –aparte de la ya mencionada en cuanto a ganancia (ver 5.4.1)– que es *suave*, y diferenciable en todos los puntos, cumpliendo además la relación:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5.8}$$

$$\frac{\mathrm{d}S}{\mathrm{d}x} = S(x)(1 - S(x)) \tag{5.9}$$

Además, un vistazo a la derivada (figuras 5.3 y 5.18) muestra que tiene su máximo para $S(x) = 1/2$ y sus mínimos para $S(x)$ acercándose a 0 o a 1 que, por otro lado, son los

[13] Nótese la disminución en el cálculo y volumen de datos necesitados. La matriz $X^\star$ es $d \times n$, X^tX es ya $d \times d$ y esta última regla involucra solamente productos vectoriales.



$$S(x) = \frac{1}{1 + e^{-(x+\theta_2)/\theta_1}} \tag{5.10}$$

Figura 5.18: Función sigmoidal con control de pendiente y desplazamiento.

valores límite de la neurona. Como veremos, el incremento en un peso será proporcional a esta derivada, con lo que se conseguirá un mayor cambio para aquellas neuronas en estado "dudoso" (cerca de 0 en valor absoluto) forzándolas a decantarse hacia uno u otro límite, y menor para aquellas que ya tengan un valor consolidado (referenciamos de nuevo a 5.4.1). Siguiendo con la figura 5.1, vemos cómo es posible también definir a gusto el comportamiento de la función, introduciendo parámetros de control θ_1 y θ_2. Nótese que éste último ya lo hemos introducido con anterioridad (es el lindero entrenable) y se puede considerar parte del sumatorio.

5.8.1 El algoritmo

El algoritmo de *backpropagation* 5.8 [RUME86a] (propagación hacia atrás) generaliza el método de corrección del error calculado hasta ahora, de manera que pueda ser aplicado también a capas ocultas, donde no se dispone *a priori* de la salida correcta (ésta sólo se conoce, en principio, para la capa de salida). El algoritmo puede sintetizarse de la manera siguiente:

1. Seleccionar el siguiente par de entrenamiento y aplicar el vector de entrada a la red.
2. Calcular la salida de ésta.
3. Calcular el error entre la salida real y la esperada de la red, y propagarlo hacia atrás (es decir, en sentido contrario al utilizado en el paso anterior) a través de los pesos y modificando éstos de manera que se minimice el error que tendría la misma entrada aplicada de nuevo.

El proceso se repite hasta que la red se comporta de una manera considerada satisfactoria (el error para cada par de entrenamiento es soportable). Los pasos 1 y 2 son los que se han venido realizando hasta ahora.

Cada capa transmite sus resultados a la siguiente de la manera usual: tómese una capa cualquiera k, su matriz de pesos W_k y la salida de la capa $k-1$, $\vec{y}_{k-1}$. Así, se tiene que $\vec{y}_k = f(W_k \vec{y}_{k-1})$, siendo f la función de activación. Si la red dispone de m capas, y denotamos el vector de entrada como $\vec{x}$, el resultado $\vec{y}_m$ de la red puede definirse como:

1. $\vec{y}_0 = \vec{x}$
2. $\vec{y}_m = f(W_m \vec{y}_{m-1})$

donde todos los $\vec{x}, \vec{y}$ son vectores columna.

El paso crucial, naturalmente, es el de ajuste de pesos, para el cual conocemos métodos de aprendizaje para los pesos de la capa de salida. Ahora bien, ¿ cómo se calcula el error para las capas ocultas? La respuesta es que se aprovecha el error de la capa siguiente, propagándolo hacia atrás, de manera que se multiplique el error proveniente de cada neurona con el peso que la conecta a la capa precedente.

Expresaremos la salida de una neurona en dos partes: el cálculo propiamente dicho, y la aplicación de la función de activación f que, más adelante, substituiremos por S. Así,

$$y'_j = \sum_i w_{ij} y_i \tag{5.11}$$

$$y_j = f(y'_j) \tag{5.12}$$

De nuevo denotaremos por $\hat{y}_j$ la salida *esperada* de la neurona j, aunque sólo será conocida para las de la capa de salida. Mostraremos cómo se llega a las fórmulas de actualización de pesos de manera constructiva. Para ello, recordemos dos de los elementos clave vistos hasta el momento:

Descenso de gradientes (§5.7). Sea $f(x_1, x_2, \ldots, x_n)$ una función escalar de varias variables. Si se quiere minimizar ésta, a base de cambios en sus variables, el incremento de cada una será proporcional a la derivada de la función respecto a ella. Formalmente,

$$\Delta x_i \propto \frac{\partial f}{\partial x_i}$$

Es obvio que así se desciende por la coordenada de pendiente más pronunciada.

Medida del error (§5.6). El error total es la suma de los cuadrados de los errores para todos los ejemplos de entrada:

$$E = \frac{1}{2} \sum_p \sum_j (\hat{y}_{pj} - y_{pj})^2 .$$

El hecho de utilizar esta medida y no otra (p.e. una simple suma de los errores) es debido a que –como se verá– se deja tratar muy fácilmente desde el punto de vista matemático.

Para nuestros propósitos estas variables son los pesos, y la función a minimizar es el error que, por supuesto, depende de ellos. Así, nuestra tarea será calcular la derivada parcial del error con respecto a un peso cualquiera de la red, y ver cómo se ha de alterar éste para minimizar aquél. Formalmente, tendremos que

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}.$$

Para empezar, nótese que E es una suma de los errores para cada ejemplo de entrada. Al ser la derivada de una suma igual a la suma de las derivadas, podemos concentrarnos en un ejemplo cualquiera, lo que simplificará la notación. Además –más importante, y por la misma razón– si realmente queremos hacer un descenso del gradiente para minimizar el error, todos los cambios en un peso (debido cada uno a la presentación de un ejemplo) han de realizarse *de una sola vez* al acabar un *epoch*[14] (y no inmediatamente después de cada ejemplo). Si no se hace así, E deja de ser una función a minimizar por su gradiente, pasando a ser un mero "indicador" de la red. Otra opción sería tomar directamente como función a minimizar el error para un solo ejemplo[15].

Calculemos, pues, la derivada parcial

$$\frac{\partial E}{\partial w_{ij}}$$

donde w_{ij} es el peso de la conexión entre la salida de una neurona cualquiera i y una de las entradas de otra j. Para ello la podemos descomponer –usando la regla de la cadena– sabiendo que E está en función de y_j. Así,

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial y'_j}\frac{\partial y'_j}{\partial w_{ij}}.$$

Vayamos por partes. Para calcular $\partial y'_j/\partial w_{ij}$ ya sabemos cómo depende una de la otra (ecuación (5.11)) y podemos volver a aplicar la regla de la cadena para obtener:

$$\frac{\partial y'_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\sum_i w_{ij} y_i) = y_i$$

donde además

$$\frac{\partial y_j}{\partial y'_j} = \frac{\mathrm{d}f(y'_j)}{\mathrm{d}y'_j} = \frac{\mathrm{d}S(y'_j)}{\mathrm{d}y'_j} = S(y'_j)(1 - S(y'_j)) = y_j(1 - y_j)$$

Para calcular ahora $\partial E/\partial y_j$ basta con darse cuenta que E depende primeramente de las salidas de las neuronas más cercanas a la capa de salida, antes que de las más lejanas. Por tanto, suponiendo que j no sea ya una neurona de la salida, existirán otras k más cercanas a la salida –tal y como lo hemos dibujado, más a la derecha– que utilizarán y_j. En otras palabras, y_k es función de todas las y_j de sus entradas. Este razonamiento nos permite expresar los cálculos en función de otros ya hechos:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y'_k}\frac{\partial y'_k}{\partial y_j}$$

[14] A la presentación de todo el conjunto con fines de entrenamiento (es decir, modificando pesos) se le denomina *epoch*.

[15] De hecho, hay controversia sobre cuál es la mejor manera. Nosotros seguiremos la opción ya descrita.

Ahora bien, también sabemos cómo se obtiene y'_k a partir de sus entradas (ecuaciones (5.11) y (5.12)), con lo que tenemos:

$$\frac{\partial y'_k}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_i w_{ik} y_i = w_{jk}$$

Uniéndolo todo, y definiendo el error parcial ε_j como $\varepsilon_j = \partial E / \partial y_j$, obtenemos finalmente

$$\frac{\partial E}{\partial w_{ij}} = y_i \varepsilon_j = y_i y_j (1 - y_j) \sum_k \varepsilon_k w_{jk}$$

lo que indica que la derivada parcial de E respecto de un peso cualquiera en una capa se escribe en términos de cálculos ya hechos para capas más cercanas a la salida. Lo único que resta es encontrar una expresión para, precisamente, la capa de salida, pues obviamente no podrá depender de otras capas más a la derecha. La *diferencia* es que conocemos la salida *esperada* de esa capa con lo que el primer error ε_s –siendo s una neurona de salida– será:

$$\varepsilon_s = \frac{\partial E}{\partial y_s} = \frac{1}{2} \frac{\partial}{\partial y_s} (\hat{y}_s - y_s)^2 = -(\hat{y}_s - y_s).$$

Así, se llega a las siguientes fórmulas de *backpropagation*:

$$\Delta w_{ij} = \alpha y_i \varepsilon_j ,$$

con

$$\varepsilon_j = \begin{cases} y_j (1 - y_j) \sum_k w_{jk} \varepsilon_k , & \text{si } j \text{ es oculta} \\ (\hat{y}_j - y_j) y_j (1 - y_j) & \text{si } j \text{ es de salida} \end{cases}$$

Éste es el algoritmo básico. Es interesante notar que, dado el carácter de la función sigmoidal, la salida de una neurona nunca llegará efectivamente a 0 ó 1 (harían falta pesos infinitamente positivos o negativos). Normalmente, por tanto, nos conformaremos con valores cercanos a 0.1 y 0.9, respectivamente. También se puede observar que no hemos hecho más que obtener una *regla delta generalizada*, simplemente estableciendo (según notación de 5.4.5) la relación:

$$r_{ij} = y_i \varepsilon_j .$$

Como se mencionó al inicio, la manera natural de modificar pesos es hacerlo tras cada *epoch*, a riesgo de no descender por la pendiente más pronunciada. Hay, no obstante, una manera de realizar las modificaciones tras cada ejemplo, y consiste en jugar con el parámetro α. Si éste es suficientemente pequeño, los cambios también lo serán y nos aproximaremos a la solución. Ello nos lleva al problema de la elección de α. Una muy pequeña hará extremadamente lento el proceso, mientras que una muy grande lo hará rápido pero podría causar oscilaciones. Se ha propuesto una solución a este problema, consistente en la introducción de un parámetro adicional η denominado *momentum*, con la idea de tener en cuenta los incrementos pasados de cara a los futuros:

$$\Delta w_{ij}(t+1) = \eta \Delta w_{ij}(t) + \alpha y_i \varepsilon_j$$

De este modo se intenta mantener una cierta inercia en el sistema, habiéndose probado experimentalmente que una buena política es usar conjuntamente α y η grandes.

5.8.2 Estudio cualitativo

El algoritmo básico –y sus derivaciones– se ha aplicado con cierto éxito a diversos problemas, entre los cuales merece la pena destacar los siguientes:

- La NEC japonesa realizó un sistema óptico reconocedor de caracteres mediante la combinación de *backpropagation* con algoritmos convencionales. Se anunció una precisión del 99%.
- Sejnowski y Rosenberg crearon NetTalk [SEJN86], un sistema que convertía texto inglés escrito en hablado, de apariencia bastante impresionante.
- Cottrell , Munro y Zipser desarrollaron un compresor de imágenes, capaz de reducir por un factor de ocho [COTT87].

Su vasta utilización es debida fundamentalmente a su sencillez y a su solidez teórica. No obstante, presenta otras características que no son tan obvias si uno no lo estudia en profundidad. Las separaremos en dos grupos. Entre las características buenas podemos encontrar:

1. Se puede efectuar en fases separadas, esto es, se pueden ir añadiendo neuronas de manera incremental, aplicando entonces el algoritmo a los nuevos pesos (para que se creen) y a los viejos (para que se adapten). En ciertos casos, el número total de pasos es substancialmente menor.
2. El algoritmo es capaz de reconocer varios conceptos simultáneamente. Esto no es extraño, pues cada nodo de la salida puede representar ya un concepto diferente. A veces, incluso, es mejor partir ya de una red con todos los nodos de salida que no irlos añadiendo poco a poco, como se ha comentado en el punto anterior.
3. La red puede *predecir*. Supóngase que no se utiliza todo el conjunto de pares de entrenamiento en el algoritmo, sino sólo una parte. Entonces el resto (digamos, un 20-25%) se reserva para poner a prueba la red, a ver si trabaja correctamente, dado que conocemos la respuesta para estos casos.

Las desventajas, sin embargo, son bastante fuertes, y se pueden resumir en cuatro fundamentales:

Saturación. A medida que los pesos se van ajustando, es posible que lleguen a alcanzar valores muy altos, donde la derivada de la sigmoide es pequeña. Esto provoca que los errores propagados hacia atrás sean también pequeños (pues son proporcionales a la derivada), con lo que el proceso entero deviene estacionario. Se puede intentar evitar con valores bajos del parámetro η, aunque esto provoca un entrenamiento extremadamente lento. A este problema se le conoce también por *parálisis de la red.*

Mínimos locales. Como se ha visto, el algoritmo realiza un descenso del gradiente, siguiendo la pendiente más pronunciada en busca del mínimo. El caso es que la superficie del espacio de error está compuesta de múltiples valles, colinas, etc, siendo bastante posible que la red quede "atrapada" en un valle que no sea el más profundo (es decir, un mínimo local de la función) ya que, desde ese punto, todos los cambios son de "subida". Existen

métodos estadísticos combinados utilizando la máquina de Cauchy que resuelven este problema.

Inestabilidad. Entrenando una red, es de desear que el hecho de aprender un concepto (por ejemplo, la letra 'A'), no provoque el "olvido" de otro (la letra 'B'). Esto ocurre si no se van presentando ejemplos de vez en cuando, lo que, traducido al mundo real, significa que el algoritmo no es bueno si se requiere de él que trabaje en entornos cambiantes, donde un par de entrenamiento ya presentado puede no volver a darse nunca más, con lo cual el proceso de entrenamiento nunca convergerá[16].

Tiempo alto de entrenamiento y elección difícil del tamaño del paso. Para alcanzar la precisión deseada se requerirían cambios infinitesimales y, por ende, un tiempo infinito de entrenamiento. Ello provoca –como se vió– la elección de un factor de paso, con los problemas que esto conlleva: si es demasiado grande, puede provocar saturación o inestabilidad, mientras que si es demasiado pequeño, la convergencia puede resultar muy lenta.

Un aspecto aparte, pero a tener en cuenta, es la disposición inicial: número de nodos, topología, pesos iniciales y parámetros del algoritmo. Existen algunas "recetas" pero, como en gastronomía, no pasan de ahí. Hay que construir la red y ejecutar el algoritmo probando varias combinaciones hasta encontrar una que nos satisfaga.

5.9 El algoritmo de Counterpropagation

Este algoritmo (contrapropagación) fue desarrollado por Robert Hecht-Nielsen [HECH87], y está basado en trabajos previos de Kohonen y Grossberg . Su arquitectura es capaz de asociar un vector de entrada arbitrario a uno de salida, actuando como una *look-up table* con capacidad de generalización. Esta última permite obtener una salida correcta incluso si la entrada sólo lo es parcialmente, o está incompleta. Consta de dos capas, a las que denominaremos capa Grossberg (con matriz de pesos G) y capa Kohonen (con matriz de pesos K), dispuestas según la figura 5.19. En ella se muestra también –pese a que, como se vió, no es en sí una capa como las demás– la capa de entrada, pues facilita la discusión.

Las salidas de cada capa se calculan de la manera usual, esto es, producto escalar de entradas y pesos. Siguiendo las convenciones del apartado 5.4.2, la salida vectorial de la capa Kohonen sería $\vec{k} = K\vec{x}$, siendo $\vec{x}$ un vector de entrada. La diferencia aquí reside en la función especial que se aplica a esta salida. Así, la neurona con salida mayor es tomada como valor 1. El resto son tomadas como 0. El vector que finalmente calcula la red es $G\vec{k}$, paso que –después de pasar por la función ya comentada– se reduce a devolver, para cada neurona de G, aquel peso que la conecta a la única neurona de K diferente de 0.

5.9.1 Entrenamiento de la red

El entrenamiento se lleva a cabo en dos fases y mediante dos algoritmos diferentes. Veámoslas en detalle:

[16] Aquí es interesante remarcar la gran implausibilidad biológica de este método.

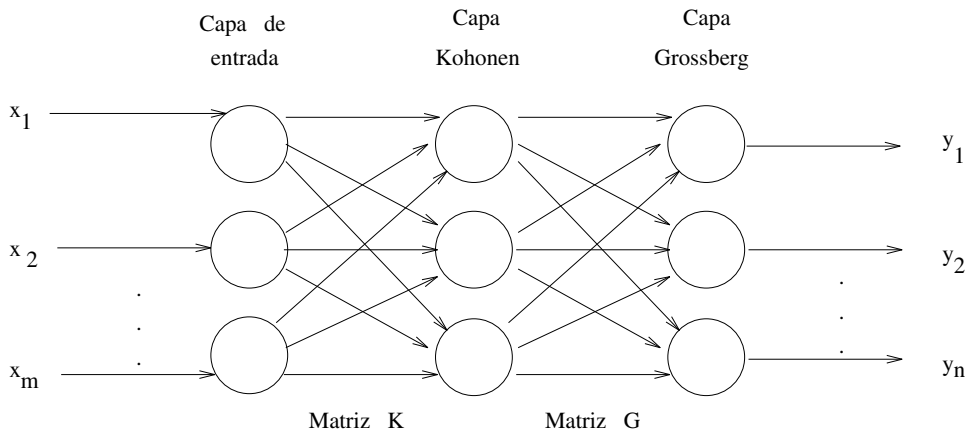


Figura 5.19: Arquitectura de contrapropagación.

Fase 1: consiste en modificar los pesos de la matriz K, utilizando los vectores de entrada disponibles y el algoritmo generalmente conocido como *Kohonen learning* (aprendizaje Kohonen). Éste es un método no supervisado, esto es, solamente se aplican a la red vectores de entrada (sin sus correspondientes salidas deseadas) de manera que una o varias neuronas de la capa Kohonen se activen. El algoritmo es el siguiente:

1. Aplicar un vector de entrada $\vec{x}$.
2. Calcular el vector $\vec{k} = K\vec{x}$, salida de la capa Kohonen.
3. Sea m la neurona con el mayor producto escalar, es decir, $k_m = max(k_1, k_2, \ldots)$.
4. Ajustar el vector de pesos de la neurona m según la fórmula:

$$K_m(t+1) = K_m(t) - \alpha(\vec{x} - K_m(t))$$

donde $\alpha < 1$ es la razón de aprendizaje, que se va reduciendo a medida que avanza el entrenamiento.

El proceso se repite hasta que se crea conveniente. Obsérvese que este algoritmo *clasifica* los vectores de entrada en base a su grado de similitud. Esto se realiza ajustando los pesos de K de manera que vectores de entrada similares activen la misma neurona de K, siendo la medida de similitud el producto escalar[17],y haciendo que vectores similares lo sean todavía más. La capa Grossberg se encargará luego de seleccionar la salida correcta. Nótese también que, al ser éste un algoritmo no supervisado, no se puede saber (ni interesa) cuál será la neurona de la capa Kohonen que se vaya a activar en cada momento. Tan sólo es necesario asegurarse de que el entrenamiento separa entradas diferentes.

Fase 2: modificar la matriz de pesos G mediante el algoritmo *Outstar* de Grossberg . Éste es supervisado, requiriendo por tanto pares completos de entrenamiento (entradas $\vec{x}$ y salidas $\vec{y}$), y consta de los siguientes pasos:

[17] Es conveniente preprocesar los vectores de entrada mediante una normalización, consistente simplemente en su cociente por el módulo, obteniendo así vectores unitarios.

1. Aplicar a la vez un vector de entrada $\vec{x}$ y uno de salida $\vec{y}$.
2. Calcular, de igual manera que en la fase 1, el vector $\vec{k}$ y la neurona m.
3. Ajustar los pesos G_{mi} entre la neurona m y todas las de la capa G según la fórmula:

$$G_{mi}(t+1) = G_{mi}(t) + \beta(y_i - G_{mi}(t))$$

donde β es la razón de aprendizaje, que se suele inicializar a 0.1 y se va reduciendo a medida que avanza el entrenamiento.

De esta manera, los pesos de la capa Grossberg tienden al valor medio de las salidas deseadas, mientras que los de la capa Kohonen lo hacen hacia el de las entradas. Así, esta última produce una salida eligiendo una neurona determinada, que es asociada con la salida correcta por la capa Grossberg.

5.9.2 Red completa de contrapropagación

Enseguida pasaremos a explicar el porqué de este nombre. En la figura 5.20 se muestra la red completa. Aquí, como en la gran mayoría de las redes, existen dos modos de funcionamiento: el modo *normal*, en que simplemente se calcula el resultado de la red a la vista de una entrada, y el de *entrenamiento*, donde además se modifican pesos. Normalmente se aplica un solo vector de entrada cada vez. En nuestro caso, se suministran dos. En el modo normal, se provee a la red a la vez de dos vectores de entrada $\vec{x}$ e $\vec{y}$, obteniéndose una salida compuesta por otros dos $\vec{x'}$ e $\vec{y'}$, aproximaciones de $\vec{x}$ e $\vec{y}$, respectivamente (todos ellos normalizados). En el modo de entrenamiento, se suministra a la red los vectores $\vec{x}$ e $\vec{y}$ tanto a la entrada como a la salida. Así, $\vec{x}$ se utiliza para entrenar la salida que luego será $\vec{x'}$, e $\vec{y}$ para la que será $\vec{y'}$. El entrenamiento se lleva a cabo según se explicó más arriba para cada capa. Nótese que, para la red, lo que para nosotros son dos vectores separados, es un sólo vector de longitud igual a la suma de sus longitudes, no afectando para nada al algoritmo.

El resultado del entrenamiento es una asociación en la que la presentación de dos vectores $\vec{x}$ e $\vec{y}$ produce sus réplicas en la salida. Esto, por supuesto, no tiene gran interés, hasta que uno no se da cuenta de que, si se suministra solamente el vector $\vec{x}$ a la entrada (con el vector $\vec{y}$ a 0), se obtienen entonces tanto $\vec{x'}$ como $\vec{y'}$, es decir, se consigue una *asociación* entre $\vec{x}$ e $\vec{y'}$. Además, si existe la inversa de esta asociación, se puede obtener, simplemente suministrando el vector $\vec{y}$ (con $\vec{x}$ a 0) a la entrada, obteniendo así $\vec{x'}$. De aquí el nombre *de contrapropagación*.

5.9.3 Estudio cualitativo

El algoritmo se ha aplicado con bastante éxito a diversos problemas, entre los que destacan el reconocimiento de caracteres y la compresión de datos, realizados muchos por el propio Hecht-Nielsen. Comparado a *backpropagation*, es un algoritmo que llega a reducir el tiempo de entrenamiento en un factor de 100, con lo cual va bien para problemas en que no se puede pagar el precio de largas sesiones de entrenamiento. Como desventaja, no es tan general como *backpropagation*, ya que su pobre estructura interna no le permite realizar ciertas asociaciones. Existen extensiones al esquema básico, que consisten en permitir que se active un grupo de neuronas en la capa de Kohonen por cada vector de entrada (y no sólo una), con lo que se pueden representar asociaciones más complejas. No obstante, el número óptimo de neuronas de estos grupos no se ha podido establecer con precisión.

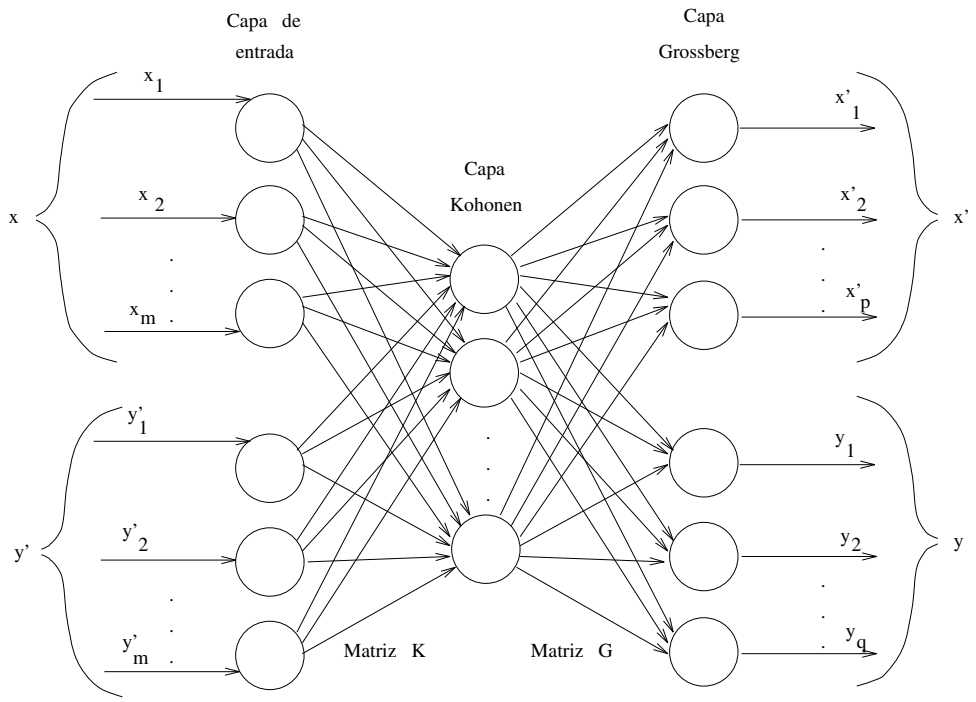


Figura 5.20: Arquitectura completa de la red de contrapropagación.

5.10 Métodos estadísticos

Los métodos estadísticos se utilizan tanto en el modo normal como en el de entrenamiento de redes neuronales, siendo su misión prinicipal evitar un problema típico de los métodos deterministas: los mínimos locales. Empezaremos viendo cómo se pueden incorporar al entrenamiento (cálculo de los pesos) y, en la sección 5.11, estudiaremos su uso en el modo normal (cálculo de la salida).

Los métodos vistos hasta ahora (*perceptron learning*, *backpropagation*, ...) son *deterministas*, en el sentido que siguen un algoritmo dado paso a paso. Con los mismos datos, y en el mismo orden, dos redes acabarán teniendo exactamente los mismos pesos. Un acercamiento diferente es el entrenamiento estadístico, en que se efectúan cambios aleatorios en los pesos, reteniendo aquellos que resultan en una mejora, usualmente medida por el error total (véase 5.6). Así, en el fondo, entrenar una red neuronal es resolver un problema de optimización no lineal consistente en una búsqueda en un espacio n-dimensional (el espacio de pesos) que minimice una función *objetivo* (el error total). Un espacio como éste está plagado de mínimos locales (valles n-dimensionales). Para ilustrar esto, y la manera en que los métodos estadísticos lo solucionan, considérese una situación como la de la figura 5.21.

En ella se muestra una función (la función objetivo) de una sola variable (por simplicidad, ya que correspondería a un único peso en una red). Supóngase ahora que el peso se inicializa al punto A. Si los cambios en el peso son pequeños, nunca se saldrá del "valle" de A, pues cualquier

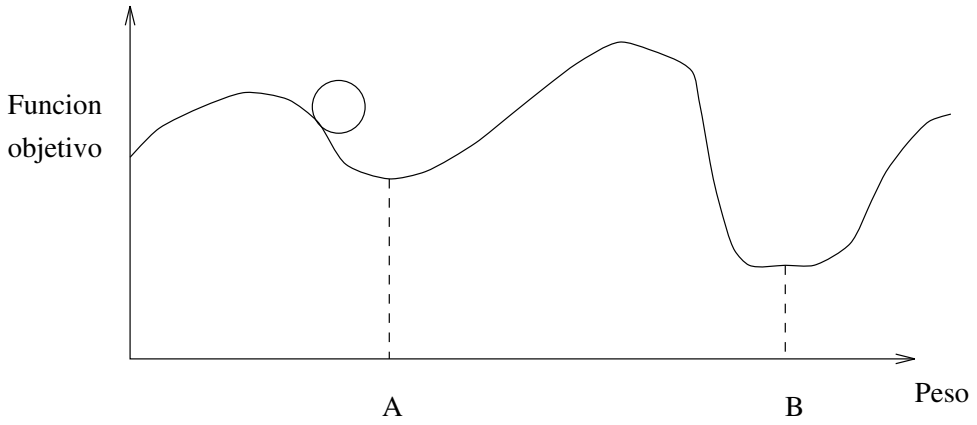


Figura 5.21: El problema de los mínimos locales.

cambio será para peor (es decir, incrementará el valor de la función). Por el contrario, si son en exceso grandes, tanto los valles de A como de B serán visitados una y otra vez (y, de hecho, también cualquier otro punto). Por tanto, el peso nunca alcanzará de una manera estable el punto B. La solución pasa por empezar por valores grandes en los cambios, e ir reduciendo éstos de manera gradual. De esta manera se asegura la estabilización en un mínimo global. Esto se puede ver volviendo a la figura. Si colocamos una pelota de manera que recorra la función por la parte superior, y la desplazamos lateralmente, ocurrirá lo siguiente: si empezamos con movimientos violentos, la bola se desplazará de un lado a otro sin quedar fija en ningún sitio. A medida que bajemos la fuerza, le costará más salir de los valles, llegando un momento en que tendrá la suficiente para salir de A pero no de B, con lo que se quedará allí (independientemente de adónde haya llegado primero). Si seguimos disminuyendo la fuerza progresivamente, la bola acabará prácticamente inmóvil en el fondo del valle de B, que es el mínimo global de la función.

Una red neuronal puede ser entrenada de la misma manera que la bola, actuando de manera aleatoria sobre los pesos. Primero se realizan cambios grandes, quedándonos con los que reducen la función objetivo, reduciendo paulatinamente los cambios hasta llegar a la estabilización. Esta manera de proceder recuerda al proceso de enfriamiento de metales (en inglés, *simulated annealing)* por lo que es también conocida por ese nombre[18]. La distribución de estados de energía viene determinada por la fórmula:

$$P(s) \propto e^{-s/kT}$$

donde $P(s)$ representa la probabilidad de que el sistema se encuentre en un estado con energía s, k es la constante de Boltzmann y T es la temperatura (en grados Kelvin). A altas temperaturas, $P(s)$ es cercana a uno para todos los estados, con lo que un estado de alta energía es igualmente probable que uno de baja. Al disminuir la temperatura, la probabilidad de los estados de alta energía se reduce comparada con la de los de baja. Al llegar ésta a cero, es altamente improbable que el sistema permanezca en un estado de alta energía.

[18] En un metal sometido a temperaturas superiores a su punto de fusión, los átomos se mueven violentamente de manera aleatoria, tendiendo (como en todo sistema físico) a un estado de mínima energía, impidiéndolo las velocidades de los átomos a altas temperaturas. Al ir enfriándose de manera controlada, se llega finalmente a un mínimo global.

Entrenamiento de Boltzmann

Es sencillo aplicar esta técnica al entrenamiento de redes neuronales, de la siguiente manera:

1. Definir una variable T que represente una temperatura "artificial". Empezar con un valor alto.
2. Suministrar a la red un conjunto de entradas y calcular las salidas y la función objetivo (el error total).
3. Efectuar un cambio (llamémosle c) aleatorio en un peso, recalculando las salidas y el error.
4. Si el error disminuye, mantener el cambio.
5. Si el error aumenta, calcular la probabilidad de aceptar ese cambio según la distribución de Boltzmann como sigue:

$$P(c) = e^{-c/kT}$$

donde $P(c)$ representa la probabilidad de un cambio de valor c en la función objetivo y k es una constante análoga a la de Boltzmann pero elegida *ad hoc* para el problema en cuestión. Si $P(c)$ es mayor que un número elegido al azar entre 0 y 1 (de una distribución uniforme) aceptar el cambio; sino, rechazarlo.

El último punto permite a la función salir de los mínimos locales, esto es, "escalar" los valles en busca de otros mínimos, aceptando de manera momentánea cambios en un punto en donde cualquier pequeño cambio empeora la función objetivo. El algoritmo completo consiste en aplicar los pasos 3, 4 y 5 para todos los pesos de la red, reduciendo gradualmente la temperatura hasta llegar a un valor aceptablemente bajo de la función objetivo. En este momento se suministra otro vector de entrada y se comienza de nuevo (quizá repitiendolos en algún momento) hasta que el comportamiento global sea satisfactorio.

Quedan por determinar tres subalgoritmos: la elección del tamaño del cambio en los pesos, el orden en que se actualizan y la manera de ir reduciendo la temperatura.

Tamaño del cambio. Se suele determinar de acuerdo a una distribución gaussiana, siendo la probabilidad de un cambio de tamaño w igual a:

$$P(w) = e^{-w^2/T^2}.$$

Selección de los pesos. Ya que nos interesa solamente el valor del cambio en sí Δw, y no su probabilidad $P(w)$, se puede utilizar un algoritmo de Montecarlo, de la siguiente manera:

1. Calcular la función de probabilidad acumulada de $P(w)$, equivalente a la integral de $P(w)$ de 0 a w. Debe calcularse numéricamente y tabularse por incrementos.
2. Elegir un número aleatoriamente (distribución uniforme) de entre el intervalo de valores de la función anterior y utilizarlo como si fuera una probabilidad $P(w)$, encontrando el correspondiente valor de Δw.

Reducción de la temperatura. La razón de decrecimiento de T (dependiente del tiempo) debe ser proporcional [GEMA84] al recíproco de su logaritmo, es decir,

$$T(t) = \frac{T(0)}{\log(t+1)}.$$

Siguiendo estos pasos, la convergencia está garantizada. El problema es que, tanto teórica como experimentalmente, se ha podido verificar que adolece de tiempos de entrenamiento excesivamente prolongados.

Entrenamiento de Cauchy

Los largos tiempos de entrenamiento del método anterior llevaron a variar la elección del tamaño de los incrementos en los pesos. Una variante consiste precisamente en reemplazar la distribución de Boltzmann por la de Cauchy[19], que se caracteriza por tener los extremos más largos y de caída más suave, incrementándose por lo tanto las probabilidades de efectuar saltos más grandes. Así, la razón de reducción de la temperatura pasa de ser inversamente logarítmica a inversamente lineal, reduciendo drásticamente el tiempo total de entrenamiento, donde

$$T(t) = \frac{T(0)}{t+1}$$

siendo la distribución de Cauchy:

$$P(w) = \frac{T(t)}{T(t^2) + w^2}$$

donde $P(w)$ es la probabilidad de un cambio de tamaño x que, en este caso, puede ser integrada analíticamente, dando lugar a la expresión:

$$\Delta w = \sigma\{T(t) \tan P(w)\}$$

donde σ es la razón de aprendizaje. Ahora el método de Montecarlo se aplica más fácilmente incluso, pues basta con elegir un número al azar de entre el intervalo abierto $(-\pi/2, \pi/2)$ (debido a la función tangente) y substituirlo por $P(w)$ en la anterior fórmula.

5.11 Redes recurrentes

Todos los modelos de red presentados hasta el momento comparten una característica: no hay conexiones *hacia atrás*, es decir, de manera que se puedan formar ciclos. Esto asegura la estabilidad de las redes, esto es, las salidas dan un valor concreto y permanente, hasta que se cambia la entrada. Esta es, en efecto, una gran ventaja, que comporta, no obstante, un alto precio: las redes *no recurrentes* (pues así se llaman) tienen un comportamiento limitado comparado con el de las llamadas *redes recurrentes*.

[19] A los sistemas que utilizan estos métodos se les suele conocer por el nombre de *máquinas* de Boltzmann y Cauchy, respectivamente.

Si pensamos en estas últimas, caeremos en la cuenta de que, debido a su continua realimentación, tienen un comportamiento dinámico y, posiblemente, inestable: se comienza aplicando una entrada, se calcula la correspondiente salida y, a diferencia de las redes vistas hasta el momento, la salida pasa ahora a alimentar la entrada, recalculándose de nuevo aquella, y así sucesivamente. El proceso puede converger (las salidas se parecen cada vez más hasta llegar a un valor estable) o no (lo que nos lleva a los sistemas caóticos, que caen fuera de este texto). John Hopfield [HOPF82] ha trabajado extensamente en este campo, razón por lo que muchas configuraciones son conocidas por su nombre. También Grossberg (por ejemplo, [GROS87]) le ha dado mucho.

5.11.1 Redes de Hopfield

Centrémonos en las redes estables. Observando la figura 5.22 vemos la configuración más simple de red recurrente. De nuevo se muestra la capa de entrada para facilitar el entendimiento, actuando sólo como distribuidora de las salidas hacia las entradas (obsérvese entonces que la conectividad es *total*). La entrada real está representada por las conexiones x, con peso asociado igual a uno. El cálculo de la salida de estas neuronas es el clásico, con función de activación lindero, en principio con linderos diferentes para neuronas diferentes, y donde se incluye el tiempo para facilitar el entendimiento. Así, y siguiendo la notación de la figura 5.22:

$$y'_j(t+1) = x_j + \sum_{i \neq j}^{n} w_{ij}\, y_i(t) \tag{5.13}$$

$$y_j(t+1) = \begin{cases} 1 & \text{si } y'_j(t+1) > L_j \\ 0 & \text{si } y'_j(t+1) < L_j \\ y_j(t) & \text{si } y'_j(t+1) = L_j \end{cases} \tag{5.14}$$

El entrenamiento es muy simple: hebbiano, según la fórmula:

$$\Delta w_{ij} = \frac{1}{n} p_i p_j \propto p_i p_j \,,$$

siendo $p_i p_j$ dos componentes de un mismo patrón de entrada.

El funcionamiento de la red se observa mejor geométricamente. Las figuras 5.23 y 5.24 muestran el caso de capas de 2 y 3 neuronas, en que el sistema puede estar en 4 y 8 estados posibles, respectivamente (entendiendo por estado el vector de salida una vez estabilizado). En general, para n neuronas, se obtiene un cubo n-dimensional. Cuando se suministra a la red un nuevo vector de entrada, ésta se mueve de un vértice del cubo a otro, estabilizándose en uno de ellos. Si el vector está incompleto o es incorrecto, se acaba en el más cercano al original.

Existen criterios de estabilidad en función de los pesos. Si denominamos w a la matriz, entonces [COHE83] la red recurrente será estable si w es simétrica con ceros en la diagonal principal, esto es, si:

1. $\forall i \;\; w_{ii} = 0$
2. $\forall i, j \;\; w_{ij} = w_{ji}$

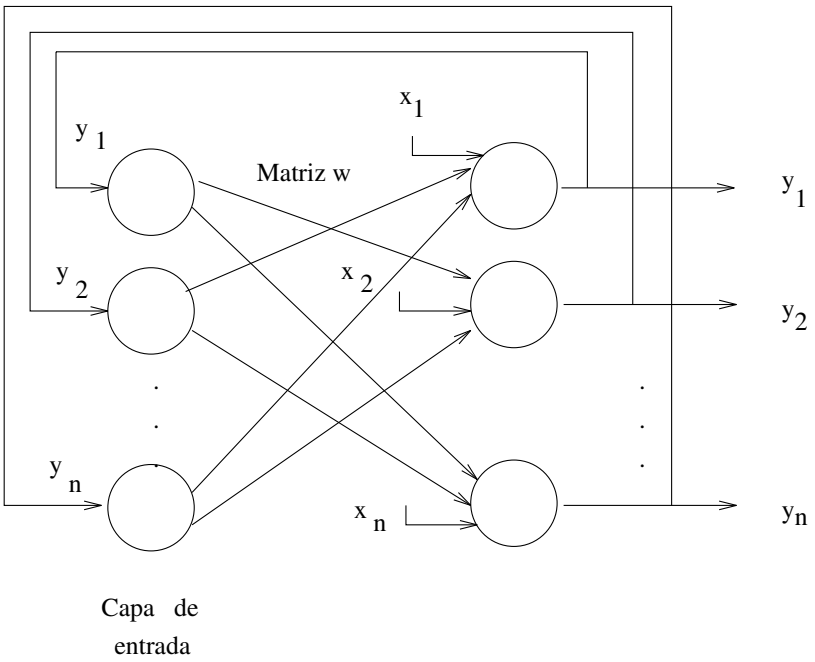


Figura 5.22: Red recurrente de una sola capa.

Es interesante mostrar la prueba de estabilidad, pues ayuda a comprender el funcionamiento de estas redes. Supóngase, pues, que podemos encontrar una función, dependiente de la entrada actual, los pesos y las salidas de la red en un momento dado, tal que vaya decreciendo a medida que la red evoluciona después de la presentación de una entrada. Esta función alcanzará un mínimo en algún instante, indicando que la red se ha estabilizado[20]. Demostraremos que la siguiente función (llamada de Liapunov) cumple tal requisito:

$$F(w, \vec{x}, \vec{y}) = -\frac{1}{2} \sum_i \sum_j w_{ij}\, y_i y_j - \sum_j x_j\, y_j + \sum_j L_j\, y_j \tag{5.15}$$

Dado un cambio Δy_j en la salida de la neurona j, el cambio en F será:

$$\Delta F = -\left[\sum_{i \neq j} w_{ij}\, y_i + x_j - L_j \right] \Delta y_j = -(y'_j - L_j)\Delta y_j$$

Aquí se presentan tres casos, que son justamente las comparaciones con el lindero L_j de la neurona j en (5.13):

1. Si $(y'_j - L_j) > 0$, este término será positivo y, de (5.13) y (5.14), concluimos que y_j debe

[20] No debe confundirse esta expresión con la medida del error total vista con anterioridad: el descenso del gradiente en el error se realiza modificando los *pesos*, mientras que ahora se trata de modificar los *estados* de las neuronas.

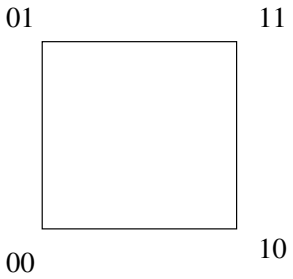


Figura 5.23: Estados correspondientes a una red de 2 neuronas.

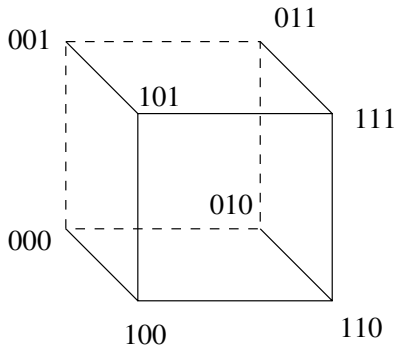


Figura 5.24: Estados correspondientes a una red de 3 neuronas.

ser ahora igual a uno, con lo que Δy_j sólo puede ser uno (si antes era cero) o cero (si antes era uno). En cualquier caso, Δy_j es positivo o cero, con lo que ΔF es negativo o cero (la función no aumenta).

2. Simétricamente, si $(y'_j - L_j) < 0$, ahora Δy_j es negativo o cero, con lo que ΔF también debe serlo.

3. Por último, en caso que $(y'_j - L_j) = 0$, $\Delta y_j = 0$ y, consecuentemente, la función no varía.

Sin embargo, el criterio de simetría de la matriz de pesos es condición suficiente pero no necesaria. Existen sistemas (todos los no recurrentes y algunos recurrentes) que son estables sin que se dé esta condición.

Las redes de Hopfield y, en general, todas las recurrentes, presentan buenas capacidades de asociación. Si los nodos de entrada son idénticos a los de salida, entonces la red puede utilizarse también como *autoasociador* (asociación de un vector consigo mismo), posibilidad interesante para reconstruir entradas incompletas o parcialmente erróneas.

Cuando se quiere recuperar un vector ya incorporado a la red, se le muestra (con ceros en los componentes desconocidos) y, tras varias iteraciones, la red lo reconstruirá. Cabe decir que el orden de actualización de las neuronas es irrelevante siempre y cuando a todas les acabe llegando el turno en un momento u otro y tantas veces como sea necesario.

El mismo Hopfield (en [HOPF84]) desarrolló una variante claramente orientada a la autoasociación, en la que los componentes de los vectores de entrada están formadas por elementos del conjunto $\{-1, +1\}$ y la información de los vectores a recordar se almacena en los pesos según la fórmula vectorial:

$$W = \sum_i p_i^t \times p_i \tag{5.16}$$

donde p_i es el i-ésimo patrón de entrada[21], W la matriz de pesos y denotamos por el símbolo $\times$ el producto externo[22]. En estas condiciones, para recuperar un patrón se presenta éste (es decir, su versión incompleta) a la salida (esto es, la primera salida es forzada a ser la propia entrada incompleta. lo que inicializa el vector $\vec{y}$) y se deja evolucionar a la red hasta que eventualmente se estabilice.

5.11.2 Extensiones al modelo básico

Hay dos maneras naturales de generalizar el modelo: hacerlo continuo e incorporar los métodos estadísticos vistos. Las dos tienen su razón de ser: por un lado, si los vectores de entrada no son ortogonales, los asociadores lineales no pueden garantizar un perfecto recuerdo; en este caso, es mejor recurrir a los no lineales. Por otro, las grandes ventajas de los métodos estadísticos en el tratamiento de mínimos ya han sido comentadas. Además, una red de Hopfield de N unidades totalmente interconectadas puede almacenar del orden de $0.15N$ vectores diferentes. Sin embargo, esta capacidad puede incrementarse considerablemente empleando un método de entrenamiento más orientado a minimización del error como, por ejemplo, el mismo criterio del perceptrón.

Modelo de función de activación continua

Una ventaja de las redes neuronales es la gran facilidad con que podemos experimentar con ellas. En este caso, es posible substituir la función de activación lindero por la sigmoide. Si se juega con la pendiente (como vimos en la figura 5.1) obtendremos funciones tan alejadas de la versión discreta (la lindero) como queramos. Lo único a tener en cuenta será que la red no acabará en uno de los vértices del hipercubo, sino cerca de él. A medida que disminuyamos la pendiente (esto es, la suavicemos), los puntos estables se alejarán de los vértices, y desaparecerán al acercarnos a pendiente cero.

Modelo estadístico

Si modificamos la relación (5.14) de manera que incorpore la distribución de probabilidad de Boltzmann, volvemos a encontrarnos con el método de *simulated annealing*. De hecho, la máquina de Boltzmann es una generalización de las redes de Hopfield con el añadido de capas ocultas y donde las neuronas cambian su estado de acuerdo con una regla estocástica. En el caso que nos ocupa, basta con hacer que la probabilidad de un cambio de estado venga

[21] Obsérvese que no es más que un tipo de aprendizaje hebbiano.

[22] El producto externo de un vector $\vec{x}$ de dimensión n y otro $\vec{y}$ de dimensión m es una matriz M de $m \times n$ donde $M_{ij} = x_i y_j$.

dada, no solamente por el signo de $(y'_j - L_j)$, sino por su valor. Así, siendo $C_j = y'_j - L_j$, tal probabilidad seguirá la fórmula:

$$p_j(t+1) = \frac{1}{1 + e^{-\Delta C_j / T}}$$

siendo $\Delta C_j = y'_j(t+1) - y'_j(t)$ el incremento del valor de una neurona. Así, para cada una, la función de activación consiste en darle el valor uno si su probabilidad es suficientemente alta y cero en caso contrario. De manera similar a los métodos precedentes, se reduce gradualmente la temperatura hasta alcanzar el equilibrio.

5.11.3 Ejemplo: el problema del viajante de comercio

Este problema es un clásico de la Algorítmica, y como tal se suele usar como referencia al desarrollar nuevos métodos de resolución de problemas. En realidad es una tarea de optimización, aplicable a muchos otros casos. El enunciado –bajo la forma del viajante– es el siguiente: dado un conjunto de ciudades y sus distancias respectivas, encontrar el recorrido más corto tal que las visite *todas* y *una sola vez* cada una. La dificultad, evidentemente, reside en que se pide el más corto (sino, cualquier enumeración de los elementos del conjunto sería válida). Este problema pertenece a la clase de los NP-completos, esto es, no puede ser resuelto en tiempo polinómico por una máquina determinista pero sí por una no determinista. La completitud significa que cualquier otro problema de la clase NP puede ser reducido a éste (es decir, si llegamos a conocer una solución para un NP-completo, habremos resuelto también los demás NP). Esta clasificación, en términos prácticos, significa además que la única manera conocida de encontrar una solución *óptima* (donde por óptima entendemos que no hay ninguna mejor) es generando todas las posibilidades y calculando la longitud del recorrido. En nuestro caso, para n ciudades, existen $(n-1)!/2$ posibles recorridos lo que, para hacerse una idea, ¡ equivale a generar y calcular del orden de 0.7×10^{80} recorridos solamente para 60 ciudades! La solución pasa por establecer heurísticas que, a falta de otra cosa, encuentren soluciones razonablemente buenas. La siguiente fue desarrollada por Hopfield y Tank (en [HOPF85]), y tiene la ventaja de su tremenda velocidad comparada con algoritmos clásicos de caminos mínimos.

Sea n entonces el número de ciudades, y denotemos por d_{ij} la distancia entre la i y la j. La solución será una lista de n ciudades. Utilizaremos una red recurrente con función de activación sigmoidal de alta ganancia (es decir, pendiente grande) donde cada ciudad vendrá representada por n neuronas (recuérdese que la conectividad es total). Las n neuronas dedicadas a una ciudad indicarán el orden en que ésta debe ser visitada, pudiendo estar solamente una de ellas con valor 1 y las $n-1$ restantes a 0. La siguiente tabla muestra un ejemplo de tal disposición para $n = 4$.

Si queremos utilizar las ideas vistas en esta sección, debemos empezar por encontrar una función objetivo, a minimizar. Dicha función debe cumplir dos requisitos fundamentales:

1. Debe dar valores bajos para aquellas configuraciones de la red que tengan un sólo 1 en cada fila y columna.
2. Idem para configuraciones que representen caminos mínimos.

Separaremos –para mejor entendimiento– la función en dos subexpresiones, correspondientes

Ciudad	Orden de visita			
	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	0	0	1	0
4	0	1	0	0

Figura 5.25: Interpretación: la ciudad 2 se visita primero, luego la 4, la 3 y la 1.

a los dos puntos mencionados. Siendo E la expresión total, entonces

$$E = E_1 + E_2$$

donde, denotando por y_{cp} el valor de la neurona que se ocupa de la posición p de la ciudad c, definimos:

$$\begin{aligned} E_1 &= \frac{A}{2}\sum_{c}^{n}\sum_{p}^{n}\sum_{q\neq p}^{n} y_{cp}y_{cq} \\ &+ \frac{B}{2}\sum_{p}^{n}\sum_{c}^{n}\sum_{d\neq c}^{n} y_{cp}y_{dp} \\ &+ \frac{C}{2}\left[(\sum_{c}^{n}\sum_{p}^{n} y_{cp}) - n^2\right]^2 \end{aligned}$$

siendo A, B, y C constantes elegidas con valores altos, ayudando los factores $1/2$ a simplificar cálculos posteriores. Como puede verse, (5.17) es cero si, y sólo si, cada ciudad tiene, como mucho, *un* 1. Similarmente, (5.17) es cero si, y sólo si, cada posición tiene, como mucho, *un* 1. Asimismo, (5.17) es cero si, y sólo si, hay exactamente n unos en la matriz.

La expresión correspondiente al segundo término es la siguiente:

$$\frac{D}{2}\sum_{c}^{n}\sum_{d\neq c}^{n}\sum_{p}^{n} d_{cd}y_{cp}(y_{d,p-1} + y_{d,p+1}) \tag{5.17}$$

donde D es también una constante alta. El sumatorio (5.17) equivale a la longitud del camino representado por la red en cada momento. Nótese que –por simplicidad– se han tomado los subíndices de la posición módulo n. El siguiente paso es establecer los pesos. Para ello debemos relacionar las expresiones anteriores con la forma general que debe tener la función objetivo, según la fórmula (5.15). Esto nos lleva a la relación:

$$\begin{aligned} w_{cp,dq} = \; & -A\delta_{cd}(1-\delta_{pq}) && \text{(evita más de un 1 en una ciudad)} \\ & -B\delta_{pq}(1-\delta_{cd}) && \text{(evita más de un 1 en una posición)} \\ & -C && \text{(número de unos igual a } n\text{)} \\ & -Dd_{cd}(\delta_{q,p-1}+\delta_{q,p+1}) && \text{(distancia)} \end{aligned}$$

donde

$$\delta_{xy} = \begin{cases} 1 & \text{si } x = y \\ 0 & \text{en otro caso} \end{cases}$$

Varios resultados utilizando esta red se pueden encontrar en [HOPF85]. Allí se añadió una entrada extra a cada neurona con peso siempre a 1 y valor igual a Cn (cuyo efecto es desplazar la salida y_{cp} de cada neurona en este valor) y se empleó la tangente hiperbólica como función de activación, dando lugar a las fórmulas (compárense con (5.13) y (5.14)):

$$y'_j(t+1) = Cn + \sum_{i \neq j}^{n} w_{ij}\, y_i(t) \tag{5.18}$$

$$y_j(t+1) = \frac{1}{2}(1 + \tanh \frac{y'_j(t+1)}{\Theta}) \tag{5.19}$$

pudiendo controlar su pendiente mediante Θ. La red fue probada para $n = 10$ y, en 16 de 20 pruebas, se llegó a recorridos válidos, donde éstos eran además mínimos para aproximadamente el 50% de los casos. Téngase en cuenta que existen, en este caso, un total de 181.440 recorridos válidos. Como desventaja, nótese que el número de neuronas necesarias es $n^2 = 100$ lo que, para n superiores, podría llegar a ser excesivo. Existen otros trabajos (por ejemplo, [BOUT88]) que proponen funciones objetivo alternativas, más simples y con nuevos algoritmos de convergencia, con lo que sigue siendo un campo muy abierto.

5.12 Memorias asociativas bidireccionales

La capacidad de memorización vista en la sección precedente es básicamente *autoasociativa*, es decir, un patrón puede ser completado o corregido, pero no asociado con otro. La razón es que las redes están estructuradas en una sola capa y, por consiguiente, la salida debe aparecer en las mismas neuronas que la entrada. Una posibilidad es añadir capas, dedicando unas neuronas a la entrada y otras (diferentes) a la salida. Esto da lugar a las memorias asociativas bidireccionales (MAB), las cuales son, por contra, *heteroasociativas*, manteniendo las capacidades de abstracción y generalización. En la figura 5.26 se muestra la MAB más sencilla, compuesta por dos capas. El vector de entrada actúa a la salida de la primera capa. Siguiendo su notación, el funcionamiento es el siguiente:

1. Suministrar un vector por la entrada $\vec{x}$, que puede ser incompleto o incorrecto.
2. Calcular $\vec{y} = f(W\vec{x})$.
3. Calcular $\vec{x} = f(W^t\vec{y})$.

Los pasos 2. y 3. se repiten hasta que se llegue a un punto estable, esto es, hasta que ni $\vec{y}$ ni $\vec{x}$ cambien. La asociación que tiene memorizada la red es entre el primer valor de $\vec{x}$ y el último de $\vec{y}$. La función de activación f puede ser cualquiera de las vistas hasta ahora: una buena elección es siempre la sigmoide. En cuanto a la función objetivo, Kosko [KOSK87a]

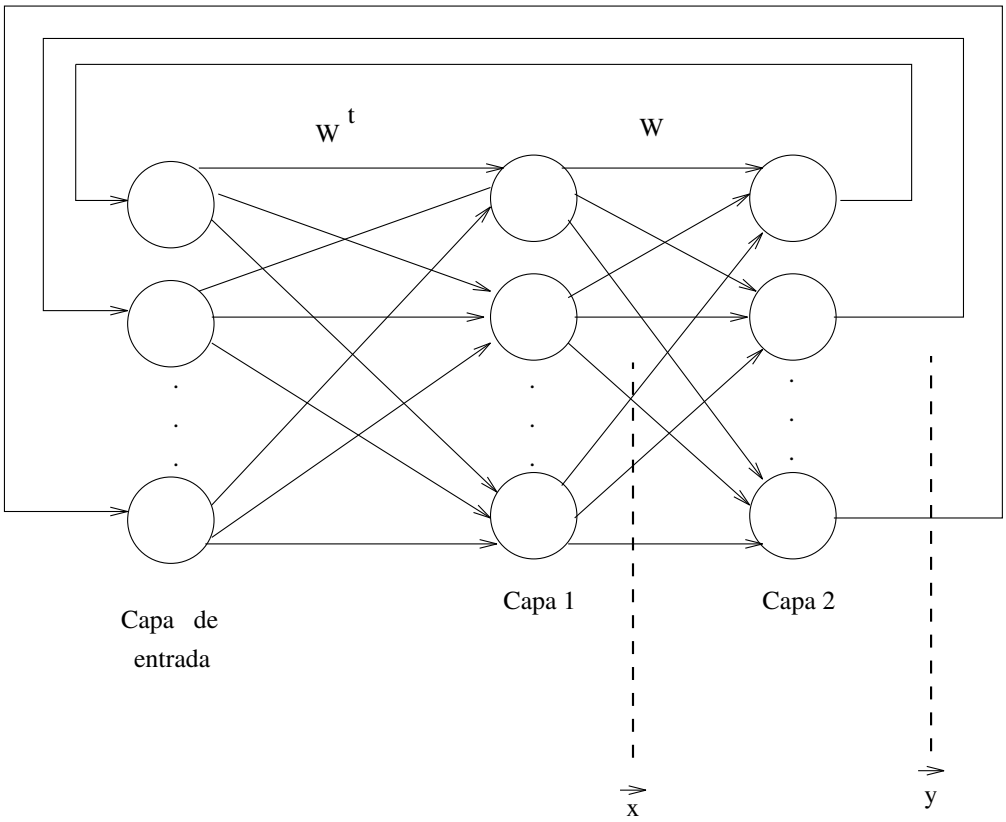


Figura 5.26: Arquitectura de una BAM.

estableció una para la que se puede demostrar –de manera similar a como se hizo para las redes de Hopfield– que tiende a un mínimo global:

$$F(w, \vec{x}, \vec{y}) = -\sum_i \sum_j w_{ij} x_i y_j \tag{5.20}$$

La matriz de pesos W se calcula de manera similar a la fórmula de autoasociación de Hopfield (ecuación 5.16), sólo que ahora –al tratarse de heteroasociación– se deben suministrar pares completos de entrenamiento $< p_i, q_i >$:

$$W = \sum_i p_i^t \times q_i$$

donde denotamos por el símbolo $\times$ el producto externo de vectores, que están compuestos por -1 y $+1$ en vez de 0 y 1[23].

Veamos un ejemplo, donde, para simplificar, tomamos la función lindero (con lindero igual a cero para todas las neuronas) en substitución de la sigmoide. Supongamos que queremos asociar los siguientes pares:

[23] Kosko [KOSK87b] encontró que así se producían mejores resultados.

$$p_1 = \begin{pmatrix} -1 \\ +1 \\ +1 \end{pmatrix}; p_2 = \begin{pmatrix} +1 \\ +1 \\ +1 \end{pmatrix}; p_3 = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix}$$

$$q_1 = \begin{pmatrix} +1 \\ +1 \\ -1 \end{pmatrix}; q_2 = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}; q_3 = \begin{pmatrix} +1 \\ -1 \\ -1 \end{pmatrix}$$

La matriz de pesos será:

$$W = p_1^t \times q_1 + p_2^t \times q_2 + p_3^t \times q_3$$

$$= \begin{pmatrix} -1 & -1 & +1 \\ +1 & +1 & -1 \\ +1 & +1 & -1 \end{pmatrix} + \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix} + \begin{pmatrix} -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} -3 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -3 \end{pmatrix}$$

Obsérvese que la matriz es simétrica. Si ahora aplicamos p_1 a la red, obtendremos:

$$Wp_1 = \begin{pmatrix} -3 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -3 \end{pmatrix} \begin{pmatrix} -1 \\ +1 \\ +1 \end{pmatrix} = \begin{pmatrix} +4 \\ +1 \\ -5 \end{pmatrix}$$

vector que, después de pasar por la función lindero, da de nuevo el vector

$$q_1 = \begin{pmatrix} +1 \\ +1 \\ -1 \end{pmatrix}$$

En este caso, es interesante ver qué pasa si continuamos con el cálculo (pues nos ha salido a la primera debido a la sencillez del ejemplo), hallando la expresión $W^t q_1$:

$$W^t q_1 = \begin{pmatrix} -3 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -3 \end{pmatrix} \begin{pmatrix} +1 \\ +1 \\ -1 \end{pmatrix} = \begin{pmatrix} -5 \\ 1 \\ 4 \end{pmatrix} \longrightarrow \begin{pmatrix} -1 \\ +1 \\ +1 \end{pmatrix} = p_1$$

Como vemos, se crea una realimentación, manteniéndose a la vez p_1 y q_1. La capacidad para generalizar de la red también es sencilla de ver. Tomemos un p_4 incorrecto, de la siguiente manera:

$$p_4 = \begin{pmatrix} +1 \\ -1 \\ +1 \end{pmatrix}$$

El resultado es $q_2 = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$, el vector solución más cercano, ya que p_4 difiere en sólo un componente de p_2 y p_3, y cualquiera entre q_2 y q_3 es válido.

Podría pensarse, a primera vista, que el hecho de que W sea simétrica es condición indispensable para la estabilidad, como pasaba en las redes de Hopfield. Esto no es así: el mismo Kosko demostró que las MAB son estables incondicionalmente, debido fundamentalmente a la relación de transposición entre las matrices de pesos utilizadas. Por supuesto, si W es cuadrada y simétrica, entonces $W = W^t$ y, por tanto, las dos capas son la misma, con lo que, en este caso, la MAB correspondería exactamente a una red de Hopfield.

5.13 Autoorganización

Se sabe que, en muchos casos, estímulos cercanos son codificados en áreas corticales cercanas. El propio Kohonen ha trabajado en lo que denominó *mapas autoorganizativos* [KOHO88], clasificando patrones de entrada según algoritmos no supervisados, capaces de construir estas organizaciones. Esta construcción es progresiva –con entradas presentadas aleatoriamente– y modificando los pesos de manera que se refuerce la proximidad entre la distribución de la entrada y la configuración (discreta) de los pesos. Así, el algoritmo crea una representación organizada del espacio de entrada, a partir de un desorden inicial. Esta adaptación de los pesos se realiza en dos pasos: primero, autoorganización (ordenación); luego, convergencia para cuantificar mejor el espacio de entrada.

La red se compone de una capa de n neuronas estructuradas en forma de retículo (codificado, generalmente, en un vector o una matriz). El conjunto de neuronas $N = \{1, 2, \ldots, n\}$ tiene una topología estructural que viene dada por una *función de vecindad* V definida sobre $N \times N$. Tal función es definida simétrica e inversamente proporcional a la distancia, esto es:

1. $V(i, j) = V(j, i)$
2. $V(i, j)$ decrece si aumenta la distancia entre i y j
3. $V(i, j) = |i - j|$ si se codifica en un vector. En una matriz se deben buscar definiciones en función de sus dos índices.
4. $V(i, i) = 1$.

El espacio de entrada Ω está incluído en $\Re^d$, siendo d una constante natural positiva. Las neuronas (de hecho, deberíamos llamarlas simplemente *unidades*, pues no realizan la clase de cómputo vista hasta el momento) están *totalmente* conectadas a las entradas; en otras palabras, una componente de una entrada puede *acceder* a la red por cualquier unidad. Para mantener la notación (aunque, como veremos a continuación, no se corresponde exactamente), seguiremos denominando w_{ij} al peso (o fuerza) de conexión entre la unidad i y la componente j de una entrada.

La idea consiste en representar una unidad i por el siguiente vector:

$$\vec{w}_i = (w_{i1}, w_{i2}, \ldots, w_{id}).$$

Si hace falta, se pueden normalizar los vectores $\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_n$ de manera que podamos representarlos en el mismo espacio Ω que las entradas. Así las cosas, el estado de la red en tiempo t viene dado por:

$$w(t) = (\vec{w}_1(t), \vec{w}_2(t), \ldots, \vec{w}_n(t))$$

Entonces, para un estado cualquiera w, la respuesta de la red a una entrada $\vec{x}$ es la unidad *ganadora* i_0, aquella más cercana a $\vec{x}$, a la que denominamos $i(\vec{x}, w)$. Así, la red define una aplicación

$$\begin{aligned} \Phi_w: \quad & \Omega \rightarrow N \\ & \vec{x} \rightarrow i(\vec{x}, w) \end{aligned}$$

siendo el objetivo del algoritmo de aprendizaje converger hacia un estado de la red tal que su aplicación correspondiente descubra la topología existente.

Dado un estado w, denotemos por $G_i(w)$ el conjunto de entradas de Ω tales que i es la unidad ganadora para ellas, esto es, $G_i(w) = \Phi_w^{-1}(i)$. El conjunto de clases $G_i(w)$ es la teselación de Voronoï euclídea del espacio Ω con relación a w. En estas condiciones, el algoritmo es el siguiente:

- Elegir valores iniciales para $w(0) = (\vec{w}_1(0), \vec{w}_2(0), \ldots, \vec{w}_n(0))$ de manera aleatoria.
- Siendo $w(t)$ el estado actual,
  - Presentar la entrada $\vec{x}(t+1)$ correspondiente, elegida de Ω según la distribución de probabilidad P que se desee.
  - Calcular la unidad ganadora i_0 según la fórmula:

$$dist(\vec{w}_{i_0}(t), \vec{x}(t+1)) = \min_j \; dist(\vec{w}_j(t), \vec{x}(t+1))$$

  - Actualizar los pesos según:

$$\vec{w}_i(t+1) = \vec{w}_i(t) - \alpha_t V(i_0, i)(\vec{w}_i(t) - \vec{x}(t+1))$$

    para cada $i \in N$

Este proceso refuerza la similitud entre la entrada $\vec{x}(t+1)$ y las respuestas de la unidad i_0 y de sus vecinas, disminuyendo paulatinamente con la distancia a i_0. Los parámetros a determinar son: $\alpha_t < 1$, razón de aprendizaje, pequeña y positiva (que puede decrecer con el tiempo, motivo de su subíndice), la función de vecindad V (que también puede depender del tiempo), la dimensión d del espacio de entrada y la distribución de probabilidad P. La función de distancia es la euclídea:

$$dist(\vec{x}, \vec{y}) = \sqrt{\sum_i^n (x_i - y_i)^2}$$

. Se puede incluso establecer un intervalo (o un entorno, si trabajamos con una matriz de vecindad) de actualización de pesos, modificando sólo aquéllos que pertenezcan a él.

Al tratarse de un algoritmo no supervisado, no puede saberse con anterioridad al entrenamiento qué neuronas se asociarán con cada clase. Eso sí, si se asume que los vectores de entrada se pueden agrupar en clases, una clase específica tenderá a controlar una neurona concreta, rotando su vector de pesos hacia el centro de la clase, forzando que sea esa y no otra la ganadora cuando se presente una entrada de la clase. En el modo normal, la neurona ganadora es precisamente la indicadora de la clase correcta.

5.14 Características generales de las redes neuronales

Los rasgos más significativos de las redes –que son, básicamente, los que han provocado su intenso estudio y originado más de una controversia–, se sintetizan en su capacidad de aprender de la propia experiencia a base de ejemplos, generalizar sobre nuevos y extraer sus rasgos importantes eliminando datos irrelevantes. Podríamos englobarlos en tres grandes grupos:

Aprendizaje: mejora paulatina de la respuesta de una red, ya sea delante de entradas nuevas o presentadas con anterioridad. Las redes son capaces de mejorar su trabajo autoadaptándose, con o sin conocimiento de la respuesta o comportamiento esperado o correcto, a través de la modificación de sus pesos asociados[24].

Abstracción: extracción del ideal de entrada mediante ejemplos (posiblemente distorsionados). Prácticamente todas las redes aprenden a *abstraer*, es decir, a ignorar aspectos poco o nada importantes quedándose con los rasgos básicos de los patrones de entrada. Por ejemplo, una red entrenada para reconocer la letra 'A', lo hará a pesar de que ésta se le presente con un cierto grado de distorsión, considerando así todo un conjunto de entradas como pertenecientes a la *clase de la letra 'A'* ('A' escritas a mano por diferentes personas, diferentes tipos de letra, etc).

Generalización: capacidad de producir salidas correctas a pesar de entradas incorrectas o incompletas, y efectuar predicciones sobre nuevas. Desde el punto de vista de la eliminación del ruido y la capacidad de reconstrucción de imágenes, las redes han tenido grandes éxitos en problemas del mundo real, como reconocimiento de patrones, visión artificial, etc, bastándoles un cierto número de ejemplos para poder reconocer futuras entradas nunca vistas con anterioridad.

Desde el punto de vista del cálculo sobre los patrones de entrada, las tareas que son capaces de realizar son básicamente tres:

Completar un patrón.

Clasificar un patrón.

Computar una función de un patrón.

Para ello, en esencia, los algoritmos conexionistas encuentran un atractor en el espacio de potencial que definen. El resultado, usualmente un óptimo local, es la situación de este atractor en dicho espacio, mientras el entrenamiento de una red neuronal es lo que da forma a este espacio: cada experiencia (o ejemplo) de entrenamiento se incorpora bajo la forma de un atractor, y el algoritmo asegura que se encontrará, para cada entrada, el atractor más (o uno de los más) parecidos.

Cabe remarcar que todas estas características son debidas a la dinámica de la computación ejercida por la red, y no por ninguna intención explícitamente preprogramada. Ahora bien, no todo son ventajas. Las redes presentan cierto número de inconvenientes todavía no resueltos,

[24] Existen otros mecanismos, como supresión o incorporación de nuevos nodos y/o conexiones, que afectan a la arquitectura de la red, pero no serán tratados aquí.

el mayor de los cuales es su total libertad de traducción de los resultados por parte humana, debido a la imposibilidad de intentar encontrar las razones por las cuales una red se comporta como lo hace dependiendo del orden de presentación o de las razones de aprendizaje. En la misma línea, nos encontramos con que se hace muy difícil la interpretación de los pesos, especialmente los de las capas ocultas, si se consideran redes de más de 5 ó 6 neuronas, aunque se trate de una sola capa.

5.15 Conexionismo e Inteligencia Artificial simbólica.

Como se ha visto, el enfoque conexionista de la Inteligencia Artificial es radicalmente diferente de las aproximaciones simbólicas tradicionales. Lo que es indudable es que ambas tendencias se enfrentan a los mismos problemas generales, esto es, en las áreas de búsqueda, representación del conocimiento y aprendizaje automático. El paradigma está actualmente orientado –aunque bastante confusamente– en tres líneas, no siempre mutuamente excluyentes:

1. El que podríamos denominar *clásico*, que incluye las tareas ya mencionadas de reconocimiento de patrones, análisis de datos, etc. En general, problemas nuéricos o de muy bajo nivel.

2. El enfocado a la construcción de sistemas inteligentes exclusivamente formados por estructuras conexionistas. Es decir, se asume que tanto la representación como el razonamiento sobre ella son no simbólicos. Proclama, por lo tanto, la *substitución total* de la IA simbólica por la conexionista.

3. El que propugna el desarrollo de arquitecturas *híbridas* simbólico-conexionistas. Se distinguen dos familias claramente contrapuestas:

   - Las que descomponen el sistema en partes conexionistas y partes simbólicas, trabajando de manera cooperativa, cada una diseñada por su lado. Los problemas tradicionalmente simbólicos son realizados por la parte simbólica, y similarmente para la conexionista. Aquí se pueden utilizar parte de los sistemas desarrollados en 1.
   - Las que ven la parte simbólica (es decir, la creación de los símbolos) como el resultado emergente de la dinámica de la parte conexionista, es decir, los símbolos y su manipulación proceden –y tienen su razón de ser– de la parte conexionista.

Bajo otro punto de vista –el de la relación con las ciencias cognitivas y la psicología– podemos establecer también tres grandes líneas que, por supuesto, no se corresponden exactamente con las anteriores:

1. El conexionismo es sólo un *modelo de implementación* de otras teorías, sin relevancia psicológica propia.

2. El conexionismo muestra, a un nivel de detalle más fino, las mismas estructuras tratadas por la IA convencional. Este nivel sí es psicológicamente relevante.

Métodos de trabajo		
Área	Conexionismo	IA Simbólica
Búsqueda	Estabilización	Espacio de Estados
Representación	Conexiones y Pesos	Frames, Scripts, Redes semánticas, Lógica de primer orden
Aprendizaje	Backpropagation Aprendizaje por refuerzo Aprendizaje asociativo	Espacio de Versiones Aprendizaje por Analogía

Figura 5.27: Comparación de métodos en Conexionismo e Inteligencia Artificial simbólica.

3. El conexionismo explica un rango restringido de fenómenos mentales (pero no cognitivos). Tan sólo las técnicas convencionales pueden modelar capacidades verdaderamente cognitivas.

En la tabla de la figura 5.27 se listan algunos de los métodos empleados en cada una de las dos áreas –sin la intención de ser exhaustivos– que han sido estudiados en los capítulos anteriores. Como se dijo, todos ellos presentan ventajas e inconvenientes.

En conjunto, todas las aproximaciones donde interviene de algún modo el conexionismo tienen como motivación principal el poner remedio a la falta de robustez y flexibilidad de los sistemas simbólicos, y a sus limitaciones en sus entradas y salidas, es decir, en su interacción con el mundo real.

5.16 Ejercicios

1. Hasta ahora se han presentado dos teorías sobre la naturaleza de la inteligencia: la de los símbolos físicos y el conexionismo. Reflexionar sobre las características de ambas.

2. Comprobar que la segunda capa de la red de la figura 5.12 efectúa una conjunción lógica. Modificar los pesos (incluido el lindero) para que efectúe otras funciones.

3. Construir una red que distinga entre los puntos de dentro y fuera de un cuadrado, considerándolo primero como un polígono convexo de 4 lados, y luego como la unión de dos triángulos. Suponer dadas las coordenadas del cuadrado.

4. Implementar, en LISP, un perceptrón monocapa formado por 3 neuronas. Diseñar una función linealmente separable (se aconseja hacerlo gráficamente) y entrenar el perceptrón para que la reconozca.

5. Comprobar (geométricamente: bastan una regla y un lápiz) la convergencia del descenso del gradiente del ejemplo de las figuras 5.16 y 5.17 para la secuencia propuesta en el texto.

6. El contorno convexo (en inglés, *convex hull*) de un conjunto de vectores $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ es el conjunto formado por los vectores $\vec{x}$ tal que:

$$\vec{x} = \sum_i^n \alpha_i \vec{x}_i$$

donde los α_i son coeficientes reales no negativos, que deben sumar, en total, uno. Dados dos conjuntos de vectores, mostrar que, o bien son linealmente separables, o bien sus contornos convexos intersecan. Pista: supóngase que los dos casos son ciertos, y considérese la clasificación de un punto situado en la intersección de los contornos.

7. Implementar, en LISP, el algoritmo de *backpropagation*. Basándose en los ejercicios anteriores, entrenar una red multicapa de manera que aprenda conjuntos de pares de entrenamiento dados, fijando el error máximo que se permitirá. Consideraciones a tener en cuenta:

   (a) Se puede utilizar la siguiente fórmula para el cálculo del error de un par de entrenamiento:
   $$\text{Error} = \sum_i (R'_i - R_i)^2,$$
   donde el error total es la suma de los errores de cada par del conjunto.

   (b) Es muy posible que el conjunto de entrenamiento se deba presentar muchas veces a la red. Obsérvese la evolución del error total *epoch* tras *epoch*.

8. Utilizar, basándose en el ejemplo del viajante de comercio del texto, una red de Hopfield para implementar un conversor A/D (analógico/digital) de n bits, teniendo en cuenta lo siguiente:

   - Las neuronas representan los amplificadores operacionales del circuito. En total hay n.
   - Los pesos, las resistencias entre ellos. Recuérdese el criterio de estabilidad enunciado.
   - Utilizar la función de activación lindero, para así obtener directamente salidas binarias.

   Una posible función a minimizar es la siguiente:
   $$F = -\frac{1}{2}(I - \sum_i^n 2^i y_i)^2$$
   siendo I la entrada (analógica) e $\vec{y}$ el vector binario de salida. Esta función decrece a medida que la diferencia (en suma de cuadrados) entre la entrada y la salida se va haciendo cero, esto es, a medida que $\vec{y}$ se acerca al valor binario de I. El factor $1/2$ simplemente ayuda a simplificar la derivada. Se debe encontrar primero el valor de los pesos, siendo uno muy sencillo:
   $$w_{ij} = -2^{i+j}$$
   Se recomienda empezar eligiendo valores de n pequeños, por ejemplo $n = 4$.

9. Proponer problemas de los tres tipos especificados en el texto (completar y clasificar patrones y calcular funciones sobre patrones) y aplicar los diversos métodos explicados, eligiendo en cada caso el (o los) más adecuado(s).

Bibliografía

[AAMO94] A. Aamodt and E. Plaza. "**Case-Based Reasoning: Foundational Issues, Methodological variations, and System Approaches**". *AI Communications*, 7(1):39–59, 1994.

[AHO83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[AMAR67] S. Amari. "**A Theory of Adaptive Pattern Classification**", 1967.

[ANDE81] J. A. Anderson and M. C. Moser. "**Categorization and selective neurons**". In G. E. Hinton and J. A. Anderson, editors, *Parallel Models of Association Memory.* Erlbaum, Hillsdale, N. J, 1981.

[ANGL87] D. Angluin. "**Learning regular sets from queries and counterexamples**". *Information and Computation*, 75:87–106, 1987.

[ANGL88] D. Angluin. "**Queries and concept learning**". *Machine Learning*, 2:319–342, 1988.

[ANGL90] D. Angluin. "**Negative results for equivalence queries**". *Machine Learning*, 5:121–150, 1990.

[ANGL92a] D. Angluin. "**Computational learning theory: survey and selected bibliography**". In *Proc. 24th Annual ACM Symposium on the Theory of Computing*, pages 351–369. ACM Press, 1992.

[ANGL92b] D. Angluin, M. Frazier, and L. Pitt. "**Learning conjunctions of Horn clauses**". *Machine Learning*, 9:147–164, 1992.

[ANTH92] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

[ARBI91] M. Arbib. "**Neural computing perspective**". *Applied Artificial Intelligence*, 5:171–185, 1991.

[ARME93] E. Armengol and E. Plaza. "**Elements of Explanation-based Learning**". Technical report, Institut d' Investigació en Intel.ligència Artificial CEAB-CSIC, Blanes, Girona, Noviembre 1993.

[ARME94] E. Armengol and E. Plaza. "**A Knowledge Level Model of Case-Based Reasoning**". In *Machine Learning ECML-94.* Springer Verlag, Lecture notes in Artificial Intelligence, 1994. Próxima aparición.

[BAIM88] P.W. Baim. "**A method for attribute selection in inductive learning systems**". *IEEE Trans. on pattern analysis and machine intelligence*, 10(6):888–896, 1988.

[BARE87] R.E. Bareiss and B.W. Porter. "**PROTOS: An Exemplar-based Learning Apprentice**". In *Proc. Fourth IWML*, 1987.

[BARE89] R. Bareiss. *Exemplar-Based Knowledge Acquisition.* Academic Press, 1989.

[BARL91] R. Barletta. "**An Introduction to Case-based Reasoning**". *AI Expert*, 6(8), 1991.

[BÉJA92] J. Béjar and U. Cortés. "**LINNEO$^+$: Herramienta para la adquisición de conocimiento y generación de reglas de clasificación en dominios poco estructurados**". In *Proceedings del III Congreso Iberoamericano de Inteligencia Artificial (IBERAMIA 92). La Habana (Cuba)*, pages 471–482, Febrero 1992.

[BÉJA93] J. Béjar, U. Cortés, and M. Poch. "**LINNEO$^+$: A Classification Methodology for Ill-structured Domains**". Technical Report LSI-93-22-R, Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, 1993.

[BELA91] Ll. Belanche. "**To be or nought to be: una qüestió irrellevant?**". Master's thesis, Facultat d'Informàtica de Barcelona, 1991.

[BERT93] A. Bertoni and M.Dorigo. "**Implicit Parallelism in Genetic Algorithms**". Technical Report TR-93-001-Revised, ICSI, April 1993. Appeared in Artificial Intelligence 61,2, 307-314.

[BISW91] G. Biswas et al. "**Conceptual clustering and exploratory data analysis**". In *Proceedings of the 8th international workshop on Machine Learning*, pages 591–595, 1991.

[BONI92] P.P. Bonissone and S. Ayud. "**Similarity Measures for Case-Based Reasoning Systems**". In *Proc. IPMU. International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 483–487. Universitat de les Illes Balears, 1992.

[BOOK89] L.B. Booker, D.E. Goldberg, and J.H. Holland. "**Classifier Systems and Genetic Algorithms**". *Artificial Intelligence*, 40(2):235–282, 1989.

[BOUT88] D. E. Van den Bout and T. K. Miller. "**A Travelling Salesman Objective Function That Works**". In *IEEE International Conference on Neural Networks*, volume 2, pages 299–303, San Diego 1988, 1988. IEEE, New York.

[BUCH78] B. G. Buchanan and E. A. Feigenbaum. "**Dendral and Meta-Dendral**". *Artificial Intelligence*, 11:5–24, 1978.

[BUSH59] W. Estes R. Bush, editor. *Studies in Mathematical Learning Theory.* Stanford University Press, 1959.

[CAPL90] L.J. Caplan and C. Schooler. "**Problem Solving by Reference to Rules or Previous Episodes: The Effects of Organized Training, Analogical Models, and Subsequent Complexity of Experience**". *Memory & Cognition*, 18(2):215–227, 1990.

[CARB83a] J. Carbonell. *Machine Learning: An Artificial Intelligence Approach.*, chapter Learning by analogy: formulating and generalizing plans from past experience. Tioga, 1983.

[CARB83b] J. G. Carbonell. "**Derivational analogy and its role in problem solving**". In *Proc. AAAI*, 1983.

[CARB86] J. Carbonell. *Machine Learning: An Artificial Intelligence Approach (vol 2)*, chapter Analogy in Problem Solving. Morgan-Kaufmann, 1986.

[CARB87] J.G. Carbonell and Y. Gil. "**Learning by Experimentation**". In *Proc. IWML*, pages 22–25, University of California,Irvine, 1987.

[CEST86] I. Bratko B. Cestnik, I. Kononenko. *ASSISTANT 86: A knowledge-elicitation tool for sophisticated users.* Sigma Press, 1986.

[CHEE88] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. "**AUTOCLASS: A Bayesian classification system**". In Morgan Kaufmann, editor, *Fifth International Conference on Machine Learning. Ann Arbor, MI.*, pages 54–64, 1988.

[CLAR92] P. Clark and R. Holte. "**Lazy Partial Evaluation: An Integration of EBG and Partial Evaluation**". In *Proc. Conference on Machine Learning*, pages 82–91, 1992.

[COHE82] P.R. Cohen and E. A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume III. Adisson–Wesley Publishing Company, Inc., 1982.

[COHE83] M. A. Cohen and S. Grossberg. "**Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks**". *IEEE Transactions on Systems, Man, and Cybernetics*, 13:815–826, 1983.

[COTT87] G. W. Cottrell, P. Munro, and D. Zipser. *Learning Internal Representations from Gray-Scale Images: An example of extensional programming.* Ninth Annual Conference of the Cognitive Science Society. Erlbaum, Seattle, WA, 1987.

[CREI88] R. Creiner et alt. *Analogica.* Morgan Kaufmann, 1988.

[CUMM91] R. Cummins. "**Cross-domain inference and problem embedding**". In *Philosophy and AI: essays at the interface*, pages 23–38. 1991.

[DAVI86] T. R. Davies and S. J. Russell. "**A logical approach to reasoning by analogy**". In *Proc. 10th International Joint Conference on Artificial Intelligence*, pages 264–270, 1986.

[DAVI87] L. Davis. *Genetic algorithms and simulated annealing.* Pitman, 1987.

[DAWK89] R. Dawkins. *The selfish gene.* Oxford University Press, 1989.

[DEJO75] K. DeJong. *The Analysis and behaviour of a Class of Genetic Algorithms.* PhD thesis, University of Michigan, 1975.

[DEJO86] G. F. DeJong and R. Mooney. "**Explanation-based learning: An alternative view**". *Machine Learning*, 1(2):145–176, 1986.

[DEJO93] K.A. De Jong, W.M. Spears, and D.F. Gordon. "**Using Genetic Algorithms for Concept Learning**". *Machine Learning*, 13(2/3):161–188, 1993.

[DIET79] T. Dietterich. "**The methodology of knowledge layers for inducing descriptions of sequentially ordered events**". Master's thesis, University of Illinois, Urbana, 1979.

[DIET81] T. G. Dietterich and R. S. Michalski. "**Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods**". *Artificial Intelligence*, 16:257–294, 1981.

[DLLC85] J. Carreras i Martí, editor. *Diccionari de la Llengua Catalana.* Enciclopèdia Catalana, Barcelona, 1985.

[DUBE88] R. Dubes and A. Jain. *Algorithms for Clustering Data.* Prentice-Hall, Englewood Cliffs, USA, 1988.

[DUDA73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[ELLM89] T. Ellman. "**Explanation-based Learning: A survey of programs and perspectives**". *Computing Surveys*, 21:163–222, 1989.

[ESHE89] L.J.Eshelman, R.Caruna, and J.D.Schaffer. "**Biases in the crossover landscape**". In J.D.Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[EVAN68] T. Evans. "**A heuristic program to solve geometric analogy**". In M. Minsky, editor, *Semantic Information Processing.* MIT Press, Cambridge, Massachusets, 1968.

[FEIG61] E. Feigenbaum. "**The simulation of verbal learning behavior**". In *Western Joint Computer Conference*, pages 121–132, 1961.

[FEIG84] E. A. Feigenbaum and H. A. Simon. "**EPAM-like models of recognition and learning**". *Cognitive Science*, 8:305–336, 1984.

[FIKE72] R. E. Fikes, P. E. Hart, and N. J. Nilsson. "**Learning and executing generalized robot plans**". *Artificial Intelligence*, 3:251–288, 1972.

[FISH87] D. H. Fisher. "**Knowledge Acquisition Via Incremental Conceptual Clustering**". *Machine Learning*, 2:139–172, 1987.

[FISH91] D. Fisher and M. Pazzani. *Concept Formation: Knowledge and Experience on unsupervised learning*, chapter Computational models of concept learning, pages 3–43. Morgan Kaufmann Publisers, Inc., 1991.

[FISH92] D. Fisher, L. Xu, and N. Zard. "**Ordering Effects in Clustering**". In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 163–168, 1992.

[FORR93a] S. Forrest and M. Mitchel. "**What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation**". *Machine Learning*, 13(2/3):285–319, 1993.

[FORR93b] S. Forrest and M. Mitchell. "**Relative Building-Block Fitness and the Building-Block Hypothesis**". Technical report, Santa Fe Institute, 1993.

[FUKU75] K. Fukushima. "**Cognitron: a self-organizing multilayered neural network**". *Biological Cybernetics*, 20:121–136, 1975.

[GADN88] H. Gadner. *La nueva ciencia de la mente: historia de la revolución cognitiva.* Cognición y desarrollo humano. Ediciones Paidós, 1988.

[GALL91] C. Gallistel, A. Brown, S. Carey, R. Gelman, and F. Keil. "**Lessons from animal learning for the study of cognitive development**". In S. Carey and R. Gelman, editors, *The epigenesis of mind: essays on biology and cognition*, pages 3–36. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1991.

[GEMA84] S. Geman and D. Geman. "**Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images**". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

[GENA89] J.H. Genari, P. Langley, and D. Fisher. "**Models of incremental concept formation**". *Applied Artificial Intelligence*, 40:11–61, 1989.

[GLUC85] M. A. Gluck and J. E. Corter. "**Information, uncertainty and the utility of categories**". In Lawrence Erlbaum Associates, editor, *Seventh Annual Conference of the Cognitive Science Society. Irvine, CA.*, pages 283–287, 1985.

[GOLD89] D.E.Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[GROS76] S. Grossberg. "**Adaptive Pattern Classification and Universal Recoding I & II**". *Biological Cybernetics*, 23:187–202, 1976.

[GROS87] S. Grossberg. *The adaptive brain.* Nprth-Holland, 1987.

[HAMM88] K.J. Hammond. "**Case-based Planning**". In *Proc. CBR Workshop.* DARPA, 1988.

[HAMM89] K.J. Hammond. *Case-Based Planning. Viewing Planning as a Memory Task*, volume 1 of *Perspectives in Artificial Intelligence.* Academic Press, Inc., 1989.

[HANS86] S. J. Hanson and M. Bauer. "**Conceptual Clustering , Categorization and Polymorphy**". *Machine Learning*, 3:343–372, 1986.

[HANS90] S. J. Hanson. "**Conceptual Clustering and Categorization: Bridging the Gap Between Induction and Causal Models**". In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach (Volume III)*, pages 235–268. Kaufmann, San Mateo, CA, 1990.

[HAYE77] F. Hayes-Roth and J. McDermott. "**Knowledge Acquisition from Structural Descriptions**". In *Proceedings of the IJCAI*, pages 356–362, Cambridge, Mass., 1977.

[HAYE78] F. Hayes-Roth and J. McDermott. "**An Interference Matching Technique for Inducing Abstractions**". *Communications of the ACM*, 21(5), 1978.

[HAYE84] F. Hayes-Roth. "**The knowledge-based expert system: a tutorial**". *Computer*, (17 (9)), 1984.

[HEBB49] D. O. Hebb. *The Organization of Behaviour.* John Wiley and Sons, New York, 1949.

[HECH87] R. Hecht-Nielsen. "**Counterpropagation Networks**". *Applied Optics*, 26:4979–4984, 1987.

[HINT81] G. E. Hinton. "**Implementing Semantic Networks in Parallel Hardware**". In G. E. Hinton and J. A. Anderson, editors, *Parallel Models of Associative Memory.* Erlbaum, Hillsdale, NJ, 1981.

[HOLL92] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, 1992.

[HOPF82] J. J. Hopfield. "**Neural networks and physical systems with emergent collective computational abilities**". In *Proc. Nat. Acad. Sci. USA, Vol 79, pp 2554-2558*, 1982.

[HOPF83] J. J. Hopfield, D. I. Feinstein, and R. G. Palmer. "**"Unlearning" Has a Stabilizing Effect in Collective Memories**". *Nature*, 304:158–159, 1983.

[HOPF84] J. Hopfield. "**Neurons with graded responses have collective computational properties like those of two-state neurons**". In *Procs. of the national Academy of Science*, 1984.

[HOPF85] J. J. Hopfield and D. W. Tank. "**Neural Computation of Decisions in Optimization Problems**". *Biological Cybernetics, vol. 52, pp 147-152*, 1985.

[JANI93] C.Z. Janikow. "**A Knowledge-Intensive Genetic Algorithm for Supervised Learning**". *Machine Learnig*, 13(2/3):189–228, 1993.

[JONE83] G. V. Jones. "**Identifying Basic Categories**". *Psychological Bulletin*, 94(3):423–428, 1983.

[KEDA88] S. Kedar-Cabelli. *Analogical Reasoning*, chapter Analogy - From a unified perspective. Kluwer Academic Press, 1988.

[KOHO77] T. Kohonen. *Associative memory: a system-theoretical approach.* Springer, 1977.

[KOHO88] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1988.

[KOLO83] J. L. Kolodner. "**Reconstructive memory: A computer model**". *Cognitive Science*, 7:281–328, 1983.

[KOLO85] J.L. Kolodner. "**A Process Model of Case-based Reasoning in Problem Solving**". In *Proc. IJCAI*, 1985.

[KOLO87] J.L. Kolodner. "**Extending Problem Solver Capabilities Through Case-based Inference**". In *Proc. Fourth IWML*, 1987.

[KOLO88] J.L. Kolodner. "**Retrieving Events form a Case Memory: A Parallel Implementation**". In *Proc. CBR Workshop*. DARPA, 1988.

[KOLO91] J. L. Kolodner. "**Case-Based Reasoning and Learning. Introduction to CBR Methods**". ACAI91, Bilbao, 1991.

[KOLO93a] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Pub. Inc., 1993.

[KOLO93b] J. Kolodner. "**Special Issue on Case-Based Reasoning**". *Machine Learning*, 10(3):195–363, 1993.

[KOSK87a] B. Kosko. "**Bi-directional associative memories**". *IEEE Transactions on Systems, Man and Cybernetics*, 1987.

[KOSK87b] B. Kosko. "**Constructing an associative memory**". *Byte*, September 1987.

[KOSK92] B. Kosko. *Neural networks and fuzzy systems*. Prentice-Hall, 1992.

[KOTO88a] P. Koton. "**A Medical Reasoning Program that Improves with Experience**". In *Proc. SCAMC (IEEE)*, 1988.

[KOTO88b] P. Koton. "**Reasoning About Evidence in Causal Explanation**". In *Proc. AAAI*, 1988.

[KOTO88c] P. Koton. "**Using a Case Memory to Integrate Case-based and Causal Reasoning**". In *Proc. Tenth Conference of the Cognitive Science Society*, 1988.

[KOZA92] J.R. Koza. *Genetic Programming*. The MIT Press, 1992.

[KUHN78] T.S. Kuhn. "**Segundos Pensamientos sobre Paradigmas**". In *Segundos Pensamientos sobre Paradigmas*. Tecnos, Madrid, 1978.

[LAIR86a] J. Laird, A. Newell, and P. Rosenbloom. "**SOAR: An architecture for general intelligence**". Technical report, Department of Computer Science, Carnegie-Mellon University, 12 1986.

[LAIR86b] J. Laird, P. Rosenbloom, and A. Newell. *Universal subgoaling and chunking*. Kluwer Academic Publishers, 1986.

[LAIR90] P.D. Laird. "**A survey on computational learning theory**". In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence: a Sourcebook*, pages 173–215. North-Holland, 1990.

[LAKO87] G. Lakoff. *Women, Fire and Dangerous Things*. The University of Chicago Press, 1987.

[LANG84] P. Langley, G. l. Bradshaw, and H. A. Simon. "**Rediscovering Chemistry with the BACON System**". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 307–329. Springer, Berlin, Heidelberg, 1984.

[LEBO86] M. Lebowitz. "**Integrated Learning: Controlling Explanation**". *Cognitive Science*, 10(2):219–240, 1986.

[LEBO87] M. Lebowitz. "**Experiments with incremental concept formation: UNIMEM**". *Machine Learning*, 2:103–138, 1987.

[LENA79] D. B. Lenat, F. Hayes-Roth, and P. Klahr. "**Cognitive Economy in Artificial Intelligence**". In *Proc. IJCAI*, 1979.

[LENA84] D. B. Lenat. "**The Role of Heuristics in Learning by Discovery: Three Case Studies**". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 243–306. Springer, Berlin, Heidelberg, 1984.

[LITT88] N. Littlestone. "**Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm**". *Machine Learning*, 2:285–318, 1988.

[LOPE91] R. López de Mántaras. *A distance-based attribute selection measure for decision tree induction*. Machine Learning. Kluwer Academic, 1991.

[LOPE93a] B. López. *Aprenentatge de plans per a sistemes experts*. PhD thesis, Universitat Politècnica de Catalunya, Facultat d' Informàtica de Barcelona, 1993.

[LOPE93b] B. López and E. Plaza. "**Case-Based Planning for Medical Diagnosis**". In J. Komorowski and Z.W. Raś, editors, *Methodologies for Intelligent Systems*, pages 96–105, University of Trodheim, Norway, 1993. Springer-Verlag. Proceedings ISMIS'93.

[MALL89] R.S. Mallory. "**Sources of Classification Accuracy in PROTOS**". Technical Report AI89-118, AI Lab, The University of Texas at Austin, December 1989.

[MALS73] C. von der Malsburg. "**Self-organization of orientation sensitive cells in striate cortex**". *Kybernetik*, 14:85–100, 1973.

[MANY94] F. Manyà. "**Notes de Lògica**". Technical report, Departament d' Informàtica i Enginyeria Industrial, Universitat de Lleida, 1994.

[MART91] M. Martín. "**LINNEO: Eina per l' ajut en la construcció de bases de coneixements en dominis poc estructurats**". Master's thesis, Facultat d' Informàtica de Barcelona, 1991.

[MEDI89] D. Medin. "**Concepts and Conceptual Structure**". *American Psycologist*, pages 1469–1481, 1989.

[MERV81] C. Mervis and E. Rosch. "**Categorization of natural objects**". *Annual review of Psicology*, (32):89–115, 1981.

[MICH80a] R. S. Michalski. "**Knowledge acquisition through conceptual clustering: A theoretical framework and algorithm for partitioning data into conjuctive concepts**". *International Journal of Policy Analysis and Information Systems*, 4:219–243, 1980.

[MICH80b] R. S. Michalski. "**Pattern Recognition as Rule-Guided Inductive Inference**". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):349–361, 1980.

[MICH84a] R. Michalski and R. E. Steep. "**A Theory and Methodology of Inductive Learning**". In J. Carbonell, editor, *Machine Learning: An Artificial Intelligence Approach*, chapter 11, pages 331–363. Ed. Tioga, Palo Alto, California, Ed. Tioga, Palo Alto, California, 1984.

[MICH84b] R. S. Michalski and R. E. Stepp. "**Learning from Observation: Conceptual Clustering**". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Springer, Berlin, Heidelberg, 1984.

[MICH86] R. Michalski and R. E. Steep. "**Conceptual Clustering: Inventing goal oriented classifications of structured objects**". In J. Carbonell, editor, *Machine Learning: An Artificial Intelligence Approach II*, chapter 11. Ed. Tioga, Palo Alto, California, 1986.

[MICH93] R. Michalski. "**A unifying theoretical framework for machine learning and methods for synthetic and multistrategy learning**". In *Fifth Advanced Course on Artificial Intelligence*. Capri, Italy, 1993.

[MINS54] M. Minsky. *Theory of Neural-Analog Reinforcement Systems and Its Aplication to the Brain-Model Problem*. PhD thesis, Princeton University, 1954.

[MINS67] M. Minsky. *Computation. Finite and infinite machines*. Prentice Hall, 1967.

[MINS69a] M. Minsky. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, Massachusets, 1969.

[MINS69b] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[MINT88] S. Minton. *Learning effective search control knowledge: An explanation-based approach*. PhD thesis, Carnegie Mellon, Computer Science Department, 3 1988.

[MINT89] S. Minton, J. Carbonell, C. Knoblock, D. Koukka, O. Etzioni, and Y.Gil. "**Explanation based learning: a problem solving perspective**". *Artificial Intelligence*, 40:63–118, 1989.

[MINT90] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. "**Explanation-based learning: A problem solving perspective**". In *Machine Learning. Paradigms and methods.*, pages 63–118. J. Carbonell, Ed., 1990.

[MITC82] T. M. Mitchell. "**Generalization as Search**". *Artificial Intelligence*, 18:203–226, 1982.

[MITC83] T. Mitchell, P. Utgoff, and R. Banerji. "**Learning by experimentation: Acquiring and refining problem-solving heuristics**". In *Machine Learning: An Artificial Intelligence Approach*, pages 163–190. R.S. Michalski, J.G. Carbonell and T.M. Mitchell, 1983.

[MITC86] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. "**Explanation-based learning: A unifying view**". *Machine Learning*, 1(1):47–80, 1986.

[MITC90] T.M. Mitchell, J. Allen, P. Chalosair, I. Cheng, O. Etzioni, and M. Ringuette. "**Theo: A Framework for Self-improving Systems**". In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990.

[MITC92] M. Mitchell, S. Forrest, and J.H. Holland. "**The Royal Road for Genetic Algorithms: Fitness Landscape and GA Performance**". In *Proceeding of the First European Conference on Artificial Life*, 1992.

[MORE92] A. Moreno. "**Generalización de fórmulas lógicas y su aplicación al aprendizaje automático**". Master's thesis, Facultat d' Informàtica de Barcelona, 1992.

[MOST83] D. J. Mostow. "**Machine transformation of advice into a heuristic search procedure**". In *Proc. International Workshop on Machine Learning*, pages 110–116, 1983.

[MURP82] G. L. Murphy. "**Cue Validity and Levels of Categorization**". *Psychological Bulletin*, 91(1):174–177, 1982.

[NATA91] B.K. Natarajan. *Machine Learning: a Theoretical Approach*. Morgan Kauffman, 1991.

[NEWE72] A. Newell and H.A.Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[NILS80] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA, 1980.

[NUÑE91a] G. Núñez. *Caracterización no monótona de la inferencia inductiva y su aplicación al aprendizaje basado en similitudes*. Phd thesis, FIB, UPC, 1991.

[NÚÑE91b] G. Núñez, M. Alvarado, U. Cortés, and Ll. Belanche. "**About the attribute relevance's nature**". In *Procs. of the TECCOMP 91*, 1991.

[PEAR87] J. Pearce. *An introduction to animal cognition*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

[PITT43] W. S. McCulloch and W. H. Pitts. "**A Logical Calculus of the Ideas Immanent in Nervous Activity**". *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[PITT90] L. Pitt and M.K. Warmuth. "**Prediction preserving reducibility**". *Journal of Computer and System Sciences*, 41:430–467, 1990.

[PLAZ92] E. Plaza. "**Tendencias en Inteligencia Artificial; hacia la cuarta década**". In A. del Moral, editor, *Nuevas tendencias en Inteligencia Artificial*, pages 379–415. U. Deusto, 1992.

[PLAZ93] E. Plaza and J. L. Arcos. "**Reflection and Analogy in Memory-based Learning**". In *Workshop on Multistrategy Learning*, pages 42–49, 1993.

[PORT86] B.W. Porter and R.E. Bareiss. "**PROTOS: An Experiment in Knowledge Acquisition for Heuristic Classification Tasks**". Technical Report AI TR86-35, The University of Texas at Austin, Artificial Intelligence Laboratory, 1986.

[QUIN79] J.R. Quinlan. *Discovering rules from large collections of examples: a case study.* Edinburgh University Press, 1979.

[QUIN86] J. R. Quinlan. "**Induction of decision trees**". *Machine Learning*, 1:81–106, 1986.

[RIES83] C. K. Riesbeck. "**Knowledge reorganization and reasoning style**". Technical Report 270, Department of Computer Science. Yale University. New Haven, Conn., 1983.

[RIES89] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Publishers, 1989.

[ROSE53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "**Equation of State Calculations for Fast Computing Machines**". *Journal of Chemistry and Physics*, 21:1087–1091, 1953.

[ROSE61] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington DC, 1961.

[ROSE86] P. S. Rosenbloom and A. Newell. "**The chunking of goal hierarchies: A generalized model of practice**". In *Machine Learning: An Artificial Intelligence Approach, Vol II*, pages 247–288. R.S. Michalski, J.G. Carbonell and T.M. Mitchell Eds. Morgan Kaufmann, Los Altos, California, 1986.

[ROSS90] B.H. Ross, S.J. Perkins, and P.L. Tenpenny. "**Reminding-based Category Learning**". *Cognitive Psycology*, 22:460–492, 1990.

[ROUR94] J. Roure. "**Study of methods and heuristics to improve the fuzzy classifications of LINNEO$^+$**". Master's thesis, Facultat d' Informática de Barcelona Universitat Politècnica de Catalunya, 1994.

[RUBI77] S. Rubin and R. Reddy. "**The locus mode of search and its use in image interpretation**". In *Proceedings of the 5th IJCAI*, pages 281–287, Cambridge, Mass., 1977.

[RUME86a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "**Learning Internal Representations by Error Propagation**". In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: explorations in the microstructure of cognition; vol. 1: Foundations.* The MIT Press, Cambridge, Massachusetts, 1986.

[RUME86b] David Rumelhart and James McClelland. "**On Learning the Past Tenses of English Verbs**". In J. McClelland and D. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition; Vol. 2: Psychological and Biological Models*. MIT Press, Cambridge, Mass., 1986.

[RUME86c] D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: exploring the microstructure of cognition*. MIT Press, Cambridge, Massachusets, 1986.

[RUSS86] S. J. Russell. "**Preliminary steps toward the automation of induction**". In *Proc. 7th National Conference on Artificial Intelligence*, pages 477–484, 1986.

[SÁNC89] E. Sánchez. "**Importance in knowledge-based systems**". *Information Systems*, (14 (6)):454–464, 1989.

[SCHA82] R.C. Schank. *Dynamic Memory. A Theory of Remindings and Learning in Computers and People*. Cambridge University Press, 1982.

[SCHI89] A.D. Schielmann and N.M. Acioly. "**Mathematical Knowledge Developed at Work: The Contribution of Practice Versus the Contribution of Schooling**". *Cognition and Instruction*, 6(3):185–221, 1989.

[SCHL86] D. Fisher J. Schlimmer. "**A case study of incremental concept induction**". In *Procs. of the fifth nat. conf. on artificial intelligence*, 1986.

[SEJN86] T. J. Sejnowski and C. Rosenberg. *NETtalk: A Parallel Network that Learns to Read Aloud*. Johns Hopkins University, 1986.

[SHAP82] E.Y. Shapiro. *Algorithmic Program Debugging*. PhD thesis, Yale University, 1982.

[SHOB88] D. L. Medin and E. J. Shoben. "**Context and structure in conceptual combination**". *Cognitive Psichology*, (20):158–190, 1988.

[SILV83] B. Silver. "**Learning equation solving methods from worked examples**". In *Proc. International Machine Learning Workshop*, 1983.

[SIMO89] H. Simon. "**21st Carnegie-Mellon Symposium on Cognition.**". In P.Klarh and K.Kotovsky., editors, *Complex Information Processing. The impact of H. Simon.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[SMIT81] E. Smith and D. Medin. *Categories and Concepts*. Harvard University Press. Cambridge Massachusetts., 1981.

[SUSS75] G. J. Sussman. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975.

[SUTT84] R.S Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

[SUTT88] R.S. Sutton. "**Learning to Predict by the Methods of Temporal Differences**". *Machine Learning*, 3(1):9–44, 1988.

[SYCA91] K.P. Sycara. "**Case-Based Reasoning**". European Summer School on Machine Learning, ES2ML-91, Priory Corsendonk, Belgium, 1991.

[SYSV89] G.Sysverda. "**Uniform Crossover in genetic algorithms**". In J.D.Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[TANK86] D. W. Tank and J. J. Hopfield. "**Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit**". *IEEE Transactions on Circuits and Systems*, 33:533–541, 1986.

[THOM91] K. Thompson and P. Langley. *Concept Formation: Knowledge and Experience on unsupervised learning*, chapter Concept formation in structured domains, pages 127–161. Morgan Kaufmann Publisers, Inc., 1991.

[THOM93] K. Thompson and K. McKusick. *COBWEB/3: A portable implementation*. Sterling software/AI research branch. NASA ARC, Mail Stop 269-2 Moffett Field, CA 94035 USA, 1.4 edition, July 1993.

[THRU91] S.B. Thrun et alt. "**The MONK's problems. A performance comparison of different learning algorithms**". Technical report, Carnegie Mellon University, 1991.

[UTGO90] P. Utgoff. *Incremental learning of decision trees*. Machine Learning. Kluwer Academic, 1990.

[VALI84] L.G. Valiant. "**A theory of the learnable**". *Communications of the ACM*, 27:1134–1142, 1984.

[VELD90] W. van de Velde. "**Incremental induction of topologically minimal trees**". In *Procs. of the seventh IJCAI*, 1990.

[VELO92] M.M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992.

[VERE75] S. A. Vere. "**Induction of Concepts in the Predicate Calculus**". In *Proceedings of the IJCAI*, pages 281–287, 1975.

[VERE77] S. A. Vere. "**Induction of Relational Productions in the Presence of Background Information**". In *Proceedings of the 5th IJCAI*, pages 349–355, Cambridge, Mass., 1977.

[VERE78] S. Vere. "**Inductive learning of relational productions**". In D. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*. Academic Press, New York, 1978.

[VERE80] S. Vere. "**Multilevel counterfactuals for generalizations of relational concepts and productions**". *Artificial Intelligence*, 14:138–164, 1980.

[VOSE91] M.Vose and G.Liepins. "**Schema disruption**". In Morgan Kaufmann, editor, *Proceedings of the Fourth International Conference on Genetic Programming*, pages 237–242, 1991.

[WALD77] R. Waldinger. "**Achieving several goals simultaneously**". In *Machine Intelligence 8*, pages 163–190. E. Elcock and D. Michie Eds. Ellis Horwood, London, 1977.

[WARM89] M.K. Warmuth. "**Towards representation independence in PAC learning**". In *Proc. Intl. Workshop on Analogical and Inductive Inference AII-89*, number 397 in Lecture Notes on Artificial Intelligence, pages 78–103. Springer-Verlag, 1989.

[WATA94] O. Watanabe. "**A formal study of learning via queries**". *Mathematical Systems Theory*, 27:211–229, 1994.

[WATE70] D. Waterman. "**Generalization learning techniques for automating the learning of heuristics**". *Artificial Intelligence*, 1:121–170, 1970.

[WATK89] C.J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989.

[WHIT93] D. Whitley. "**A Genetic Algorithm Tutorial**". Technical Report CS-93-103, Computer Science Department. Colorado State University, November 1993.

[WIDR60] B. Widrow and M. E. Hoff. "**Adaptive Switching Circuits**". *IRE WESCON convention record, parte 4*, pages 96–104, 1960.

[WILK88] D. C. Wilkins. "**Knowledge base refinement using apprenticeship learning techniques**". In *Proc. 7th National Conference on Artificial Intelligence*, pages 646–651, 1988.

[WILL69] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. "**Non-holographic Associative Memory**", 1969.

[WINS70] P. H. Winston. *Learning Structural Descriptions from Examples.* Phd thesis, MIT, Project MAC, Cambridge, Massachusetts, January 1970.

[WINS75] P. H. Winston. "**Learning structural descriptions from examples**". In P. H. Winston, editor, *The psychology of computer vision.* McGraw Hill, New York, 1975.

[WINS82] P.H. Winston. "**Learning new principles from precedents and exercises**". *Artificial Intelligence*, 19:321–350, 1982.

[WINS92] P. H. Winston, editor. *Artificial Intelligence: Third Edition.* Addison-Wesley, Reading, MA, 1992.

[ZHON92] S. Zhongzhi. *Principles of Machine Learning.* International Academic Publishers, 1992.

Índice